

Stephen Wismath
Alexander Wolff (Eds.)

LNCS 8242

Graph Drawing

21st International Symposium, GD 2013
Bordeaux, France, September 2013
Revised Selected Papers



 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Stephen Wismath Alexander Wolff (Eds.)

Graph Drawing

21st International Symposium, GD 2013
Bordeaux, France, September 23-25, 2013
Revised Selected Papers



Springer

Volume Editors

Stephen Wismath
University of Lethbridge
Department of Mathematics
and Computer Science
4401 University Dr.
Lethbridge, AB T1K-3M4, Canada
E-mail: wismath@uleth.ca

Alexander Wolff
Universität Würzburg
Institut für Informatik
Am Hubland
97074 Würzburg, Germany

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-319-03840-7

e-ISBN 978-3-319-03841-4

DOI 10.1007/978-3-319-03841-4

Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013954558

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

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

© Springer International Publishing Switzerland 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers that were presented at the 21st International Symposium on Graph Drawing, which was held during September 23–25, 2013, in Bordeaux, France. The symposium was hosted by the University of Bordeaux I and was attended by 99 participants from 15 countries. We thank David Auber, the local arrangements chair, and his team for their warm hospitality and their effort to keep the conference fee low. Everyone enjoyed the nice weather, the beauty of Bordeaux, and the excellent cuisine.

As was the case for GD 2012, paper submissions were partitioned into two tracks and there was a separate poster track. Track 1 deals with combinatorial and algorithmic aspects; track 2 with visualization systems and interfaces. There was a record of 110 submissions: 94 papers—79 long (with 12 pages) and 15 short (with 6 pages)—and 16 posters (each with a two-page description). Each submission was reviewed by at least three Program Committee members. The committee decided to accept 42 papers and 12 posters. The acceptance rates were 36/72 in track 1, 6/22 in track 2, and 12/16 for posters. Only one short paper was accepted (in track 1). We thank the Program Committee, the sub-reviewers, the session chairs, and all authors for their hard work.

As a novelty, GD 2013 had *three* invited talks, one from applications, one from theory, and one from industry. Tamara Munzner from the University of British Columbia had a look at graph drawing through the lens of a framework for analyzing visualization methods. Emo Welzl from ETH Zurich showed us how to count crossing-free geometric graphs. And Joe Marks, co-founder and CTO of Upfront Analytics Ltd., analyzed the value of research in an industrial environment. We thank the speakers for their excellent talks, which complemented each other nicely and were very well received.

As another novelty, GD 2013 awarded prizes for the best presentations. The conference participants who stayed until the last talk voted for the winners. Michael Bannister from the University of California at Irvine received the largest number of votes and hence the first prize for his talk on the paper “Superpatterns and Universal Point Sets.” Maarten Löffler from Utrecht University received the second prize for his talk “Colored Spanning Graphs for Set Visualization.”

Following a well-established tradition, the 20th Annual Graph Drawing contest was held during the conference. It had two main categories: an off-line and an on-line challenge. This year’s contest committee was chaired by Carsten Gutwenger (University of Dortmund). We thank the committee for preparing challenging problems and problem instances. A report of the contest is included in these proceedings.

We also wish to thank our sponsors; “diamond” sponsor Microsoft, “gold” sponsor Tom Sawyer Software, “silver” sponsor Vis4, “bronze” sponsor yWorks, and the local sponsors Région Aquitaine, CNRS, LaBRI, Université Bordeaux I,

and LACUB. Without their support, the registration fees would have been roughly twice as high.

We thank Philipp Kindermann from the University of Würzburg for his technical support in producing these proceedings.

The 22nd International Symposium on Graph Drawing (GD 2014) will be held in Würzburg, Germany, September 24–26, 2014. Christian Duncan and Antonios Symvonis will be the program committee chairs; Alexander Wolff will be the organizing committee chair.

September 2013

Stephen Wismath
Alexander Wolff

Organization

Program Committee

Therese Biedl	University of Waterloo, Canada
Ulrik Brandes	Universität Konstanz, Germany
Emilio Di Giacomo	Università degli Studi di Perugia, Italy
Vida Dujmović	University of Ottawa, Canada
William S. Evans	University of British Columbia, Canada
Stefan Felsner	TU Berlin, Germany
Fabrizio Frati	University of Sydney, Australia
Yifan Hu	AT&T Labs Research, USA
Christophe Hurter	École Nationale de l'Aviation Civile, France
Andreas Kerren	Linnéuniversitetet, Sweden
William J. Lenhart	Williams College, USA
Martin Nöllenburg	KIT, Germany
János Pach	EPFL, Switzerland
Helen Purchase	University of Glasgow, UK
Falk Schreiber	Universität Halle, Germany
Andreas Spillner	Universität Greifswald, Germany
Antonios Symvonis	National Technical University of Athens, Greece
Alexandru Telea	Universiteit Groningen, The Netherlands
Csaba D. Tóth	CSUN and University of Calgary, Canada
Frank van Ham	IBM Software Group, The Netherlands
Sue Whitesides	University of Victoria, Canada
Steve Wismath (Co-chair)	University of Lethbridge, Canada
Alexander Wolff (Co-chair)	Universität Würzburg, Germany

Organizing Committee

David Auber (Chair)	LaBRI, Université de Bordeaux, France
Romain Bourqui	LaBRI, Université de Bordeaux, France
Guy Melanon	LaBRI, Université de Bordeaux, France
Bruno Pinaud	LaBRI, Université de Bordeaux, France

Graph Drawing Contest Committee





Christian Duncan	Quinnipiac University, USA
Carsten Gutwenger (Chair)	TU Dortmund, Germany
Lev Nachmanson	Microsoft Research, USA
Georg Sander	IBM Frankfurt, Germany

Additional Reviewers





Aerts, Nieke
Angelini, Patrizio
Asinowski, Andrei
Bachmaier, Christian
Bekos, Michael
Binucci, Carla
Bläsius, Thomas
Bose, Prosenjit
Cornelsen, Sabine
Czauderna, Tobias
Da Lozzo, Giordano
Di Battista, Giuseppe
Didimo, Walter
Dragicevic, Pierre
Dumitrescu, Adrian
Dwyer, Tim
Erten, Cesim
Fulek, Radoslav
Gansner, Emden
Gemsa, Andreas
Grilli, Luca
Hartmann, Anja
Hoffmann, Udo
Hong, Seok-Hee
Huron, Samuel
Joret, Gwenaël
Keszegh, Balázs
Klein, Karsten
Kleist, Linda

Langerman, Stefan
Lazard, Sylvain
Maheshwari, Anil
Mchedlidze, Tamara
Montecchiani, Fabrizio
Morin, Pat
Mustața, Irina-Mihaela
Niedermann, Benjamin
Nocaj, Arlind
Ortmann, Mark
Pampel, Barbara
Patrignani, Maurizio
Poon, Sheung-Hung
Pálvölgyi, Dömötör
Radoičić, Radoš
Roselli, Vincenzo
Rutter, Ignaz
Saeedi, Noushin
Schaefer, Marcus
Scheepens, Roeland
Schulz, Hans-Joerg
Sheffer, Adam
Tóth, Géza
Ueckerdt, Torsten
Verbeek, Kevin
Walny, Jagoda
Wood, David R.
Wybrow, Michael
Zimmer, Björn

Sponsors

Diamond Sponsor		Gold Sponsor	
 <h1>Microsoft</h1>		 <h1>Tom Sawyer[®]</h1> <p>S O F T W A R E</p>	
Silver Sponsor	Bronze Sponsor		
 <h1>Vis4</h1>	 <h1>yWorks</h1> <p><i>the diagramming company</i></p>		

Other Sponsors

 <p>cnrs dépasser les frontières</p>	 <p>UNIVERSITÉ BORDEAUX 1 Sciences Technologiques</p>	 <p>LaBRI</p>	 <p>RÉGION AQUITAINE COMMUNAUTÉ URBAINE DE BORDEAUX LA CUB www.lacub.fr</p>
---	--	--	---

Invited Talks

Graph Drawing through the Lens of a Framework for Analyzing Visualization Methods

Tamara Munzner

University of British Columbia, Department of Computer Science
Vancouver BC, Canada

tmm@cs.ubc.ca, <http://www.cs.ubc.ca/~tmm>

Abstract. The visualization community has drawn heavily on the algorithmic and systems-building work that has appeared with the graph drawing literature, and in turn has been a fertile source of applications. In the spirit of further promoting the effective transfer of ideas between our two communities, I will discuss a framework for analyzing the design of visualization systems. I will then analyze a range of graph drawing techniques through this lens. In the early stages of a project, this sort of analysis may benefit algorithm developers who seek to identify open problems to attack. In later project stages, it could guide algorithm developers in characterizing how newly developed layout methods connect with the tasks and goals of target users in different application domains.

The Counting of Crossing-Free Geometric Graphs — Algorithms and Combinatorics

Emo Welzl*

Institute for Theoretical Computer Science
ETH Zurich, CH-8092 Zurich, Switzerland
emo@inf.ethz.ch

Abstract. We are interested in the understanding of crossing-free geometric graphs—these are graphs with an embedding on a given planar point set where the edges are drawn as straight line segments without crossings. Often we are restricted to certain types of graphs, most prominently triangulations, but also spanning cycles, spanning trees, or (perfect) matchings (and crossing-free partitions), among others. A primary goal is to enumerate, to count, or to sample graphs of a certain type for a given point set—so these are algorithmic questions—, or to give estimates for the maximum and minimum number of such graphs on any set of n points—these are problems in extremal combinatorial geometry.

In this talk we will encounter some of the recent developments (since my GD'06 talk). Among others, I will show some of the new ideas for providing extremal estimates, e.g. for the number of crossing-free spanning cycles: the support-refined estimate for cycles versus triangulations, the use of pseudo-simultaneously flippable edges in triangulations, and the employment of Kasteleyn's beautiful linear algebra method for counting perfect matchings in planar graphs—here, interestingly, in a weighted version. Moreover, Raimund Seidel's recent 2^n -algorithm for counting triangulations is discussed, with its extensions by Manuel Wettstein to other types of graphs (e.g. crossing-free perfect matchings).

Keywords: computational geometry, geometric graphs, counting, sampling, enumeration.

* Supported by EuroCores/EuroGiga/ComPoSe SNF grant 20GG21_134318/1.

The Value of Research

Joe Marks

Upfront Analytics Ltd.

¹Marine Terrace, Dun Laoghaire, Co. Dublin, Ireland

Abstract. Which outcome would you prefer for your current research project: a paper with 500 citations, or a business worth \$5M? Or maybe it doesn't matter to you as long as you're doing interesting work and having fun. But even if academic and industrial researchers tend to value research differently, deciding what to work on is the first and fundamental step in any inquiry, and so it is worth thinking about the process and criteria that researchers use for project selection.

To get you thinking about this issue, I will present a selection of recent industrial projects with which I was involved as a researcher or manager. In the tradition of reality TV, you the audience will be invited to predict a 10-year citation count, estimate a dollar value, and assess general coolness for each of the projects presented. Active participation is encouraged! Although the focus is on research from the media and entertainment industry, hopefully you will come away with thoughts about how to estimate the value of your own research from both an academic and a commercial perspective. I will end with some speculative potential graph-drawing projects to which we can apply our just-practiced judgment on project selection.

Table of Contents

Upward Drawings

- On the Upward Planarity of Mixed Plane Graphs 1
Fabrizio Frati, Michael Kaufmann, János Pach, Csaba D. Tóth, and David R. Wood
- Upward Planarity Testing: A Computational Study 13
Markus Chimani and Robert Zeranski

Planarity

- Characterizing Planarity by the Splittable Deque 25
Christopher Auer, Franz J. Brandenburg, Andreas Gleißner, and Kathrin Hanauer
- Strip Planarity Testing 37
Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati
- Morphing Planar Graph Drawings Efficiently 49
Patrizio Angelini, Fabrizio Frati, Maurizio Patrignani, and Vincenzo Roselli

Invited Talk

- Graph Drawing through the Lens of a Framework for Analyzing Visualization Methods (Extended Abstract) 61
Tamara Munzner

Beyond Planarity

- A Linear-Time Algorithm for Testing Outer-1-Planarity 71
Seok-Hee Hong, Peter Eades, Naoki Katoh, Giuseppe Liotta, Pascal Schweitzer, and Yusuke Suzuki
- Straight-Line Grid Drawings of 3-Connected 1-Planar Graphs 83
Md. Jawaherul Alam, Franz J. Brandenburg, and Stephen G. Kobourov
- New Bounds on the Maximum Number of Edges in k -Quasi-Planar Graphs 95
Andrew Suk and Bartosz Walczak

Recognizing Outer 1-Planar Graphs in Linear Time 107
*Christopher Auer, Christian Bachmaier, Franz J. Brandenburg,
 Andreas Gleißner, Kathrin Hanauer, Daniel Neuwirth, and
 Josef Reislhuber*

Geometric Representations

Straight Line Triangle Representations 119
Nieke Aerts and Stefan Felsner

Extending Partial Representations of Circle Graphs 131
Steven Chaplick, Radoslav Fulek, and Pavel Klavík

On Balanced \dagger -Contact Representations 143
Stephane Durocher and Debajyoti Mondal

Strongly-Connected Outerplanar Graphs with Proper Touching
 Triangle Representations 155
J. Joseph Fowler

3D et al.

Achieving Good Angular Resolution in 3D Arc Diagrams 161
Michael T. Goodrich and Paweł Pszozna

A Duality Transform for Constructing Small Grid Embeddings of 3D
 Polytopes 173
Alexander Igamberdiev and André Schulz

Block Additivity of \mathbb{Z}_2 -Embeddings 185
Marcus Schaefer and Daniel Štefankovič

Universality

Exploiting Air-Pressure to Map Floorplans on Point Sets 196
Stefan Felsner

Superpatterns and Universal Point Sets 208
*Michael J. Bannister, Zhanpeng Cheng, William E. Devanny, and
 David Eppstein*

Simultaneous Embedding: Edge Orderings, Relative Positions,
 Cutvertices 220
Thomas Bläsius, Annette Karrer, and Ignaz Rutter

Practical Graph Drawing

Sketched Graph Drawing: A Lesson in Empirical Studies	232
<i>Helen C. Purchase</i>	
Many-to-One Boundary Labeling with Backbones	244
<i>Michael A. Bekos, Sabine Cornelsen, Martin Fink, Seok-Hee Hong, Michael Kaufmann, Martin Nöllenburg, Ignaz Rutter, and Antonios Symvonis</i>	
Streamed Graph Drawing and the File Maintenance Problem	256
<i>Michael T. Goodrich and Pawel Pszozna</i>	
COAST: A Convex Optimization Approach to Stress-Based Embedding	268
<i>Emden R. Gansner, Yifan Hu, and Shankar Krishnan</i>	

Subgraphs

Colored Spanning Graphs for Set Visualization	280
<i>Ferran Hurtado, Matias Korman, Marc van Kreveld, Maarten Löffler, Vera Sacristán, Rodrigo I. Silveira, and Bettina Speckmann</i>	
Drawing Non-Planar Graphs with Crossing-Free Subgraphs	292
<i>Patrizio Angelini, Carla Binucci, Giordano Da Lozzo, Walter Didimo, Luca Grilli, Fabrizio Montecchiani, Maurizio Patrignani, and Ioannis G. Tollis</i>	
Exploring Complex Drawings via Edge Stratification	304
<i>Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, Fabrizio Montecchiani, and Ioannis G. Tollis</i>	
Drawing Planar Graphs with a Prescribed Inner Face	316
<i>Tamara Mchedlidze, Martin Nöllenburg, and Ignaz Rutter</i>	

Crossings

Metro-Line Crossing Minimization: Hardness, Approximations, and Tractable Cases	328
<i>Martin Fink and Sergey Pupyrev</i>	
Fixed Parameter Tractability of Crossing Minimization of Almost-Trees	340
<i>Michael J. Bannister, David Eppstein, and Joseph A. Simons</i>	
Strict Confluent Drawing	352
<i>David Eppstein, Danny Holten, Maarten Löffler, Martin Nöllenburg, Bettina Speckmann, and Kevin Verbeek</i>	

Geometric Graphs and Geographic Networks

A Ramsey-Type Result for Geometric ℓ -Hypergraphs	364
<i>Dhruv Mubayi and Andrew Suk</i>	
Minimum Length Embedding of Planar Graphs at Fixed Vertex Locations	376
<i>Timothy M. Chan, Hella-Franziska Hoffmann, Stephen Kiazzyk, and Anna Lubiw</i>	
Stub Bundling and Confluent Spirals for Geographic Networks	388
<i>Arlind Nocaj and Ulrik Brandes</i>	

Angular Restrictions

On Orthogonally Convex Drawings of Plane Graphs (Extended Abstract)	400
<i>Yi-Jun Chang and Hsu-Chun Yen</i>	
Planar and Plane Slope Number of Partial 2-Trees	412
<i>William Lenhart, Giuseppe Liotta, Debajyoti Mondal, and Rahnuma Islam Nishat</i>	
Slanted Orthogonal Drawings	424
<i>Michael A. Bekos, Michael Kaufmann, Robert Krug, Stefan Näher, and Vincenzo Roselli</i>	

Grids

Drawing Arrangement Graphs in Small Grids, or How to Play Planarity	436
<i>David Eppstein</i>	
Incremental Grid-Like Layout Using Soft and Hard Constraints	448
<i>Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow</i>	
Using ILP/SAT to Determine Pathwidth, Visibility Representations, and other Grid-Based Graph Drawings	460
<i>Therese Biedl, Thomas Bläsius, Benjamin Niedermann, Martin Nöllenburg, Roman Prutkin, and Ignaz Rutter</i>	

Curves and Routes

Untangling Two Systems of Noncrossing Curves	472
<i>Jiří Matoušek, Eric Sedgwick, Martin Tancer, and Uli Wagner</i>	

Drawing Permutations with Few Corners	484
<i>Sergey Bereg, Alexander E. Holroyd, Lev Nachmanson, and Sergey Pupyrev</i>	

Dynamic Traceroute Visualization at Multiple Abstraction Levels	496
<i>Massimo Candela, Marco Di Bartolomeo, Giuseppe Di Battista, and Claudio Squarcella</i>	

Graph Drawing Contest

Graph Drawing Contest Report	508
<i>Christian A. Duncan, Carsten Gutwenger, Lev Nachmanson, and Georg Sander</i>	

Posters

3D Graph Printing in GLuskap	514
<i>Joel Bennett and Stephen Wismath</i>	

Optical Graph Recognition on a Mobile Device	516
<i>Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, and Josef Reislhuber</i>	

Browser-Based Graph Visualization of Dynamic Data with VisGraph . . .	518
<i>Jos de Jong and Giovanni E. Paziienza</i>	

Exact and Fixed-Parameter Algorithms for Metro-Line Crossing Minimization Problems	520
<i>Yoshio Okamoto, Yuichi Tatsu, and Yushi Uno</i>	

Convex-Arc Drawings of Pseudolines	522
<i>David Eppstein, Mereke van Garderen, Bettina Speckmann, and Torsten Ueckerdt</i>	

The Density of Classes of 1-Planar Graphs	524
<i>Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, Kathrin Hanauer, and Daniel Newwirth</i>	

BGPlay3D: Exploiting the Ribbon Representation to Show the Evolution of Interdomain Routing	526
<i>Patrizio Angelini, Lorenzo Antonetti Clarucci, Massimo Candela, Maurizio Patrignani, Massimo Rimondini, and Roberto Sepe</i>	

Ravenbrook Chart: A New Library for Graph Layout and Visualisation	528
<i>Nick Levine and Nick Barnes</i>	

Small Grid Embeddings of Prismatoids and the Platonic Solids	530
<i>Alexander Igamberdiev, Finn Nielsen, and André Schulz</i>	

The Graph Landscape – a Visualization of Graph Properties	532
<i>Peter Eades and Karsten Klein</i>	
Application of Graph Layout Algorithms for the Visualization of Biological Networks in 3D	534
<i>Tim Angus, Tom Freeman, and Karsten Klein</i>	
Plane Cubic Graphs and the Air-Pressure Method	536
<i>Stefan Felsner and Linda Kleist</i>	
Author Index	539

On the Upward Planarity of Mixed Plane Graphs^{*}

Fabrizio Frati¹, Michael Kaufmann², János Pach³,
Csaba D. Tóth⁴, and David R. Wood

¹ School of Information Technologies, The University of Sydney, Australia
brillo@it.usyd.edu.au

² Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany
mk@informatik.uni.tuebingen.de

³ EPFL, Lausanne, Switzerland and Rényi Institute, Budapest, Hungary
pach@cims.nyu.edu

⁴ California State University Northridge, USA and University of Calgary, Canada
cdtoth@acm.org

⁵ School of Mathematical Sciences, Monash University, Melbourne, Australia
david.wood@monash.edu

Abstract. A *mixed plane graph* is a plane graph whose edge set is partitioned into a set of directed edges and a set of undirected edges. An *orientation* of a mixed plane graph G is an assignment of directions to the undirected edges of G resulting in a directed plane graph \mathcal{G} . In this paper, we study the computational complexity of testing whether a given mixed plane graph G is *upward planar*, i.e., whether it admits an orientation resulting in a directed plane graph \mathcal{G} such that \mathcal{G} admits a planar drawing in which each edge is represented by a curve monotonically increasing in the y -direction according to its orientation.

Our contribution is threefold. First, we show that the upward planarity testing problem is solvable in cubic time for *mixed outerplane graphs*. Second, we show that the problem of testing the upward planarity of mixed plane graphs reduces in quadratic time to the problem of testing the upward planarity of *mixed plane triangulations*. Third, we exhibit linear-time testing algorithms for two classes of mixed plane triangulations, namely *mixed plane 3-trees* and mixed plane triangulations in which *the undirected edges induce a forest*.

1 Introduction

Upward planarity is the natural extension of planarity to directed graphs. When visualizing a directed graph, one usually requires an *upward drawing*, that is, a drawing in which the directed edges flow monotonically in the y -direction. A drawing is *upward planar* if it is planar and upward. Testing whether a directed graph G admits an upward planar drawing is NP-hard [9], however, it is polynomial-time solvable if G has a *fixed planar embedding* [2], if it has a *single-source* [3,13], if it is *outerplanar* [15], or if it is

^{*} Pach was supported by Hungarian Science Foundation EuroGIGA Grant OTKA NN 102029, by Swiss National Science Foundation Grants 200020-144531 and 200021-137574, and by NSF Grant CCF-08-30272. Tóth was supported in part by NSERC (RGPIN 35586) and NSF (CCF-0830734).

a *series-parallel graph* [7]. *Exponential-time algorithms* [1] and *FPT algorithms* [12] for upward planarity testing are known.

In this paper we deal with *mixed graphs*. A mixed graph is a graph whose edge set is partitioned into a set of directed edges and a set of undirected edges. Mixed graphs unify the expressive power of directed and undirected graphs, as they allow one to simultaneously represent hierarchical and non-hierarchical relationships. A number of problems on mixed graphs have been studied, e.g., *coloring mixed graphs* [11,17] and *orienting mixed graphs to satisfy connectivity requirements* [5,6].

Upward planarity generalizes to mixed graphs as follows. A drawing of a mixed graph is *upward planar* if it is planar, every undirected edge is a y -monotone curve, and every directed edge is an arc with monotonically increasing y -coordinates. Hence, testing the upward planarity of a mixed graph is equivalent to testing whether its undirected edges can be oriented to produce an upward planar directed graph. Since the upward planarity testing problem is NP-hard for directed graphs [9], it is NP-hard for mixed graphs as well. Binucci and Didimo [4] studied the problem of testing the upward planarity of *mixed plane graphs*, that is, of mixed graphs with a given plane embedding. They describe an ILP formulation for the problem and present experiments showing the efficiency of their solution. Different graph drawing questions on mixed graphs (related to crossing and bend minimization) have been studied in [8,10].

We show the following results.

In Section 3 we show that the upward planarity testing problem can be solved in $O(n^3)$ time for n -vertex *mixed outerplane graphs*. Our dynamic programming algorithm uses a characterization for the upward planarity of directed plane graphs due to Bertolazzi et al. [2], and it tests the upward planarity of a mixed outerplane graph G based on the upward planarity of two subgraphs of G .

In Section 4 we show that, for every n -vertex mixed plane graph G , there exists an $O(n^2)$ -vertex *mixed plane triangulation* G' such that G is upward planar if and only if G' is upward planar. As a consequence, the problem of testing the upward planarity of mixed plane graphs is polynomial-time solvable (NP-hard) if and only if the problem of testing the upward planarity of mixed plane triangulations is polynomial-time solvable (resp., NP-hard).

In Section 5, motivated by the previous result, we present linear-time algorithms to test the upward planarity of two classes of mixed plane triangulations, namely *mixed plane 3-trees* and mixed plane triangulations in which *the undirected edges induce a forest*. The former algorithm uses dynamic programming, while the latter algorithm uses induction on the number of undirected edges in the mixed plane triangulation.

Because of space limitations, some proofs are omitted or sketched in this extended abstract. Complete proofs are available in the full version of the paper.

2 Preliminaries

A planar drawing of a graph determines a circular ordering of the edges incident to each vertex. Two planar 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* and the bounded faces are the *internal faces*. An edge of G incident to the outer face (not incident to the outer face) is called *external* (resp., *internal*). Two planar drawings with the same planar embedding have the same faces. However, they could still differ in their outer faces. A *plane embedding* is a planar embedding together with a choice for the outer face. A *plane graph* is a graph with a given plane embedding. An *outerplane graph* is a plane graph whose vertices are all incident to the outer face. A *plane triangulation* is a plane graph whose faces are delimited by 3-cycles. An *outerplane triangulation* is an outerplane graph whose internal faces are delimited by 3-cycles.

A *block* of a graph $G(V, E)$ is a maximal (both in terms of vertices and in terms of edges) 2-connected subgraph of G ; in particular, an edge of G whose removal disconnects G is considered as a block of G . In this paper, when talking about the connectivity of mixed graphs or directed graphs, we always refer to the connectivity of their underlying undirected graphs.

A vertex v in a directed graph is a *sink* (*source*) if every edge incident to v is incoming at v (resp., outgoing at v). A vertex v in a directed plane graph is *bimodal* if the incoming edges at v are consecutive in the cyclic ordering of edges incident to v (which implies that the outgoing edges at v are also consecutive). A directed plane graph is *bimodal* if every vertex is bimodal. A vertex v in a 2-connected directed outerplane graph is a *sink-switch* (*source-switch*) if the two external edges incident to v are both incoming (resp., outgoing) at v .

Bertolazzi et al. [2] characterized the directed plane graphs that are upward planar. In this paper, we will use such a characterization when dealing with two specific classes of directed plane graphs, namely directed outerplane triangulations and directed plane triangulations. Thus, we state such a characterization directly for such graph classes.

Theorem 1 ([2]). *A directed outerplane triangulation G is upward planar if and only if it is acyclic, it is bimodal, and the number of sources plus the number of sinks in G equals the number of sink-switches (or source-switches) plus one.*

Theorem 2 ([2]). *A directed plane triangulation G is upward planar if and only if it is acyclic, it is bimodal, and G has exactly one source and one sink that are incident to the outer face of G .*

A mixed plane graph is upward planar if and only if each of its connected components is upward planar. Thus, without loss of generality, we only consider connected mixed plane graphs. In the following lemma, we show that a stronger condition can in fact be assumed for each considered plane graph G , namely that G is 2-connected.

Lemma 1. *Every n -vertex mixed plane graph G can be augmented with new edges and vertices to a 2-connected mixed plane graph G' with $O(n)$ vertices such that G is upward planar if and only if G' is. If G is outerplane, then G' is also outerplane. Moreover, G' can be constructed from G in $O(n)$ time.*

Proof Sketch: While G has a cutvertex c that is incident to a face f (if G is outerplane, then f is its outer face), we consider two edges (v_1, c) and (u_2, c) that are consecutively incident to c in G and that belong to different blocks of G . We add a vertex w inside f and connect it to v_1 and u_2 . The repetition of such an augmentation leads to a 2-connected mixed plane graph G' satisfying the conditions of the lemma. \square

3 Upward Planarity Testing for Mixed Outerplane Graphs

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

Theorem 3. *The upward planarity of an n -vertex mixed outerplane graph can be tested in $O(n^3)$ time.*

Let G be any n -vertex mixed outerplane graph. By Lemma 1, an $O(n)$ -vertex 2-connected mixed outerplane graph G^* can be constructed in $O(n)$ time such that G is upward planar if and only if G^* is.

We introduce some notation and terminology. Let u and v be distinct vertices of G^* . We denote by $G^* + (u, v)$ the graph obtained from G^* by adding edge (u, v) if it is not already in G^* , and by $G^* - u$ the graph obtained from G^* by deleting u and its incident edges. Consider an orientation \mathbf{G}^* of G^* . A vertex is *sinky* (*sourcey*) in \mathbf{G}^* if it is a sink-switch but not a sink (if it is a source-switch but not a source, resp.). A vertex that is neither a sink, a source, sinky, nor sourcey is *ordinary*; that is, v is ordinary if the two external edges incident to v are one incoming at v and one outgoing at v in \mathbf{G}^* . We say the *status* of a vertex of G^* in \mathbf{G}^* is sink, source, sinky, sourcey, or ordinary.

First note that G^* is upward planar if and only if there is an upward planar directed outerplane triangulation T of G^* , that is, if and only if G^* can be augmented to a mixed outerplane triangulation, and the undirected edges of such a triangulation can be oriented in such a way that the resulting directed outerplane triangulation T is upward planar. The approach of our algorithm is to determine if there is such a T using recursion. The algorithm can be easily modified to produce T if it exists.

We observe that a directed outerplane triangulation T is acyclic if and only if every 3-cycle in T is acyclic. One direction is trivial. Conversely, suppose that T contains a directed cycle. Let C be a shortest directed cycle of T . If C is a 3-cycle, then we are done. Otherwise, an edge $(x, y) \notin C$ exists in T between two vertices x and y both in C . Thus, $C + (x, y)$ contains two shorter cycles, one of which is a directed cycle, contradicting the choice of C . Hence, to ensure the acyclicity of a directed outerplane triangulation, it suffices to ensure that its internal faces are acyclic.

A *potential edge* of G^* is a pair of distinct vertices x and y in G^* such that $G^* + (x, y)$ is outerplane, which is equivalent to saying that x and y are incident to a common internal face of G^* (notice that an edge of G^* is a potential edge of G^*). Fix some external edge r of G^* , called the *root edge*. Let $e = \{x, y\}$ be an internal potential edge of G^* . Then $\{x, y\}$ separates G^* , that is, G^* contains two subgraphs G_1^* and G_2^* , such that $G^* = G_1^* \cup G_2^*$ and $V(G_1^* \cap G_2^*) = \{x, y\}$. (Thus, there is no edge between $G_1^* - x - y$ and $G_2^* - x - y$.) W.l.o.g., $r \in E(G_1^*)$. Let $G_e^* := G_2^* + (x, y)$. Observe that G_e^* is a 2-connected mixed outerplane graph with e incident to the outer face. Also, let $e = \{x, y\} \neq r$ be an external potential edge of G^* . Then, we define G_e^* to be the 2-vertex graph containing the single edge (x, y) . Further, let $G_r^* := G^*$. For any (internal or external) potential edge $e = \{x, y\}$ of G^* and for an orientation \overrightarrow{xy} of e , let $G_{\overrightarrow{xy}}^*$ be G_e^* with e oriented \overrightarrow{xy} . Define a partial order \prec on the potential edges of G^* as follows. For distinct potential edges e and f of G^* , say $e \prec f$ if both end-vertices of f are in G_e^* . Loosely speaking, $e \prec f$ if $G^* + e + f$ is outerplane and e is “between” r and f .

A *potential arc* of G^* is a potential edge that is assigned an orientation preserving its orientation in G^* . So if e is an undirected edge of G^* or a potential edge not in

G^* , then there are two potential arcs associated with e , while if e is a directed edge of G^* , then there is one potential arc associated with e . If a potential arc \overrightarrow{xy} is part of a triangulation T of G^* , then x is a source, sourcey, or ordinary, and y is a sink, sinky, or ordinary in $G_{\overrightarrow{xy}}^*$. We define the *status* of \overrightarrow{xy} in $G_{\overrightarrow{xy}}^*$ as an ordered pair S of $S(x) \in \{\text{source, sourcey, ordinary}\}$ and $S(y) \in \{\text{sink, sinky, ordinary}\}$.

We now define a function $\text{UP}(\overrightarrow{xy}, S)$, that takes as an input a potential arc \overrightarrow{xy} and a status S of \overrightarrow{xy} , and has value “true” if and only if there is an upward planar directed outerplane triangulation $T_{\overrightarrow{xy}}$ of $G_{\overrightarrow{xy}}^*$ that respects $S(x)$ and $S(y)$; notice that, if \overrightarrow{xy} is external and does not correspond to r , then $T_{\overrightarrow{xy}}$ is a single edge.

First, the values of $\text{UP}(\overrightarrow{xy}, S)$ can be computed in total $O(n)$ time for all the external potential arcs \overrightarrow{xy} of G^* not corresponding to r and for all statuses of \overrightarrow{xy} . Indeed, $\text{UP}(\overrightarrow{xy}, S)$ is true if and only if $S(x) = \text{source}$ and $S(y) = \text{sink}$.

We show below that, for each potential arc \overrightarrow{xy} in G^* that is internal or that is external and corresponds to r , and for each status S of \overrightarrow{xy} , the value of $\text{UP}(\overrightarrow{xy}, S)$ can be computed in $O(n)$ time from values associated to potential arcs corresponding to potential edges e with $\{x, y\} \prec e$. Since there are at most $n(n+1)$ potential arcs and nine statuses for each potential arc, all the values of $\text{UP}(\overrightarrow{xy}, S)$ can be computed in $O(n^3)$ time by dynamic programming in reverse order to a linear extension of \prec . Then, there is an upward planar directed outerplane triangulation of G^* if and only if $\text{UP}(\overrightarrow{xy}, S)$ is true for some orientation \overrightarrow{xy} of r and some status S of \overrightarrow{xy} .

Let \overrightarrow{xy} be a potential arc that is internal to G^* or that corresponds to r . Let S be a status of \overrightarrow{xy} . Suppose that $\text{UP}(\overrightarrow{xy}, S)$ is true. Then, there is an upward planar directed outerplane triangulation $T_{\overrightarrow{xy}}$ of $G_{\overrightarrow{xy}}^*$ that respects $S(x)$ and $S(y)$. Such a triangulation contains a vertex $z \in V(G_{xy}^*) - x - y$ such that (x, y, z) is an internal face of $T_{\overrightarrow{xy}}$. Since $T_{\overrightarrow{xy}}$ has edge (x, y) oriented from x to y , then edges (x, z) and (y, z) cannot be simultaneously incoming at x and outgoing at y , respectively, as otherwise $T_{\overrightarrow{xy}}$ would contain a directed cycle, which is not possible by Theorem 1. Hence, edges (x, z) and (y, z) in $T_{\overrightarrow{xy}}$ are either outgoing at x and incoming at y , or outgoing at x and outgoing at y , or incoming at x and incoming at y , respectively.

Now, for any status S of \overrightarrow{xy} and for a particular vertex $z \in V(G_{xy}^*) - x - y$, we characterize the conditions for which an upward planar directed outerplane triangulation $T_{\overrightarrow{xy}}$ exists that respects $S(x)$ and $S(y)$ and that contains edges (x, z) and (y, z) oriented according to each of the three orientations described above.

Lemma 2. *There is an upward planar directed outerplane triangulation $T_{\overrightarrow{xy}}$ that respects $S(x)$ and $S(y)$, that contains edge (x, z) outgoing at x , and that contains edge (z, y) incoming at y , if and only if \overrightarrow{xz} and \overrightarrow{zy} are potential arcs of G^* and there are statuses S_1 of \overrightarrow{xz} and S_2 of \overrightarrow{zy} such that the following conditions hold: (a) $S_1(x) = S(x) \in \{\text{source, sourcey, ordinary}\}$, (b) $S_2(y) = S(y) \in \{\text{sink, sinky, ordinary}\}$, (c) $S_1(z) \in \{\text{sink, ordinary}\}$, (d) $S_2(z) \in \{\text{source, ordinary}\}$, (e) $S_1(z) = \text{sink}$ or $S_2(z) = \text{source}$, and (f) both $\text{UP}(\overrightarrow{xz}, S_1)$ and $\text{UP}(\overrightarrow{zy}, S_2)$ are true.*

Proof: (\implies) Let $T_{\overrightarrow{xy}}$ be an upward planar directed outerplane triangulation of $G_{\overrightarrow{xy}}^*$ that respects $S(x)$ and $S(y)$, that contains edge (x, z) outgoing at x , and that contains edge (z, y) incoming at y . Then, \overrightarrow{xz} and \overrightarrow{zy} are potential arcs of G^* . Further, $T_{\overrightarrow{xy}}$ determines upward planar directed outerplane triangulations $T_{\overrightarrow{xz}}$ and $T_{\overrightarrow{zy}}$

respectively of $G_{\vec{xz}}^*$ and $G_{\vec{zy}}^*$ (where $T_{\vec{xz}}$ and $T_{\vec{zy}}$ are single edges if \vec{xz} and \vec{zy} are external, respectively), as well as statuses S_1 and S_2 of \vec{xz} and \vec{zy} , respectively, such that (f) both $\text{UP}(\vec{xz}, S_1)$ and $\text{UP}(\vec{zy}, S_2)$ are true. Since \vec{xy} and \vec{xz} are consecutive outgoing arcs at x , we have (a) $S_1(x) = S(x) \in \{\text{source}, \text{sourcey}, \text{ordinary}\}$. Similarly, (b) $S_2(y) = S(y) \in \{\text{sink}, \text{sinky}, \text{ordinary}\}$. Since \vec{xz} is incoming at z , we have $S_1(z) \in \{\text{sink}, \text{ordinary}, \text{sinky}\}$. However, if $S_1(z) = \text{sinky}$, then z is not bimodal in $T_{\vec{xy}}$. Thus (c) $S_1(z) \in \{\text{sink}, \text{ordinary}\}$. Similarly, (d) $S_2(z) \in \{\text{source}, \text{ordinary}\}$. Finally, if z is ordinary in both $T_{\vec{xz}}$ and $T_{\vec{zy}}$, then z is not bimodal in $T_{\vec{xy}}$. Thus (e) $S_1(z) = \text{sink}$ or $S_2(z) = \text{source}$.

(\Leftarrow) Let $T_{\vec{xz}}$ be an upward planar directed outerplane triangulation of $G_{\vec{xz}}^*$ respecting S_1 ($T_{\vec{xz}}$ is a single edge if \vec{xz} is external). Let $T_{\vec{zy}}$ be an upward planar directed outerplane triangulation of $G_{\vec{zy}}^*$ respecting S_2 ($T_{\vec{zy}}$ is a single edge if \vec{zy} is external). Such triangulations exist because $\text{UP}(\vec{xz}, S_1)$ and $\text{UP}(\vec{zy}, S_2)$ are true. Let $T_{\vec{xy}}$ be the triangulation of $G_{\vec{xy}}^*$ determined from $T_{\vec{xz}}$ and $T_{\vec{zy}}$ by adding the arc \vec{xy} . Since $T_{\vec{xz}}$, $T_{\vec{zy}}$, and (x, y, z) are acyclic, $T_{\vec{xy}}$ is acyclic. Since x is bimodal in $T_{\vec{xz}}$, it is bimodal in $T_{\vec{xy}}$. Similarly, y is bimodal in $T_{\vec{zy}}$. As described above, the conditions on $S_1(z)$ and $S_2(z)$ imply that z is bimodal in $T_{\vec{xy}}$. Every other vertex is bimodal in $T_{\vec{xy}}$ because it is bimodal in $T_{\vec{xz}}$ or in $T_{\vec{zy}}$. Hence, $T_{\vec{xy}}$ is bimodal.

Let s_1, t_1 and w_1 (s_2, t_2 and w_2 ; s, t and w) be the number of sources, sinks, and sink-switches in $T_{\vec{xz}}$ (resp., in $T_{\vec{zy}}$; resp., in $T_{\vec{xy}}$), respectively. By Theorem 1, $s_i + t_i = w_i + 1$, for $i \in \{1, 2\}$. If z is a sink in $T_{\vec{xz}}$ and ordinary in $T_{\vec{zy}}$, then $s = s_1 + s_2$, $t = t_1 + t_2 - 1$ (for z), and $w = w_1 + w_2$. If z is a source in $T_{\vec{zy}}$ and ordinary in $T_{\vec{xz}}$, then $s = s_1 + s_2 - 1$ (for z), $t = t_1 + t_2$, and $w = w_1 + w_2$. If z is a sink in $T_{\vec{xz}}$ and a source in $T_{\vec{zy}}$, then $s = s_1 + s_2 - 1$ (for z) and $t = t_1 + t_2 - 1$ (for z) and $w = w_1 + w_2 - 1$ (for z). In all three cases, it follows that $s + t = w + 1$.

By Theorem 1, $T_{\vec{xy}}$ is upward planar. By construction, $T_{\vec{xy}}$ respects $S(x)$ and $S(y)$ and contains edge (x, z) outgoing at x and edge (z, y) incoming at y . \square

Lemma 3. *There is an upward planar directed outerplane triangulation $T_{\vec{xy}}$ that respects $S(x)$ and $S(y)$ and that contains edges (x, z) and (y, z) incoming at z if and only if \vec{xz} and \vec{zy} are potential arcs of G^* and there are statuses S_1 of \vec{xz} and S_2 of \vec{zy} such that the following conditions hold: (a) $S_1(x) = S(x) \in \{\text{source}, \text{sourcey}, \text{ordinary}\}$, (b) $S(y) \in \{\text{sinky}, \text{ordinary}\}$, (c) $S_2(y) \in \{\text{source}, \text{ordinary}\}$, (d) $S(y) = \text{ordinary}$ if and only if $S_2(y) = \text{source}$, (e) $S(y) = \text{sinky}$ if and only if $S_2(y) = \text{ordinary}$, (f) $S_1(z) \in \{\text{sink}, \text{sinky}, \text{ordinary}\}$, (g) $S_2(z) \in \{\text{sink}, \text{sinky}, \text{ordinary}\}$, (h) $S_1(z) \in \{\text{sink}, \text{ordinary}\}$ or $S_2(z) = \text{sink}$, (i) $S_2(z) \in \{\text{sink}, \text{ordinary}\}$ or $S_1(z) = \text{sink}$, and (j) both $\text{UP}(\vec{xz}, S_1)$ and $\text{UP}(\vec{zy}, S_2)$ are true.*

Lemma 4. *There is an upward planar directed outerplane triangulation $T_{\vec{xy}}$ that respects $S(x)$ and $S(y)$ and that contains edges (z, x) and (z, y) outgoing at z if and only if \vec{zx} and \vec{zy} are potential arcs of G^* and there are statuses S_1 of \vec{zx} and S_2 of \vec{zy} such that the following conditions hold: (a) $S_2(y) = S(y) \in \{\text{sink}, \text{sinky}, \text{ordinary}\}$, (b) $S(x) \in \{\text{sourcey}, \text{ordinary}\}$, (c) $S_1(x) \in \{\text{sink}, \text{ordinary}\}$, (d) $S(x) = \text{ordinary}$ if and only if $S_1(x) = \text{sink}$, (e) $S(x) = \text{sourcey}$ if and only if $S_1(x) = \text{ordinary}$, (f) $S_1(z) \in \{\text{source}, \text{sourcey}, \text{ordinary}\}$, (g) $S_2(z) \in \{\text{source}, \text{sourcey}, \text{ordinary}\}$, (h)*

$S_1(z) \in \{\text{source, ordinary}\}$ or $S_2(z) = \text{source}$, (i) $S_2(z) \in \{\text{source, ordinary}\}$ or $S_1(z) = \text{source}$, and (j) both $UP(\vec{z}\vec{x}, S_1)$ and $UP(\vec{z}\vec{y}, S_2)$ are true.

For any status S of \vec{xy} and for a particular vertex $z \in V(G_{xy}^*) - x - y$, it can be checked in $O(1)$ time whether an upward planar directed outerplane triangulation $T_{\vec{xy}}$ exists that respects $S(x)$ and $S(y)$ and that contains edges (x, z) and (y, z) by checking whether the conditions in at least one of Lemmata 2-4 are satisfied. Further, $UP(\vec{xy}, S)$ is true if and only if there exists a vertex $z \in V(G_{xy}^*) - x - y$ such that an upward planar directed outerplane triangulation $T_{\vec{xy}}$ exists that respects $S(x)$ and $S(y)$ and that contains edges (x, z) and (y, z) . Thus, we can determine $UP(\vec{xy}, S)$ in $O(n)$ time since there are less than n possible choices for z .

This completes the proof of Theorem 3. The time complexity analysis can be strengthened as follows. Suppose that every internal face of G^* has at most t vertices. Then each vertex v is incident to less than $t \cdot \deg_{G^*}(v)$ potential edges and the total number of potential arcs is less than $2 \sum_v t \cdot \deg_{G^*}(v) \leq 8tn$. Since each potential arc has nine statuses, and since there are less than t choices for z , the time complexity is $O(t^2n)$. In particular, if G^* is an outerplane triangulation, then the time complexity is $O(n)$.

4 Reducing Mixed Plane Graphs to Mixed Plane Triangulations

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

Theorem 4. *Let G be an n -vertex mixed plane graph. There exists an $O(n^2)$ -vertex mixed plane triangulation G' such that G is upward planar if and only if G' is. Moreover, G' can be constructed from G in $O(n^2)$ time.*

Proof: By Lemma 1, an $O(n)$ -vertex 2-connected mixed plane graph G^* can be constructed in $O(n)$ time such that G is upward planar if and only if G^* is.

We show how to construct a graph G' satisfying the statement of the theorem. In order to construct G' , we augment G^* in several steps. At each step, vertices and edges are inserted inside a face f of G^* delimited by a cycle C_f with $n_f \geq 4$ vertices. Such an insertion is done in such a way that one of the faces that is created by the insertion of vertices and edges into f has $n_f - 1$ vertices, while all the other such faces have 3 vertices. The repetition of such an augmentation yields the desired graph G' .

We now describe how to augment G^* . Consider any face f of G^* delimited by a cycle C_f with $n_f \geq 4$ vertices. Let $(u_1, u_2, \dots, u_{n_f})$ be the clockwise order of the vertices along C_f starting at any vertex. Insert a cycle C'_f inside f with $n_f - 1$ vertices $v_1, v_2, \dots, v_{n_f-1}$ in this clockwise order along C'_f . For any $1 \leq i \leq n_f - 1$, insert edges (v_i, u_i) and (v_i, u_{i+1}) inside C_f and outside C'_f ; also, insert edge (v_1, u_{n_f}) inside cycle $(u_{n_f}, u_1, v_1, v_{n_f-1})$. All the edges inserted in f are undirected. See Fig. 1. Denote by G'_f the graph consisting of cycle C_f together with the vertices and edges inserted in f . Observe that the face of G'_f delimited by C'_f has $n_f - 1$ vertices, while all the other faces into which f is split by the insertion of x_f and of its incident edges have 3 vertices.

We show that G^* before the augmentation is upward planar if and only if G^* after the augmentation is upward planar. One implication is trivial, given that G^* before the augmentation is a subgraph of G^* after the augmentation. For the other implication,

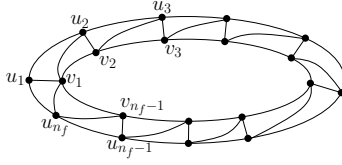


Fig. 1. Augmentation of a face f

it suffices to prove that, for any upward planar orientation C_f of C_f , there exists an upward planar orientation G'_f of G'_f that coincides with C_f when restricted to C_f .

Consider an upward planar drawing Γ_f of C_f with orientation C_f . We shall place the vertices of C'_f inside f in Γ_f , thus obtaining a drawing Γ'_f of G'_f .

Pach and Tóth [14] proved that any planar drawing of a graph G in which all the edges are y -monotone can be triangulated by the insertion of y -monotone edges inside the faces of G (the result in [14] states that the addition of a vertex might be needed to triangulate the outer face of G , which however is not the case if the outer face is bounded by a simple cycle, as in our case). Hence, there exists an index j , with $1 \leq j \leq n_f$, such that a y -monotone curve can be drawn connecting u_{j-1} and u_{j+1} inside f .

If $j < n_f$, then for $1 \leq i \leq j-1$, we place v_i inside f close to u_i , with $y(v_i) \neq y(u_i)$, so that y -monotone curves can be drawn inside f connecting v_i with u_{i-1} , with u_i , and with u_{i+1} (we draw y -monotone curves corresponding to edges of G'_f). Then, we place v_j inside f close to u_{j+1} , with $y(v_j) \neq y(u_{j+1})$, so that y -monotone curves can be drawn inside f connecting v_j with u_{j-1} , with u_j , with u_{j+1} , and with u_{j+2} (we in fact draw y -monotone curves corresponding to edges of G'_f). This is possible, since a y -monotone curve can be drawn inside f connecting v_j and u_j , by construction, and since a y -monotone curve can be drawn inside f connecting u_{j-1} and u_{j+1} , by assumption, hence a y -monotone curve can be drawn inside f connecting v_j and u_{j-1} . Then, for $j+1 \leq i \leq n_f-1$, we place v_i inside f close to u_{i+1} , with $y(v_i) \neq y(u_{i+1})$, so that y -monotone curves can be drawn inside f connecting v_i with u_i , with u_{i+1} , and with u_{i+2} (we in fact draw y -monotone curves corresponding to edges of G'_f). For any $1 \leq i \leq n_f-1$, since y -monotone curves can be drawn inside f connecting v_i with the vertices of C_f to which v_{i-1} and v_{i+1} are close, y -monotone curves can be drawn inside f representing the edges of C'_f (we in fact draw such curves). If $j = n_f$, the drawing is constructed analogously by placing v_i inside f close to u_i , for any $1 \leq j \leq n_f-1$.

The number of vertices of the mixed plane triangulation G' resulting from the augmentation is $O(n^2)$. Namely, the number of vertices inserted inside a face f of G^* with n_f vertices is $(n_f - 1) + (n_f - 2) + \dots + 3$, hence the number of vertices of G' is $\sum_f (n_f(n_f - 1)/2 - 3) = O(n^2)$, given that $\sum_f n_f \in O(n)$ (where the sums are over all the faces of G^*). Finally, the augmentation of G^* to G' can be easily performed in a time that is linear in the size of G' , hence quadratic in the size of the input graph. \square

Corollary 1. *The problem of testing the upward planarity of mixed plane graphs is polynomial-time equivalent to the problem of testing the upward planarity of mixed plane triangulations.*

5 Upward Planarity Testing of Mixed Plane Triangulations

In this section we show how to test in linear time the upward planarity of two classes of mixed plane triangulations.

A *plane 3-tree* is a plane triangulation that can be constructed as follows. Denote by H_{abc} a plane 3-tree whose outer face is delimited by a cycle (a, b, c) , with vertices a, b , and c in this clockwise order along the cycle. A cycle (a, b, c) is the only plane 3-tree H_{abc} with three vertices. Any plane 3-tree H_{abc} with $n > 3$ vertices can be constructed from three plane 3-trees H_{abd} , H_{bcd} , and H_{cad} by identifying the vertices incident to their outer faces with the same label. See Fig. 2(a).

Theorem 5. *The upward planarity of an n -vertex mixed plane 3-tree can be tested in $O(n)$ time.*

Consider an n -vertex mixed plane 3-tree H_{uvw} . We define a function $\text{UP}(xy, H_{abc})$ as follows. For each graph H_{abc} in the construction of H_{uvw} and for any distinct $x, y \in \{a, b, c\}$ we have that $\text{UP}(xy, H_{abc})$ is true if and only if there exists an upward planar orientation of H_{abc} in which cycle (a, b, c) has x as a source and y as a sink.

Observe that H_{uvw} is upward planar if and only if $\text{UP}(xy, H_{uvw})$ is true for some $x, y \in \{u, v, w\}$ with $x \neq y$. The necessity comes from the fact that, in any upward planar orientation of H_{uvw} , the cycle delimiting the outer face of H_{uvw} has exactly one source x and one sink y , by Theorem 2. The sufficiency is trivial.

We show how to compute the value of $\text{UP}(xy, H_{abc})$, for each graph H_{abc} in the construction of H_{uvw} .

If $|H_{abc}| = 3$, then let $x, y, z \in \{a, b, c\}$ with $x \neq y$, $x \neq z$, and $y \neq z$. Then, $\text{UP}(xy, H_{abc})$ is true if and only if edges (x, y) , (x, z) , and (z, y) are not prescribed to be outgoing at y , outgoing at z , and outgoing at y , respectively. Hence, if $|H_{abc}| = 3$ the value of $\text{UP}(xy, H_{abc})$ can be computed in $O(1)$ time.

Second, if $|H_{abc}| > 3$, denote by H_{abd} , H_{bcd} , and H_{cad} the three graphs that compose H . We have the following:

Lemma 5. *For any distinct $x, y, z \in \{a, b, c\}$, $\text{UP}(xy, H_{abc})$ is true if and only if:*

- (1) $\text{UP}(xy, H_{xyd})$, $\text{UP}(xd, H_{zxd})$, and $\text{UP}(zy, H_{yzd})$ are all true; or
- (2) $\text{UP}(xy, H_{xyd})$, $\text{UP}(xz, H_{zxd})$, and $\text{UP}(dy, H_{yzd})$ are all true.

Proof Sketch: (\implies) Assume that H_{abc} has an upward planar orientation \mathbf{H}_{abc} with x and y as a source and sink in $\{a, b, c\}$, respectively, (let $z \in \{a, b, c\}$ with $z \neq x, y$). Edge (z, d) might be outgoing or incoming at z , as in Figs. 2(b) and 2(c), respectively. In the first case, $\text{UP}(xy, H_{xyd})$, $\text{UP}(zy, H_{yzd})$, and $\text{UP}(xd, H_{zxd})$ are all true, while in the second case $\text{UP}(xy, H_{xyd})$, $\text{UP}(dy, H_{yzd})$, and $\text{UP}(xz, H_{zxd})$ are all true.

(\impliedby) Consider the case in which $\text{UP}(xy, H_{xyd})$, $\text{UP}(xd, H_{zxd})$, and $\text{UP}(zy, H_{yzd})$ are all true, the other case is analogous. Then, there exist upward planar orientations \mathbf{H}_{xyd} , \mathbf{H}_{zxd} , and \mathbf{H}_{yzd} of H_{xyd} , H_{zxd} , and H_{yzd} with x and y , with x and d , and with z and y as a source and sink, respectively. Orientations \mathbf{H}_{xyd} , \mathbf{H}_{zxd} , and \mathbf{H}_{yzd} together yield an orientation $\text{UP}(xy, H_{xyz})$ of \mathbf{H}_{xyz} , which is upward planar by Theorem 2. \square

For each graph H_{abc} in the construction of H_{uvw} and for any distinct $x, y \in \{a, b, c\}$, the conditions in Lemma 5 can be computed in $O(1)$ time by dynamic programming. Thus, the running time of the algorithm is $O(n)$. This concludes the proof of Theorem 5.

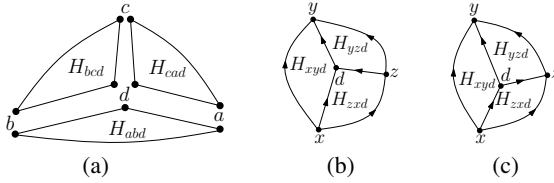


Fig. 2. (a) Construction of a plane 3-tree H_{abc} with $n > 3$ vertices. (b)-(c) Distinct orientations of edge (z, d) in two upward planar orientations of H_{abc} .

We now deal with mixed plane triangulations with no cycle of undirected edges.

Theorem 6. *The upward planarity of an n -vertex mixed plane triangulation in which the undirected edges induce a forest can be tested in $O(n)$ time.*

Proof: Let G be an n -vertex mixed plane triangulation. Let F be the set of undirected edges of G . We assume that F contains no external edge of G . Indeed, F contains at most two external edges: We can guess the orientation of all the external edges in F , and for each of the four possibilities, independently, test the upward planarity for the mixed graph G in which only the internal edges of F are undirected.

We prove the statement by induction on the size of F .

If $|F| = 0$, then G is a directed plane triangulation and its upward planarity can be tested in linear time by checking whether G satisfies the conditions in Theorem 2.

If $|F| > 0$, consider a leaf v in the forest whose edge set is F . Denote by (v, w) the only undirected edge of G incident to v . By the assumptions, (v, w) is an internal edge of G . Let (v, w, x_1) and (v, w, x_2) be the internal faces of G incident to edge (v, w) .

Suppose that both edges (x_1, v) and (x_2, v) are incoming at v . If v has an outgoing incident edge, then by the bimodality condition in Theorem 2, edge (v, w) is incoming at v in every upward planar orientation of G . Suppose that v has no outgoing incident edge. If v is the sink of G (recall that the edges incident to the outer face of G are directed), then edge (v, w) is incoming at v in every upward planar orientation of G , by the single sink condition in Theorem 2. Otherwise, edge (v, w) is outgoing at v in every upward planar orientation of G , again by the single sink condition in Theorem 2.

Analogously, if both (x_1, v) and (x_2, v) are outgoing at v , the orientation of edge (v, w) can be decided without loss of generality.

Assume that (x_1, v) and (x_2, v) are incoming and outgoing at v , respectively, the case in which they are outgoing and incoming at v is analogous. We have two cases.

Case 1: (x_1, x_2) is an edge of G . By the acyclicity condition in Theorem 2, edge (x_1, x_2) is outgoing at x_1 in every upward planar orientation of G .

If $\deg(v) = 3$, then remove v and its incident edges from G , obtaining a mixed plane triangulation G' with one fewer undirected edge than G . Inductively test whether G' admits an upward planar orientation. If not, then G does not admit any upward planar orientation, either. If G' admits an upward planar orientation G' , then construct an upward drawing Γ' of G' ; insert v in Γ' inside cycle (w, x_1, x_2) , so that $y(v) > y(x_1)$, $y(v) < y(x_2)$, and $y(v) \neq y(w)$. Draw y -monotone curves connecting v with each of w, x_1 , and x_2 . The resulting drawing Γ of G is an upward planar orientation G of G .

provided that it coincides with G' when restricted to G' , the edges (x_1, v) and (x_2, v) are drawn as y -monotone curves according to their orientations, and the edge (v, w) is drawn as a y -monotone curve.

If $\deg(v) > 3$, then the cycle (w, x_1, x_2) does not delimit a face of G , and it contains non-empty sets V' and V'' of vertices in its interior and its exterior, respectively. Then, two upward planarity tests can be performed, namely one for the subgraph G' of G induced by $V' \cup \{w, x_1, x_2\}$, and one for the subgraph G'' of G induced by $V'' \cup \{w, x_1, x_2\}$. If one of the tests fails, then G admits no upward planar orientation. Otherwise, upward planar orientations G' of G' and G'' of G'' together form an upward planar orientation G of G , provided that each edge of (w, x_1, x_2) has the same orientation in G' and in G'' .

Case 2: (x_1, x_2) is not an edge of G . Remove (v, w) from G and insert a directed edge (x_1, x_2) outgoing at x_1 inside face (x_1, v, x_2, w) . This results in a graph G' with one fewer undirected edge than G . We show that G is upward planar iff G' is.

Suppose that G admits an upward planar orientation G . Let Γ be an upward planar drawing of G . Remove edge (v, w) from G in Γ . Draw edge (x_1, x_2) inside cycle $C_f = (x_1, v, x_2, w)$, thus ensuring the planarity of the resulting drawing Γ' of G' , following closely the drawing of path (x_1, v, x_2) , thus ensuring the upwardness of Γ' .

Suppose that G' admits an upward planar orientation G' . Let Γ' be an upward planar drawing of G' . Remove (x_1, x_2) from Γ' . Since G' is acyclic, C_f has three possible orientations in G' . In Orientation 1, w is its source and x_2 its sink; in Orientation 2, x_1 is its source and w its sink; finally, in Orientation 3, x_1 is its source and x_2 its sink. If C_f is oriented in G' as in Orientation 1 (as in Orientation 2), then draw edge (v, w) inside C_f in Γ' , thus ensuring the planarity of the resulting drawing Γ of G , following closely the drawing of path (w, x_1, v) (resp., of path (v, x_2, w)), thus ensuring the upwardness of Γ . If C_f is oriented in G' as in Orientation 3, slightly perturb the position of the vertices in Γ' so that $y(v) \neq y(w)$. Draw edge (v, w) in Γ' as follows. Suppose that $y(v) < y(w)$, the other case being analogous. Draw a line segment inside C_f starting at v and slightly increasing in the y -direction, until reaching path (x_1, w, x_2) . Then, follow such a path to reach w . This results in an upward drawing of edge (v, w) inside C_f , hence in an upward planar drawing of G .

Finally, the running time of the described algorithm is clearly $O(n)$. □

6 Conclusions

We considered the problem of testing the upward planarity of mixed plane graphs. We proved that the upward planarity testing problem is $O(n^3)$ -time solvable for mixed outerplane graphs. It would be interesting to investigate whether our techniques can be strengthened to deal with larger classes of mixed plane graphs, e.g. series-parallel plane graphs. Also, since testing upward planarity is a polynomial-time solvable problem for directed outerplanar graphs [15], it might be polynomial-time solvable for mixed outerplanar graphs without a prescribed plane embedding as well.

We proved that the upward planarity testing problem for mixed plane graphs is polynomial-time equivalent to the upward planarity testing problem for mixed plane triangulations (and showed two classes of mixed plane triangulations for which the

problem can be solved efficiently). This, together with the characterization of the upward planarity of directed plane triangulations in terms of acyclicity, bimodality, and uniqueness of the sources and sinks (see [2] and Theorem 2), might indicate that a polynomial-time algorithm for testing the upward planarity of mixed plane triangulations should be pursued. On the other hand, Patrignani [16] proved that testing the existence of an acyclic and bimodal orientation for a mixed plane graph is NP-hard.

Acknowledgments. Thanks to Hooman Reisi Dehkordi, Peter Eades, Graham Farr, Seok-Hee Hong, and Brendan McKay for useful discussions on the problems considered in this paper.

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* 12(6), 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(1), 132–169 (1998)
4. Binucci, C., Didimo, W.: Upward planarity testing of embedded mixed graphs. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 427–432. Springer, Heidelberg (2011)
5. Boesch, F., Tindell, R.: Robbins’s theorem for mixed multigraphs. *Amer. Math. Monthly* 87(9), 716–719 (1980)
6. Chung, F., Garey, M., Tarjan, R.: Strongly connected orientations of mixed multigraphs. *Networks* 15(4), 477–484 (1985)
7. Didimo, W., Giordano, F., Liotta, G.: Upward spirality and upward planarity testing. *SIAM J. Discrete Math.* 23(4), 1842–1899 (2009)
8. Eiglsperger, M., Kaufmann, M., Eppinger, F.: An approach for mixed upward planarization. *J. Graph Algorithms Appl.* 7(2), 203–220 (2003)
9. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.* 31(2), 601–625 (2001)
10. Gutwenger, C., Jünger, M., Klein, K., Kupke, J., Leipert, S., Mutzel, P.: A new approach for visualizing uml class diagrams. In: Diehl, S., Stasko, J., Spencer, S. (eds.) SOFTVIS, pp. 179–188. ACM (2003)
11. Hansen, P., Kuplinsky, J., de Werra, D.: Mixed graph colorings. *Math. Meth. Oper. Res.* 45(1), 145–160 (1997)
12. Healy, P., Lynch, K.: Two fixed-parameter tractable algorithms for testing upward planarity. *Int. J. Found. Comput. Sci.* 17(5), 1095–1114 (2006)
13. Hutton, M.D., Lubiw, A.: Upward planarity testing of single-source acyclic digraphs. *SIAM J. Comput.* 25(2), 291–311 (1996)
14. Pach, J., Tóth, G.: Monotone drawings of planar graphs. *J. Graph Theory* 46(1), 39–47 (2004)
15. Papakostas, A.: Upward planarity testing of outerplanar dags. In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 298–306. Springer, Heidelberg (1995)
16. Patrignani, M.: Finding bimodal and acyclic orientations of mixed planar graphs is NP-complete. Tech. Rep. RT-DIA-188-2011, Dept. Comp. Sci. Autom., Roma Tre Univ. (2011)
17. Sotskov, J.N., Tanaev, V.S.: Chromatic polynomial of a mixed graph. *Vestsi Akademii Navuk BSSR. Ser. Fiz. Mat. Navuk* 6, 20–23 (1976)

Upward Planarity Testing: A Computational Study

Markus Chimani¹ and Robert Zeranski²

¹ Theoretical Computer Science, Osnabrück University, Germany

² Algorithm Engineering, Friedrich-Schiller-University Jena, Germany
markus.chimani@uni-osnabrueck.de, robert.zeranski@uni-jena.de

Abstract. A directed acyclic graph (DAG) is *upward planar* if it can be drawn without any crossings while all edges—when following them in their direction—are drawn with strictly monotonously increasing y -coordinates. Testing whether a graph allows such a drawing is known to be NP-complete, but there is a substantial collection of different algorithmic approaches known in literature.

In this paper, we give an overview of the known algorithms, ranging from combinatorial FPT and branch-and-bound algorithms to ILP and SAT formulations. Most approaches of the first class have only been considered from the theoretical point of view, but have never been implemented. For the first time, we give an extensive experimental comparison between virtually all known approaches to the problem.

Furthermore, we present a new SAT formulation based on a recent theoretical result by Fulek et al. [8], which turns out to perform best among all known algorithms.

1 Introduction

When drawing directed graphs, in particular DAGs, one often wants to make the edges' orientations clearly recognizable by having all edges pointing in the same general direction, w.l.o.g. upward. A *y-monotone* drawing is thus one, where the curves representing the edges have strictly monotonously increasing y -coordinates when traversing them from their source to the target vertices. More formally, a y -monotone edge intersects any horizontal line at most once, while its source vertex is drawn below its target vertex.

A second central concept in graph drawing is planarity, i.e., we want to avoid crossing edges if possible. The question of *upward planarity* of a DAG G is hence the question whether there exists a crossing-free y -monotone drawing of G . While (undirected) planarity is linear time solvable, upward planarity turns out to be NP-complete to decide [9]. Nonetheless, due to the problem's centrality, several exponential-time algorithms have been developed.

A core result is that the problem becomes polynomial time solvable if the graph's embedding (i.e., the order of the edges around their vertices) is fixed [1]. Based thereon, the historically first algorithms are fixed-parameter tractable (FPT) algorithms where the parameters are essentially bounding the (in general

exponential) number of possible embeddings; the algorithms are testing upward planarity for each possible embedding [10]. The process can be sped up using a polynomial time algorithm to solve the important special case of series-parallel graphs [7]. The special case of single-source DAGs is also polynomial time solvable [1, 5, 11, 13] but not the focus of this paper.

A different approach is based on relaxing the upward requirement (*quasi-upward planarity* [2], see below) and considering the optimization problem to minimize the violating edges. There, the embedding enumeration is coupled with a sophisticated method to obtain upper and lower bounds for partially embedded graphs, allowing for a branch-and-bound algorithm. Finally, the most recent approach [3] is to formulate the problem as an integer linear program (ILP) or boolean satisfiability problem (SAT), to be solved with a corresponding solver.

In Section 2, we summarize the core ideas of these algorithms. We also present a *new* SAT formulation based on a recent theoretical result by Fulek et al. [8].

Before this paper, the only reported implementations were for the branch-and-bound and the ILP/SAT approach. The former implementation is in fact considering the more general optimization problem (instead of the decision version), and both implementations are based on two different underlying libraries. This made a direct comparison worrisome. In this paper (Section 3), we report on our consistent implementations of all the discussed algorithms. They share as much code as was feasibly possible, to maximize the fairness of the comparison. Hence, we are for the first time able to make substantiated claims about the algorithms' respective applicability in practice.

2 Algorithms

We always consider a DAG $G = (V, E)$ to test for upward planarity. A *combinatorial* embedding of G is specified by cyclically ordering the edges around their incident vertices. A *planar* embedding additionally chooses an external face.

2.1 FPT Algorithms

A fixed-parameter tractable (FPT) algorithm, with respect to some parameter k , is an algorithm with running time $\mathcal{O}(f(k) \cdot \text{poly}(n))$ where $\text{poly}(n)$ is a polynomial function in the size of the input (here, $n := |G|$), and $f(k)$ is any computable function (typically an exponential function) only dependent on k . A central ingredient of all known combinatorial FPT algorithms is the following result [1]:

Theorem 1 (Bertalozzi, Di Battista, Liotta, Mannino). *Let $G = (V, E)$ be an embedded DAG. There is an algorithm testing whether this embedding of G allows an upward planar drawing in $\mathcal{O}(n^2)$ time.*

Let G be planar and biconnected. We can decompose the underlying undirected graph into its triconnectivity components in linear time. These can be efficiently organized as an *SPQR-tree* [6]. For notational simplicity, we may talk about an *SPR-tree* (as Q-nodes, representing single edges, are not necessary):

The SPR-tree $\mathcal{T}(G)$ is a tree with three kinds of *nodes*: S- and P-nodes represent serial and parallel components, respectively; R-nodes represent planar triconnected components. We call these components the *skeletons* associated to the nodes. An edge in a skeleton S may be real or *virtual*; in the latter case, it represents a subgraph, described by the subtree attached to S 's node in $\mathcal{T}(G)$.

Embedding Enumeration (EE). The natural approach to test upward planarity of an unembedded graph is to test every possible embedding of G , using the algorithm of Theorem 1. As the number of embeddings is, in general, exponential in the size of the graph, one has to seek for a meaningful parameter to bound the number of embeddings. The SPR-tree can be used to efficiently enumerate all possible embeddings of a graph. We can bound the number of embeddings by $\mathcal{O}(t! \cdot 2^t)$, where t is the number of nodes in our SPR-tree, which leads to an overall running time of $\mathcal{O}(t! \cdot 2^t \cdot n^2)$ to test upward planarity [10].

In the same publication, a kernelization algorithm for sparse (not necessarily biconnected) graphs is presented. Using the following preprocessing steps (until none is applicable anymore), leaves a graph with at most $30k^2 + 2k$ vertices and at most $(2k + 1)!$ embeddings; thereby, $k = |E| - |V|$ is the number of edges (minus 1) the (preprocessed) graph has more than a tree. Let a *chain* be a path in G where all inner vertices have degree 2, The preprocessing steps are: (PP1) remove vertices of degree 1; (PP2) replace chains where each inner vertex v has $\text{indeg}(v) = \text{outdeg}(v) = 1$ by single correspondingly oriented edges; (PP3) remove chains where both end vertices coincide; (PP4) for each set of *parallel* chains, remove all but one chain (parallel chains are those that have a common start vertex, a common end vertex, and an identical sequence of edge orientations along the chain). This preprocessing requires $\mathcal{O}(n^2)$ time.

After the preprocessing steps, again, all embeddings are tested in overall $\mathcal{O}(n^2 + k^4 \cdot (2k + 1)!)$ time. Observe that (PP1)–(PP4) are valid in general (although one has to specifically consider the case of biconnectivity-breaking PP4). Hence, when testing all embeddings after preprocessing, we in fact obtain an algorithm—denoted by **EE** in the following—with running time $\mathcal{O}(\min(t! \cdot 2^t \cdot n^2, n^2 + k^4 \cdot (2k + 1)!))$ for biconnected DAGs.

Upward Spirality (SPIR). Consider the SPR-tree rooted at some arbitrary node. Informally, *upward spirality* is a measure of how much a skeleton is “rolled up” around its poles (the end nodes of the virtual edge representing the node’s parent). Furthermore, one has to distinguish several *pole categories*, i.e., local properties of the embedding around the pole vertices. For details of the definitions and the following algorithms cf. [7].

For series-parallel graphs, upward spirality allows to develop a polynomial dynamic programming algorithm that traverses the SPR-tree bottom up—recall that for such graphs it only has S- and P-nodes. At each node μ , we store a set of feasible (spirality/pole category)-pairs to upward embed the graph encoded by the SPR-tree rooted at μ . This information can then be used to obtain a corresponding set for the parent node, etc. We denote this algorithm by **SPIR-sp**. Its running time is $\mathcal{O}(n^4)$, but there are large constants hidden in the \mathcal{O} -notation, cf. Section 3.1.

Using this algorithm as a building block, one can establish an FPT algorithm for general DAGs, where the different configurations of the R-nodes w.r.t. each other have to be enumerated. This leads to a running time of $\mathcal{O}(d^r n^3 + dr^2 n + d^2 n^2)$ where d is the largest diameter of any skeleton and r is the number of R-nodes. We call this algorithm SPIR.

2.2 Branch-and-Bound via Quasi-Upward Planarity (BB)

In a *quasi-upward planar* drawing, we relax upwardness such that each edge only has to be drawn y -monotonously within an arbitrary small neighborhood of its incident vertices. In [2], a branch-and-bound algorithm is established which produces a quasi-upward drawing maximizing the number of fully y -monotone edges: For a given embedding, the minimum number of non- y -monotone edges can be computed in $\mathcal{O}(n^2 \log n)$ time using minimum cost-flow techniques.

Now, we can consider all possible embeddings of the graph via the SPR-tree. At any moment we have a *partial* embedding—several embeddings are fixed while the others are free. The algorithm in [2] is able to compute upper and lower bounds for the number of non- y -monotone edges in this case. If the current lower bound is worse than the global upper bound, we can avoid testing all embeddings further down in the search tree, which have the same fixed embeddings in common.

Observe that a DAG is upward planar iff there is a quasi-upward planar drawing where all edges are y -monotone. Hence, when testing upward planarity, we can prune a partial embedding in the search tree whenever we obtain a lower bound strictly greater than 0. We denote this algorithm by BB. Formally, this algorithm could also be considered an FPT algorithm with the same worst-case running time as EE.

2.3 SAT Formulations

A SAT formulation of a decision problem instance \mathcal{I} is a propositional logic formula that is satisfiable if and only if \mathcal{I} has the answer *true*. The formula is typically given in conjunctive normal form, i.e., as a set of clauses, each of which has to be satisfied. Each clause is a disjunction of (possibly negated) variables. For the sake of readability, we will provide *rules* as propositional formulae; it is straight-forward to transform them into their corresponding clauses.

Ordered Embeddings (OE). An edge e *dominates* an edge f if there is a directed path (possibly of length 0) from e 's target to f 's source vertex in G . Clearly, f has to be drawn above e in any upward drawing. A pair of edges is *non-dominating* if neither edge dominates the other. Let \mathcal{N} denote the set of all non-dominating edge pairs of G .

In [3], a SAT formulation based on *ordered embeddings* has been proposed. We consider a strict total (vertical) order of the vertices together with a strict partial (horizontal) order of the edges; edges are comparable w.r.t. this order iff they are non-dominating each other. We model the vertical order by introducing

boolean variables $\tau(v, w)$ for each proper pair of vertices v and w . Intuitively, $\tau(v, w) = \mathbf{true}$ means that v is drawn *below* w . Since the vertex order is to be strict, $\tau(v, w) = \mathbf{false}$ means that v is *above* w . We may use the shorthand $\tau(w, v) := \neg\tau(v, w)$ for notational simplicity. To establish a strict total order, it then suffices to require transitivity via (R_τ^t) . The *upward rules* (R^u) ensure that all edges are drawn upward.

$$\begin{aligned} \tau(u, v) \wedge \tau(v, w) \rightarrow \tau(u, w) & \quad \forall \text{ pairwise distinct } u, v, w \in V. & (R_\tau^t) \\ \tau(v, w) = \mathbf{true} & \quad \forall (v, w) \in E. & (R^u) \end{aligned}$$

Similarly, we can establish the horizontal order of the edges by introducing variables $\sigma(e, f)$ for each pair $\{e, f\} \in \mathcal{N}$. Thereby (if both edges are vertically overlapping), $\sigma(e, f) = \mathbf{true}$ implies that e is to the *left* of f , and the satisfied shorthand $\sigma(f, e) := \neg\sigma(e, f)$ implies that e is to the *right* of f . Again, we simply require transitivity:

$$\sigma(e, f) \wedge \sigma(f, g) \rightarrow \sigma(e, g) \quad \forall \{e, f\}, \{f, g\}, \{e, g\} \in \mathcal{N}. \quad (R_\sigma^t)$$

Based on this (upward) order system, we can establish planarity using surprisingly simple *planarity rules*: We only have to ensure that two adjacent edges e and f are on the same side of g (non-incident to the common vertex of e and f) if they both vertically overlap with g . Let $e \cap f$ denote the common vertex of two edges e, f , and $\mathcal{P} := \{(e, f, g) \mid \{e, g\}, \{f, g\} \in \mathcal{N} \wedge e \cap f \notin g\}$ the set of edge-triplets as described above. We have:

$$\left(\tau(x, e \cap f) \wedge \tau(e \cap f, y) \right) \rightarrow \left(\sigma(e, g) \leftrightarrow \sigma(f, g) \right) \quad \forall (e, f, g = (x, y)) \in \mathcal{P}. \quad (R^p)$$

The collection of the above rules allows a satisfying truth assignment to τ and σ if and only if G is upward planar [3]. Given such an assignment, it is trivial to construct the embedding in linear time. We denote this formulation by **OE**.

Hanani-Tutte type characterization (FPSS). The classical Hanani-Tutte theorem shows that a graph drawn such that all pairs of non-adjacent edges cross an even number of times is planar. A similar result has been established by Pach and Tóth [12], and was only recently improved upon by Fulek et al. [8]:

Theorem 2 (Pach, Tóth; Fulek, Pelsmajer, Schaefer, Štefankovič). *Let $G = (V, E)$ be a DAG. If G has a y -monotone¹ drawing such that every pair of non-adjacent edges crosses an even number of times, then there is a y -monotone planar embedding of G with the same location of vertices.*

To prove this theorem, [8] gives a quadratic time algorithm testing whether G allows a y -monotone drawing with prespecified vertex positions. This is essentially done by solving an equation system over (e, v) -moves. Such a move redraws the edge e by deforming it, close to the y -coordinate of v , into a horizontal “spike” that passes around v . There is a y -monotone embedding iff there

¹ In the original publications, these theorems were stated in terms of x -monotonicity.

is a set of (e, v) -moves turning a y -monotone drawing into a drawing in which every non-adjacent pair of edges crosses an even number of times. A simple set of only two equation classes suffices to describe all possible selections of (e, v) -moves that may lead to an even-crossing solution. These equations, in fact, resemble a 2SAT formulation (each clause has at most 2 literals), with the only prerequisite to know the vertical relationship between the DAG’s vertices. This allows us to cast this powerful theoretical tool into a new SAT formulation not unlike OE:

We reuse the above boolean variables τ and rules $(R_\tau^l) \cup (R^u)$ to guarantee an upward strict total order on V . We can then use these variables as indicators to activate or deactivate the above 2SAT-clauses to link move-variables. For every edge e and node v , we introduce a boolean variable $\varrho(e, v)$ that indicates whether we perform an (e, v) -move. Let s_e and t_e denote the start and target vertex of an edge e , respectively. We obtain

$$\begin{aligned} (\tau(s_e, s_f) \wedge \tau(s_f, t_e) \wedge \tau(t_e, t_f)) &\rightarrow (\varrho(e, s_f) \leftrightarrow \neg\varrho(f, t_e)) \quad \forall e, f \in E \quad (R_0^m) \\ (\tau(s_e, s_f) \wedge \tau(t_f, t_e)) &\rightarrow (\varrho(e, s_f) \leftrightarrow \varrho(e, t_f)) \quad \forall e, f \in E \quad (R_1^m) \end{aligned}$$

where the subformulae after the implication are the clauses from the 2SAT suggested by [8], for the scenario described by the rules’ left-hand sides.

Corollary 1. *Let $G = (V, E)$ be a DAG. G is upward planar if and only if the formula composed of rules $(R_\tau^l) \cup (R^u) \cup (R_0^m) \cup (R_1^m)$ is satisfiable.*

Hybrid formulations. Constructing an upward planar embedding (in polynomial time) using only a feasible truth assignment for FPSS is non-trivial, and in fact an open problem.² In order to obtain an embedding from the FPSS formulation, we consider two variants of hybridizing FPSS and OE, as extracting an embedding is trivial for the latter. Recall that both formulations use the same τ variables to establish a vertical order of the vertices. Hence we can simply put all rules together in one large formulation, denoted by HF.

Alternatively (denoted by HL), we can first compute a satisfying assignment to FPSS. If it exists, we can “learn” the subsolution for the τ variables to fix the τ variables in OE and solve the so-restricted OE to obtain a matching σ assignment.

3 Experiments

3.1 Considered Algorithms and Their Implementations

Overall, we consider the practical performances of the following algorithms:

FPT	b&b	SAT	ILP (see note below)
EE, SPIR-sp, (SPIR)	BB	OE, FPSS, HF, HL	iOE, iFPSS, iHF, iHL

² An algorithm achieving this is currently under development [personal communication with Marcus Schaefer]

All experiments were performed on an Intel Xeon E5520, 2.27 GHz, 8 GB RAM running Debian 6. We implemented the algorithms as part of the Open Graph Drawing Framework (www.ogdf.net), using *minisat* as our SAT solver and CPLEX 12 as our ILP solver. For both solvers, we used their default settings. For all considered instances (available at www.cs.uos.de/theo/inf), we first performed the preprocessing steps (PP1)–(PP4), as described in Section 2.1.

A note on the ILPs. Given the SAT formulations, it is straight-forward to construct integer linear programs (using only binary variables) along the same lines, for which to test feasibility. This has been done for OE in [3], leading to a vastly weaker practical performance than the SAT approach. We note that FPSS (and the hybridizations) also allow such ILPs. We performed all the below experiments also for the ILP variants. However, also for iFPSS and the hybridizations, the pure feasibility testing functionality of ILP solvers—lacking sophisticated propagation and backtracking—is clearly too weak to allow compatible performance. We will hence disregard the ILP approaches in the following discussions.

A note on BB. The branch-and-bound algorithm has only been devised for biconnected graphs. Our implementation hence inherits this restriction. As described above, we strengthened BB for the special case of upward planarity testing by pruning any subproblems with a lower bound larger than 0.

In [2], the practical performance of an implementation (within GDTToolkit) has been reported on a set of random instances, denoted by *BDD* in the following. Out of the originally 300 considered instances, only 200 are still available³. Of those, we can discard 139, as they are not DAGs. The largest remaining instances have 100 vertices and take 0.03 seconds on average (cf. Table 2). In [2]—on a clearly slower PC and not restricted to the pure upward planarity test!—the graphs with 100 vertices require roughly 100 seconds on average. This gives us a hint that our implementation is not vastly inefficient.

A note on EE. As BB is already restricted to biconnected graphs, we also restricted our implementation of EE to this case, to be able to efficiently generate all embeddings purely via the SPR-tree. Our implementation is able to enumerate $\approx 100,000$ (bimodal) embeddings per second (disregarding any testing time).

A note on SPIR-sp and SPIR. The spirality-based algorithms are very theoretically demanding algorithms. We provide the seemingly first implementation of the polynomial case for series-parallel graphs (SPIR-sp). However, the algorithm’s theoretical beauty is unfortunately not matched with practicability. E.g., when combining tables in the bottom-up dynamic programming, there is a very large number of possible combinations of choices to check. Although this number is bounded by a constant, we often need to check close to the theoretical worst-case of 2^{49} ($\approx 5.6 \times 10^{14}$) combinations. This constitutes the main bottleneck of the algorithm; further theoretical research may be able to bring down this vast number. In our implementation, we parallelized this checking via OpenMP to mitigate the effect. However, it remains to slow down the algorithm dramatically.—We will see experimental evidence in Figure 3 below.

³ Personal communication with Walter Didimo.

Table 1. # of solved instances within given time frames in seconds (*Rome* and *North*)

time (s)	<i>Rome</i>					<i>North</i>			
	[0, 0.01]	(0.01, 0.1]	(0.1, 1]	(1, 10]	> 10	[0, 0.01]	(0.01, 0.1]	(0.1, 1]	> 1
FPSS	1922	993	46	4	2	710	120	6	0
OE	1474	1223	257	11	2	625	172	39	0
HL	1384	1324	253	4	2	614	185	37	0
HF	1393	1311	254	6	3	609	187	40	0

The series-parallel algorithm **SPIR-sp**, as a base case for **SPIR**, can essentially be used as a lower bound for the running time of the latter, when applying it on the SPR-subtrees induced by only S- and P-nodes. This resembles the situation that **SPIR** could decide the R-nodes and all their embedding combinations (i.e., the reason for the exponential running time) in no time. Since obtaining these bounds is already not competitive enough, we refrained from a full implementation of the even more demanding enumeration procedures within **SPIR**.

3.2 Evaluation

In the course of our investigation, we will observe the following central findings throughout all benchmark instances, summarized here:

- F1** FPSS gives the best solution time over virtually all scenarios. When extraction of the embedding is required, HL dominates OE; both dominate HF.
- F2** FPSS (and to some extent OE) are rather independent of the number of embeddings and on whether we consider a yes- or a no-instance.
- F3** Generally, all SAT approaches are preferable over their competitors.
- F4** SPIR (and SPIR-sp) and the ILP variants are not competitive.
- F5** EE and BB work well when the number of embeddings is small. For trivial instances, and for large instances with few embeddings, EE and BB can dominate the SAT approaches.
- F6** EE is usually far weaker for no-instances than for yes-instances (as all embeddings have to be checked), while the effective pruning of BB allows it to typically solve the former faster than the latter.

Instances from Literature. There are mainly three instance sets that have been used in the context of upward planarity:

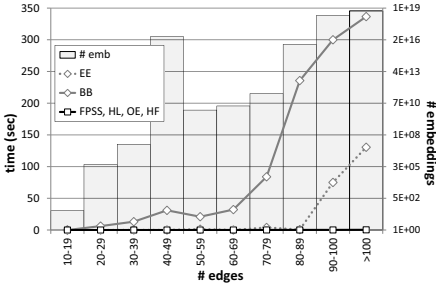
The *Rome* graphs [14] are (originally undirected) graphs with up to 100 vertices. They can be directed canonically to obtain DAGs. The *North* DAGs [4] were originally collected by AT&T and Stephen North. After filtering for bimodal planar graphs, 2967 and 836 remain, respectively. As some of our algorithms are restricted to biconnected graphs, we also consider sets *Rome2*, *North2* that are generated from the above by planar biconnectivity augmentation, obtaining (after filtering for bimodal planar graphs) 2671 and 780 instances, respectively. Furthermore, we consider the 61 *BDD* instances [2], described above in the context of BB’s implementation.

We observe in Table 1 that all SAT formulations solve the *North* instances very fast: each is solved within 1 second, most of them ($\approx 80\%$) in less than 0.01

Table 2. Number of solved instances (yes- and no-instances) within given time frames in seconds (*Rome2*, *North2*, and *BDD*). In brackets, are the number of no-instances.

time (s)	<i>Rome2</i>			<i>North2</i>				<i>BDD</i>			
	[0, 0.1]	(0.1, 1]	> 1	[0, 0.1]	(0.1, 1]	(1, 10]	> 10	[0.0, 1]	(0.1, 1]	(1, 10]	> 10
FPSS	2474 (26)	195 (6)	2 (2)	711 (33)	68 (7)	1 (0)	0 (0)	27 (3)	23 (1)	11 (2)	1 (1)
OE	1743 (12)	915 (17)	13 (5)	600 (24)	149 (13)	31 (3)	0 (0)	24 (3)	25 (1)	12 (2)	1 (1)
HL	1844 (26)	820 (6)	7 (2)	610 (33)	143 (7)	27 (0)	0 (0)	24 (3)	26 (1)	11 (2)	1 (1)
HF	1743 (11)	923 (19)	14 (5)	605 (28)	143 (9)	32 (3)	0 (0)	21 (3)	27 (1)	13 (2)	1 (1)
EE	2665 (29)	5 (4)	1 (1)	753 (27)	5 (1)	6 (4)	16 (8)	62 (7)	0 (0)	0 (0)	0 (0)
BB	2635 (22)	22 (9)	4 (3)	667 (14)	30 (11)	13 (4)	70 (11)	58 (4)	4 (3)	0 (0)	0 (0)

seconds. Generally, FPSS is the fastest SAT, solving 99% of *Rome* in under 0.1 seconds, whereas OE achieves a ratio of 90% (\rightarrow **F1**).

**Fig. 1.** *North2*; avg. runtime vs. # of edges. The different SATs are visually indistinguishable.

On the biconnected benchmark sets, we can compare the SAT approaches to the other implementations (Table 2): For *Rome2* and *BDD*, both EE and BB seem faster than any SAT formulation. This is mainly due to the triviality of many instances and the necessary overhead of SAT formulations and solvers. Already when considering ≤ 1 second computation time, all approaches solved nearly the same number of instances. We observe that instances of both these sets have only very few embeddings (*Rome2*: max. 36,864, avg. 372), *BDD*: max. 11,528, avg. 512) (\rightarrow **F5**). As an interesting side note, the lone instance in the last *BDD* column requires roughly 200 seconds for all SAT approaches; it is solved trivially by BB and EE as it has only two planar bimodal embeddings to check. The *North2* instances, however, have many embeddings (avg: 10^{17}), cf. Fig. 1. There, all SAT formulations dominate EE and BB for the non-trivial instances (\rightarrow **F1**,**F5**).

Constructed Instances. The above instances are very simple, small, and are solved too quickly to deduce general findings. Therefore, we consider a set *Rand* of generated biconnected instances with $n = 50, 100, 150, 200$ nodes and density $|E|/|V| = 1.2, 1.4, \dots, 2.4$. As suggested in [2], we start with a triangle graph and iteratively perform random edge-subdivisions and face-splits (adding an edge within a face). Now, we orient this embedded graph to be upward planar, and invert $i = 1\%, 2\%, 3\%, 4\%$ of the edges (retaining the DAG property). We generate 1120 instances, 10 per possible parameter setting, which have up to 2.5×10^{15} embeddings (3.7×10^{12} on average). See Fig. 2 for a detailed graphical analysis. We use a timeout of 600 seconds, using this value for unsolved instances when averaging. Overall (Fig. 2, top-left), we can observe $FPSS < HL < OE < HF < BB < EE$ for the running times, FPSS being the clear winner (\rightarrow **F1**,**F3**,**F5**,**F6**). The

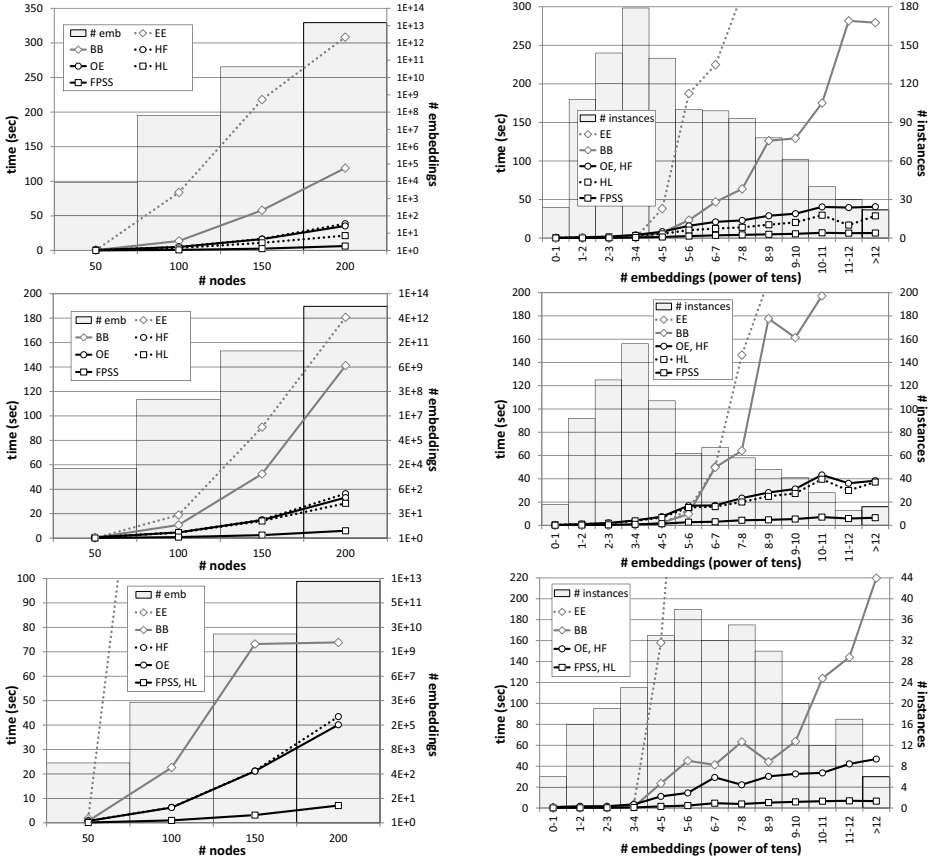


Fig. 2. *Rand* instances; avg. runtime vs. # of nodes (left) or embeddings (right). The latter is given as powers of 10, i.e., $> 10^{12}$ embeddings are considered. We group SAT formulations if they are visually indistinguishable. The first row considers all instances; the second and third row considers only the yes- and no-instances, respectively.

SATs, in particular FPSS, seem nearly oblivious to the number of embeddings in the graph (Fig. 2, top-right, \rightarrow F2). It is instructive to consider the yes- and no-instances independently: We see that EE performs somewhat reasonable for yes-instances, but fails for no-instances (where it has to check all bimodal embeddings). In contrast to this, BB becomes even faster for the latter instances, due to its efficient pruning of large classes of “hopeless” subembeddings (\rightarrow F6). The SAT approaches behave very similar for both kinds of instances (\rightarrow F2).

Now, we consider biconnected series-parallel graphs *SP* to evaluate *SPIR-sp*. We generated a test set of 4500 instances (10 instances per parameter setting) with $m = 20, 40, \dots, 300$ edges. They are constructed bottom up with probability $p = 0.1, 0.3, \dots, 0.9$ to perform a serialization instead of a parallelization. We embed the graph, choose an upward planar orientation for the edges, and invert

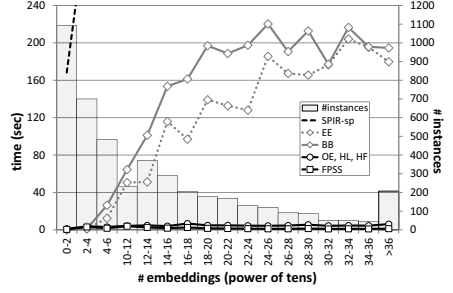
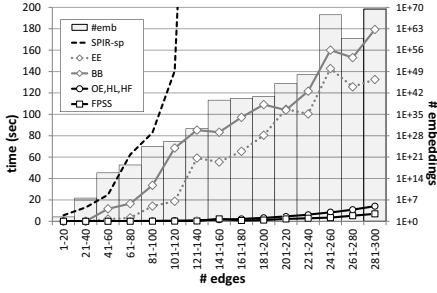


Fig. 3. *SP* instances; avg. runtime vs. # of edges (left) and embeddings (right)

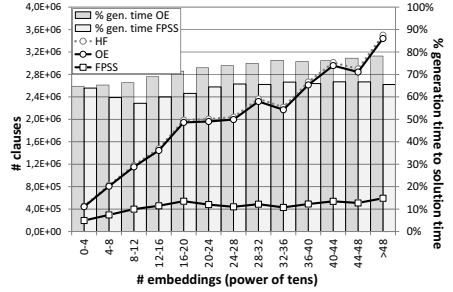
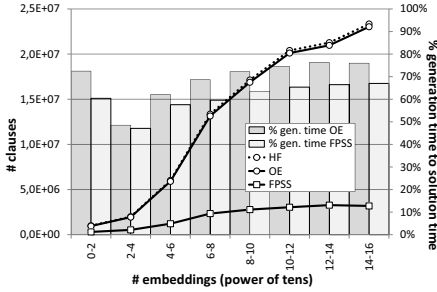


Fig. 4. *Rand* (left) and *SP* (right); the line plots show the number of generated clauses, relative to the number of embeddings (as a power of 10). The bars show the percentage of the running time spent to generate the formula (in contrast to solving it).

$i = 0, 10, 20, 30, 40, 50\%$ of the edges (retaining the DAG property). Again, we use a timeout of 600 seconds, using this value for unsolved instances when averaging. *SPIR-sp*—although formally the only polynomial time algorithm in this comparison—offers the clearly weakest performance, solving no instance with over 100 vertices in under 5 minutes and running into the time limit for all graphs with more than 120 edges (Fig. 3, left). The picture is analogous (Fig. 3, right) when looking at the runtime depending on the number of embeddings (\rightarrow **F4**). On *SP*, *EE* performs better than *BB*, but this is since nearly 90% of the instances happen to be upward planar. Considering yes- and no-instances independently, we can observe the same pattern as for *Rand* (\rightarrow **F6**).

Details on SAT. Although *FPSS* and *OE* have the same number of clauses in \mathcal{O} -notation—dominated by (R_τ^t) over the common τ variables—the former has considerably fewer additional clauses. In fact, this seems to be one of the main reasons of *FPSS*’s superior performance. To back-up this assumption, consider the line diagrams in Fig. 4. They show that *FPSS* is rather independent of the number of embeddings (\rightarrow **F2**). Impressively, the (minor) difference between *OE* and *HF* (“*OE* \cup *FPSS*”) shows that the number of clauses *FPSS* has to consider additionally to (R_τ^t) is negligible.

The bar diagrams in Fig. 4 show that the SATs spend a large portion of their time ($\approx 70\%$) with (trivially) *constructing* the formula. This explains the overhead for trivial instances where EE and BB can be faster than SAT ($\rightarrow \mathbf{F5}$).

Acknowledgements. We thank Marcus Schaefer for pointing us to [8] and its potential applicability within our SAT approach, Walter Didimo for helpful discussions on BB, and Fabrice Stelmacher and Kerstin Gössner for implementation support with BB and SPIR, respectively.

References

1. Bertolazzi, P., Di Battista, G., Liotta, G., Mannino, C.: Upward drawings of tri-connected digraphs. *Algorithmica* 12(6), 476–497 (1994)
2. Bertolazzi, P., Di Battista, G., Didimo, W.: Quasi-upward planarity. *Algorithmica* 32(3), 474–506 (2002)
3. Chimani, M., Zeranski, R.: Upward planarity testing via SAT. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 248–259. Springer, Heidelberg (2013)
4. Di Battista, G., Garg, A., Liotta, G., Parise, A., Tamassia, R., Tassinari, E., Vargiu, F., Vismara, L.: Drawing directed acyclic graphs: An experimental study. *Int. J. of Computational Geometry and Appl.* 10(6), 623–648 (2000)
5. Di Battista, G., Liu, W.P., Rival, I.: Bipartite graphs, upward drawings, and planarity. *Information Processing Letters* 36(6), 317–322 (1990)
6. Di Battista, G., Tamassia, R.: On-line planarity testing. *SIAM J. on Computing* 25, 956–997 (1996)
7. Didimo, W., Giordano, F., Liotta, G.: Upward spirality and upward planarity testing. *SIAM J. on Discrete Mathematics* 23(4), 1842–1899 (2009)
8. Fulek, R., Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Hanani–Tutte, monotone drawings, and level-planarity. *Thirty Essays on Geom. Graph Th.*, 263–287 (2013)
9. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM J. on Computing* 31(2), 601–625 (2002)
10. Healy, P., Lynch, K.: Fixed-parameter tractable algorithms for testing upward planarity. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 199–208. Springer, Heidelberg (2005)
11. Hutton, M.D., Lubiw, A.: Upward planar drawing of single source acyclic digraphs. In: *Proc. SODA 1991*, pp. 203–211 (1991)
12. Pach, J., Toth, G.: Monotone drawings of planar graphs. *J. of Graph Theory* 46(1), 39–47 (2004)
13. Papakostas, A.: Upward planarity testing of outerplanar dags. In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 298–306. Springer, Heidelberg (1995)
14. Welzl, E., Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: An experimental comparison of four graph drawing algorithms. *Computational Geometry* 7, 303–325 (1997)

Characterizing Planarity by the Splittable Deque

Christopher Auer, Franz J. Brandenburg,
Andreas Gleißner, and Kathrin Hanauer

University of Passau, Germany
{auer, brandenb, gleissner, hanauer}@fim.uni-passau.de

Abstract. A *graph layout* describes the processing of a graph G by a data structure \mathcal{D} , and the graph is called a \mathcal{D} -graph. The vertices of G are totally ordered in a *linear layout* and the edges are stored and organized in \mathcal{D} . At each vertex, all edges to predecessors in the linear layout are removed and all edges to successors are inserted. There are intriguing relationships between well-known data structures and classes of planar graphs: The stack graphs are the outerplanar graphs [4], the queue graphs are the arched leveled-planar graphs [12], the 2-stack graphs are the subgraphs of planar graphs with a Hamilton cycle [4], and the deque graphs are the subgraphs of planar graphs with a Hamilton path [2]. All of these are proper subclasses of the planar graphs, even for maximal planar graphs.

We introduce *splittable deques* as a data structure to capture planarity. A splittable deque is a deque which can be split into sub-deques. The splittable deque provides a new insight into planarity testing by a game on switching trains. Here, we use it for a linear-time planarity test of a given rotation system.

1 Introduction

In a graph layout, the vertices are processed according to a total order, which is called *linear layout*. The edges correspond to data items that are inserted to and removed from a data structure: Each edge is inserted at the end vertex that occurs first according to the linear layout and is removed at its other end vertex. These operations obey the principles of the underlying data structure, such as “last-in, first-out” for a stack or “first-in, first-out” for a queue. Stack layouts (also known as book embeddings) and queue layouts have been studied extensively, e. g., in [4, 5, 7, 8, 10–12, 16, 18], and are used for 3D drawings of graphs [16], in VLSI design [5] and in other application scenarios [12]. Moreover, Gauss codes and permutation networks of two parallel stacks are characterized by two-stack graphs [14].

Graph layouts are a powerful tool to study planar graphs. A graph G is a \mathcal{D} -graph if it has a layout in \mathcal{D} . The stack graphs are the outerplanar graphs, and the 2-stack graphs are the subgraphs of planar graphs with a Hamiltonian cycle [4]. Heath et al. [8, 12] have characterized queue graphs as the arched leveled-planar graphs. Such graphs have a planar drawing with vertices placed

on levels. Inter-level edges connect vertices between two adjacent levels and intra-level edges (the *arches*) connect the left-most vertex to accessible vertices on the right side. In [2], we have characterized the proper leveled-planar graphs, i. e., the arched leveled-planar graphs without arches, as the bipartite queue graphs. Graph layouts can be extended to subdivisions where edges of the graph are replaced by paths. A graph is planar if and only if it has a subdivision that has a layout in two stacks [8].

In [1, 2], we have studied double-ended queue (*deque*) layouts: A deque has two ends, a head and a tail, and items can be inserted and removed at both sides. It can emulate two stacks and additionally allows for queue edges, i. e., edges inserted and removed at opposite sides. In [2], we have shown that the surplus power of a deque in comparison to two stacks captures the difference between Hamiltonian paths and cycles: A graph is a deque (2-stack) graph if and only if it is the subgraph of a planar graph with a Hamiltonian path (cycle). In fact, a planar embedding of a graph with a Hamiltonian path reflects the way the edges are processed in the deque: Fig. 1(c) shows an embedded graph with the Hamiltonian path 1, 2, 3, 4, 11, 12, 13, which is the linear layout. An edge to the right of the path, e. g., edges e_6 and e_9 , is inserted and removed at the tail of the deque whereas the queue edges change sides, e. g., e_3 and e_4 are inserted at the tail and removed at the head. Although more powerful than two stacks, not all planar graphs are deque graphs since there are maximal planar graphs with no Hamiltonian path and the respective decision problem is \mathcal{NP} -hard [2]. Yannakakis has shown that four stacks are sufficient and necessary for all planar graphs [17]. However, there are non-planar graphs that have a layout in four and even three stacks. This raises the following question: What is the additional operation a deque must perform to layout exactly the planar graphs? It turns out that the ability to split the deque into pieces is the adequate operation. We show that a graph is planar if and only if it is a *splittable deque (SD)* graph.

Our proof takes an algorithmic viewpoint: We give a linear-time algorithm to test whether or not a rotation system is planar, which uses the SD to process all edges. A rotation system defines the counterclockwise order of edges around each vertex and it is planar if it admits a plane drawing of the graph. In a nutshell, the algorithm is a depth-first search (DFS) which tries to process all edges in the SD according to the rotation system. Planarity follows if this is possible. Otherwise, an edge that cannot be processed, e. g., removed from the deque, causes a crossing. The algorithm is a means to an end for our characterization of planarity and there are other algorithms especially designed for solving the same problem [6]. Nevertheless, our algorithm has the benefit that it operates on an elementary data structure, i. e., a deque, which is very simple in comparison to other ones used for general planarity tests [15]. Note that any two-cell embedding on a surface of genus k can be defined by a rotation system. In particular, it is not sufficient to only test locally at each face whether the rotation system causes crossings. Another challenge are crossings between edges incident to the same vertex, which are ignored in the general case. As the SD exploits the structure of a graph's DFS tree, our characterization is related to the characterization of planarity by de Fraysseix et al. [9].

The SD also provides a playful characterization of planarity: At GD 2012, we presented the poster “Testing Planarity by Switching Trains” [3]. There, the edges are modeled as cars which have to be appended at the head and tail of a train which models the deque. The vertices are train stations which are the sources and destinations of the cars and, at junctions, the train can be split. We also implemented a Java game¹, which uses the time-reversed variant of the SD, i. e., the mergeable deque. The player is asked to switch the cars such that all can be removed at their destination station. The graph underlying a game level is obtained from a GraphML file. If it is possible to bring all cars to their destination without an error, then the underlying graph is planar.

2 Preliminaries

We consider simple, undirected, and connected graphs $G = (V, E)$ with vertices V and edges E such that $|V| \geq 2$. A graph $G = (V, E)$ is planar if it has a plane drawing which maps the vertices to distinct points in the plane and edges $\{u, v\}$ to Jordan arcs from u to v such that Jordan arcs do not cross except at common end points. A *rotation system* \mathcal{R}_v defines a cyclic order of edges around each vertex v . From a plane drawing, we obtain a *planar rotation system* which is the counterclockwise ordering of the edges around each vertex in the drawing. Given a rotation system \mathcal{R}_v , each edge e has a successor edge $\text{Succ}_v(e)$ and a predecessor edge $\text{Pre}_v(e)$ at vertex v .

A *DFS tree* $\mathcal{T} = (V, E_{\mathcal{T}})$ is a rooted, directed spanning tree of G obtained from a DFS traversal starting at a root vertex r . We assume that the *tree edges* $E_{\mathcal{T}}$ are directed from the parent to its children. We denote by $u \rightarrow v$ that $(u, v) \in E_{\mathcal{T}}$. By $u \xrightarrow{+} v$, we denote a path of tree edges (at least one) from u to v . Vertex u is an *ancestor* of v and v is a *descendant* of u . By $u \xrightarrow{*} v$, we denote $u = v$ or $u \xrightarrow{+} v$. \mathcal{T} partitions E into tree edges $E_{\mathcal{T}}$ and *forward edges* F . For each forward edge $\{u, v\} \in F$, there is a path $u \xrightarrow{+} v$ where u is an ancestor of v .

A *linear layout* is a total ordering \prec of the vertices. If $u \prec v$, then u is called *predecessor* of v and v *successor* of u . In a graph layout, a vertex v can be seen as a processing unit that receives as *input* a data structure from which v 's edges to predecessors are removed and edges to successors are inserted. Insertions and removals obey the modus operandi of the data structure. The resulting data structure is the *output* of the vertex and the input to the immediate successor. The input to the first and output of the last vertex is empty.

A *deque* is a doubly linked list whose content is denoted by $\mathcal{C} = (e_1, \dots, e_k)$, where e_1 is at the *head* \mathbf{h} and e_k is at the *tail* \mathbf{t} . The empty deque is denoted by $()$. We denote by $e_i \in \mathcal{C}$ that e_i is in \mathcal{C} and by $e_i \ll_{\mathcal{C}} e_j$ that $i < j$ in \mathcal{C} . We omit the subscript in $\ll_{\mathcal{C}}$ if it is clear from the context which configuration is meant.

¹ <http://www.infosun.fim.uni-passau.de/br/games/derail.jar>

An edge $e \in \mathcal{C}$ can be removed if it is situated at the head or the tail of \mathcal{C} . In the following, we denote by \mathcal{C}_v the input of vertex v .

Let $G = (V, E)$ be a graph endowed with a rotation system and assume that G contains Hamiltonian path $p = (v_1, \dots, v_n)$. Path p is a (degenerate) DFS tree of G and it induces a linear layout with $v_i \prec v_j$ if and only if $i < j$ for all $1 \leq i, j \leq n$. The edges on p are directed from each vertex to its immediate successor. A rotation system of G defines the order in which the edges are inserted to and removed from a deque and Algorithm 1 shows how: It takes as input a vertex v and its rotation system \mathcal{R}_v along with a deque \mathcal{C}_v . e_p and e_s are the edges to the immediate predecessor and successor of v on p , respectively. The value \perp indicates that v is the first or last vertex. Let v_i with $1 < i < n$ be an inner vertex of p . Its rotation system is sketched in Fig. 1(a). In Algorithm 1, all edges e_1^h, \dots, e_k^h between e_p and e_s are inserted and removed at the head in counterclockwise order of the rotation system (lines 1–4). An edge is removed if it points to a predecessor and it is inserted if it points to a successor. We say that these edges are to the *left* of p at v . Then, at v , all edges e_1^t, \dots, e_j^t to the *right* of p between e_p and e_s are processed at the deque’s tail in reversed, i. e., clockwise, order of the rotation system (lines 5–10). Whereas an edge can always be inserted, removing might not be possible if the edge is not accessible at the corresponding side of the deque. In this case, Algorithm 1 aborts and returns \perp . At the endpoints v_1 and v_n of p , the rotation system can be divided at an arbitrary position. In this case, Algorithm 1 processes all edges at the head. Note that the edges of the Hamiltonian path p are not processed in the deque. The reason is that these edges can always be processed canonically without interfering with any other edges: First of all, the edge to the immediate predecessor is removed at the head and, last of all, the edge to the immediate successor is inserted at the head. Hence, we can safely ignore all edges on p . We say that a rotation system *admits* a deque layout if subsequently calling `ProcessDeque` for all vertices v_1, \dots, v_n in order never returns \perp and the input to v_1 is the empty deque and so is the output of v_n . From [2], we obtain the following proposition.

Proposition 1. *The rotation system of a graph with a Hamiltonian path is planar if and only if it admits a deque layout.*

For an example, consider Fig. 1(c). The input of vertex 3 is the deque with content (e_4, e_3) and edge e_6 is inserted to the tail of the deque, which results in (e_4, e_3, e_6) . Consider edge e' which crosses e_6 . This is also reflected in the deque layout: At vertex 4, e_4 is removed at the deque’s head and e' inserted to the tail resulting in (e_3, e_6, e') . At vertex 11, e_6 must be removed at the tail which is not possible since e' is in its way.

3 The Splittable Deque

The SD enhances the deque by allowing it to be split. This is the dual to the mergeable deque of Kosaraju [13]. In order to define SD layouts, we generalize

Algorithm 1. ProcessDeque

Input: vertex v with rotation system $\mathcal{R}_v = (e_1, \dots, e_k)$, deque \mathcal{C}_v , edges e_p (e_s) to immediate predecessor (successor); or \perp if v is first (last)

Output: new content of the deque or \perp if deque layout is not possible

```

1  $e \leftarrow \text{Succ}_v(e_p)$  if  $e_p \neq \perp$ , else  $\text{Succ}_v(e_s)$ 
2 while  $e \neq e_s \wedge e \neq e_p$  do
3   if  $e$  is edge to successor then  $\mathcal{C}_v.\text{insertAtHead}(e)$ 
4   else  $\mathcal{C}_v.\text{removeAtHead}(e)$  or return  $\perp$  if not possible  $e \leftarrow \text{Succ}_v(e)$ 
5 if  $e_s \neq \perp \wedge e_p \neq \perp$  then  $v$  is an inner vertex
6    $e \leftarrow \text{Pre}_v(e_p)$ 
7   while  $e \neq e_s$  do
8     if  $e$  is edge to successor then  $\mathcal{C}_v.\text{insertAtTail}(e)$ 
9     else  $\mathcal{C}_v.\text{removeAtTail}(e)$  or return  $\perp$  if not possible
10     $e \leftarrow \text{Pre}_v(e)$ 
11 return  $\mathcal{C}_v$ 

```

linear layouts to tree layouts. A *tree layout* is an ordered DFS tree $\mathcal{T} = (V, E_{\mathcal{T}})$ of a graph G , i. e., a DFS tree in which the children of each inner vertex are totally ordered from left to right. Remember that the linear layout of a deque layout describes the processing pipeline, i. e., if u is the immediate predecessor of v , then the output of u is the input of v . With a tree layout \mathcal{T} , a vertex v can have multiple immediate successors, namely, its children w_1, w_2, \dots, w_l in order. For a graph G , let $\mathcal{T} = (V, E_{\mathcal{T}})$ be a tree layout with root r . The input $\mathcal{C}_r = ()$ of root r is empty. Let $\mathcal{C}_v = (e_1, \dots, e_k)$ be the input of vertex $v \in V$ and let w_1, \dots, w_l be the children of v in \mathcal{T} in order. Assume for now that v has at least one child. At first, \mathcal{C}_v is split into $l \leq k$ consecutive and disjoint pieces c_{w_1}, \dots, c_{w_l} , where \mathcal{C}_v is the concatenation of c_{w_1}, \dots, c_{w_l} . These l pieces constitute l new SDs. Each forward edge from an ancestor of v in \mathcal{T} has to be removed and each forward edge to a descendant has to be inserted at one of these SDs. The SD obtained from c_{w_i} after all removals and insertions is the input \mathcal{C}_{w_i} of child w_i . If v is a leaf, its output must be empty. If \mathcal{T} is a path, the SD is never split and behaves like a deque. A graph is an *SD graph* if it has a tree layout such that all edges can be processed in the SD.

For an example, consider the graph in Fig. 1(b) whose rotation system can be obtained from the drawing. The vertices are numbered according to a DFS run starting at root 1. The red, dashed edges are ignored for the moment. All tree edges are directed from parent to children and drawn bold. In Fig. 1(d), the tree layout as defined by the DFS tree is displayed where the children are ordered from left to right according to the rotation system. In fact, the rotation system as shown in Fig. 1(d) is equal to the one obtained from Fig. 1(b). In the example, the input SD of vertex 4 is $\mathcal{C}_4 = (e_5, e_2, e_1, e_4, e_3, e_6)$. At vertex 4, the SD is split into three pieces $c_5 = (e_5, e_2)$, $c_{10} = (e_1)$, and $c_{11} = (e_4, e_3, e_6)$. Afterwards, forward edge e_4 is removed at the head of c_{11} . We obtain $\mathcal{C}_5 = (e_5, e_2)$, $\mathcal{C}_{10} = (e_1)$, and $\mathcal{C}_{11} = (e_3, e_6)$ as inputs to vertices 5, 10, and 11, respectively. In principle, \mathcal{C}_4 can

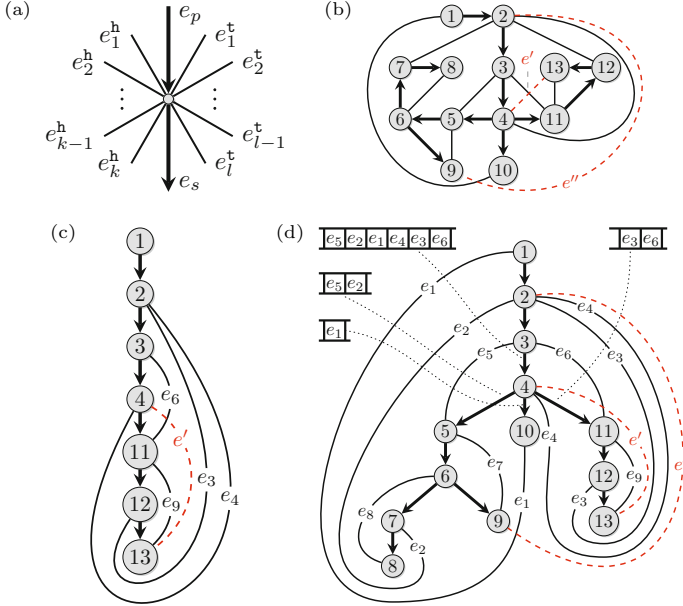


Fig. 1. (a) shows how a path splits the rotation system of a vertex and at which side of the deque the edges are processed. A planar graph with its DFS tree (directed and bold drawn edges) is shown in (b) and its tree layout in (d) along with the input SD for some vertices. (c) shows a path from the root to a leaf in the DFS tree, which corresponds to a deque layout.

also be split such that $c_{10} = (e_1, e_4)$ and $c_{11} = (e_3, e_6)$ and then e_4 is removed at the tail of c_{10} . This ambiguity occurs at vertices with more than one child and does not influence the property of a graph of being an SD graph.

4 Testing Planarity of a Rotation System by the SD

In this section, we prove our main result.

Theorem 1. *A graph G is an SD graph if and only if G is planar.*

To prove this theorem, we use Algorithm 2 which uses the SD to test whether a given rotation system is planar. By showing its correctness, Theorem 1 follows. Algorithm 2 defines the recursive routine **PlanarRS** which takes as input a vertex v endowed with a rotation system, a tree layout \mathcal{T} , the input SD \mathcal{C}_v for v , and the tree edge e_a from its parent p in \mathcal{T} or \perp if v is the root. The tree layout \mathcal{T} is obtained from a prior DFS run, where the children of each vertex v are ordered from left to right as given by the rotation system. In a nutshell, **PlanarRS** first splits \mathcal{C}_v into pieces c_{w_1}, \dots, c_{w_l} , one for each of v 's children w_1, \dots, w_l , and then removes and inserts all forward edges according to the rotation system of

Algorithm 2. PlanarRS

Input: vertex v with rotation system $\mathcal{R}_v = (e_1, \dots, e_k)$, tree layout $\mathcal{T} = (V, E_{\mathcal{T}})$, SD \mathcal{C}_v , tree edge e_p from parent p or \perp if v is the root

Output: true \mathcal{R} is planar; false if one of e_1, \dots, e_k causes a crossing

- 1 **if** v is leaf **then return** whether $\text{ProcessDeque}(v, \mathcal{R}_v, \mathcal{C}_v, e_p, \perp) = ()$
- 2 $w_1, \dots, w_l \leftarrow$ children of v in $E_{\mathcal{T}}$
- 3 Split \mathcal{C}_v into pieces c_{w_1}, \dots, c_{w_l} such that $\forall e \in c_{w_i} : e$ is removed at v, w_i or one of w_i 's descendants or return **false** if this is not possible
- 4 **foreach** $w_i \in \{w_1, \dots, w_l\}$ **do** $\tilde{\mathcal{R}}_i \leftarrow []$; removed[w_i] \leftarrow **false** $\mathcal{S} \leftarrow$ empty stack
- 5 $e_1, \dots, e_k \leftarrow$ rotation system of v with $e_1 = e_p$ if v is not root; else $e_1 = (v, w_1)$
- 6 **foreach** $e = e_1, \dots, e_k$ **do**
 - 7 $w_i \leftarrow$ child w_i with $e \in c_{w_i}$, e is removed at descendant of w_i , or $e = (v, w_i)$
 - 8 **if** e is forward edge **then** $\tilde{\mathcal{R}}_i.append(e)$ **if** $\mathcal{S}.top() = w_i$ **then continue**
 - 9 with next edge in line 5 **if** $w_i \in \mathcal{S}$ **then**
 - 10 **while** $\mathcal{S}.top() \neq w_i$ **do**
 - 11 $w_{i'} \leftarrow \mathcal{S}.pop()$; removed[$w_{i'}$] \leftarrow **true**
 - 12 $\mathcal{C}_{w_{i'}} \leftarrow \text{ProcessDeque}(v, \tilde{\mathcal{R}}_{i'}, c_{w_{i'}}, e_p, (v, w_{i'}))$
 - 13 **if** $\mathcal{C}_{w_{i'}} = \perp$ **then return false** **else if** $\neg \text{PlanarRS}(w_{i'}, \mathcal{R}_{w_{i'}}, \mathcal{T}, \mathcal{C}_{w_{i'}}, (v, w_{i'}))$ **then return false**
 - 14 **else if** \neg removed[w_i] **then** $\mathcal{S}.push(w_i)$ **else return false**
- 15 **while** $\neg \mathcal{S}.isEmpty()$ **do**
 - 16 $w_i \leftarrow \mathcal{S}.pop()$
 - 17 $\mathcal{C}_{w_i} \leftarrow \text{ProcessDeque}(v, \tilde{\mathcal{R}}_i, c_{w_i}, e_p, (v, w_i))$
 - 18 **if** $\mathcal{C}_{w_i} = \perp$ **then return false** **else if** $\neg \text{PlanarRS}(w_i, \mathcal{R}_{w_i}, \mathcal{T}, \mathcal{C}_{w_i}, (v, w_i))$ **then return false**
- 19 **return true**

v . Afterwards, it recursively calls **PlanarRS** for all its children. If at some point, the SD cannot be split adequately or an edge cannot be removed, **false** is returned and propagated back to the initial caller of **PlanarRS**. Otherwise, **true** is returned. In the following, we assume that in a drawing of G which respects the rotation system no pair of edges crosses more than once and no edge crosses any of the tree edges: As the tree layout \mathcal{T} itself contains no cycles, it is always planar regardless of the rotation system. Also, all forward edges can be drawn such that they cause no crossing with any tree edge. This is also reflected in the SD where all tree edges can be processed canonically without interfering with any forward edge: After splitting the SD, the tree edge from the parent can be removed at the head of the first SD and, as the last step, each tree edge to child w_i can be inserted at the head of the SD for child w_i . As with the deque, we only consider forward edges in the SD layout.

To actually insert and remove forward edges to an SD, **PlanarRS** uses the routine **ProcessDeque**. The observation behind is that a deque is a special case of the SD, namely, whenever the tree layout is a path: Let G be a graph endowed

with a rotation system and \mathcal{T} be a tree layout of G . Further, let p be a path from the root to a leaf of \mathcal{T} and denote by G_p the subgraph of G induced by p which inherits G 's rotation system. For instance, the subgraph G_p for the root-to-leaf path $p = 1 \xrightarrow{+} 13$ in Fig. 1(d) is shown in Fig. 1(c). G_p 's rotation system is planar if and only if it admits a deque layout by Proposition 1. Hence, if G 's rotation system is planar, then the rotation system on every root-to-leaf path admits a deque layout. In **PlanarRS**, all edges that lie on a common root-to-leaf path p are processed in the SD just like in a deque. If this is possible, then the rotation system of each root-to-leaf path is planar. This is already reflected in line 1: If v is a leaf, then the SD is not split and must be emptied by **ProcessDeque** and **true** is returned if and only if **ProcessDeque** returns an empty SD.

In line 3, the SD \mathcal{C}_v is split into pieces c_{w_1}, \dots, c_{w_l} such that for each edge $e \in c_{w_i}$ edge e is removed at v , w_i or one of w_i 's descendants. If this is not possible, **false** is returned. Consider edge e'' in Figs. 1(b) and (d), which crosses edge e_1 . Edge e_1 is inserted at the head at vertex 1 and e'' at the tail at vertex 2, i. e., $e_1 \ll e''$. At vertex 4, the deque has to be split such that $e_1 \in c_{10}$ and $e'' \in c_5$. However, as $e_1 \ll e''$ and vertex 5 is left of vertex 10, this is not possible. In general, we obtain the following lemma:

Lemma 1. *Let e and e' be two forward edges in \mathcal{C}_v such that e (e') is removed at w_i ($w_{i'}$) or one of its descendants. It is possible to split \mathcal{C}_v such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$ if and only if e and e' do not cross.*

Proof. We assume that e and e' are inserted at u and u' and removed at x and x' , respectively. Since e and e' are forward edges, there are paths $p = u \xrightarrow{+} v \rightarrow w_i \xrightarrow{*} x$ and $p' = u' \xrightarrow{+} v \rightarrow w_{i'} \xrightarrow{*} x'$.

\Leftarrow : We prove the contrapositive and assume that \mathcal{C}_v cannot be split such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$. Either e and e' are inserted to the same side of the deque or to different sides. For the first case, assume that e and e' are both inserted at the head and, w. l. o. g., w_i is left of $w_{i'}$ in the total order of children at v . This implies that $e' \ll_{\mathcal{C}_v} e$. This situation is depicted in Fig. 2(a). Remember that **PlanarRS** inserts and removes edges to the SD as with a deque. Hence, both edges e and e' are to the left of p and p' at u and u' , respectively. There is a cycle formed by path p and forward edge e . In a drawing of G which respects the rotation system, this circle encloses a region R (dark shaded in Fig. 2(a)) such that R does not contain $w_{i'}$. As $e' \ll_{\mathcal{C}_v} e$, e is inserted before e' and, thus, either u is an ancestor of u' , or $u = u'$ and the rotation system at u is $\mathcal{R}_u = (\dots, e, \dots, e', \dots, e_d, \dots)$, where e_d is the tree edge from u to u 's child on p . In either case, the edge curve of e' starts within region R and must end outside R which inevitably causes a crossing with e . The reasoning if e and e' are inserted at the tail is analogous.

In the second case, e and e' are inserted at different sides where, w. l. o. g., e is inserted at the tail and e' at the head (cf. Fig. 2(b)). Hence, $e' \ll_{\mathcal{C}_v} e$. Since \mathcal{C}_v cannot be split adequately, this implies that w_i must be left of $w_{i'}$ at v . Again, let R be the region enclosed by p and e (dark shaded in Fig. 2(b)) such that R contains $w_{i'}$. As e is to the right of p at u and e' to the left of p' at u' , the

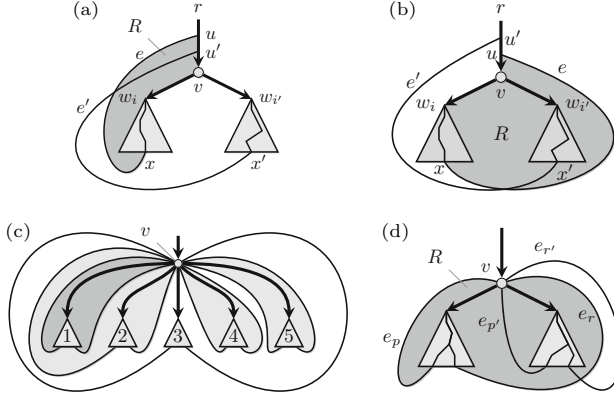


Fig. 2. The two cases where the SD cannot be split appropriately (Figs. (a) and (b)). Fig. (c) sketches nesting sectors and Fig. (d) clashing sectors.

edge curve of e' starts outside of R and must reach x' within R which leads to a crossing with e .

\Rightarrow : Again, we prove the contrapositive. We have two cases: Either e and e' lie on the same side of p and p' at u and u' , respectively, or on different sides. For the first case, we assume w. l. o. g. that w_i is left of $w_{i'}$ at v . If a crossing between e and e' is unavoidable, then u is either an ancestor of u' , or $u = u'$ and rotation system of u is $\mathcal{R}_u = (\dots, e, \dots, e', \dots, e_d, \dots)$, where e_d is the tree edge from u to u 's child on p (Fig. 2(a)). In either case, e is inserted at the head before e' and $e' \ll_{\mathcal{C}_v} e$. Thus, \mathcal{C}_v cannot be split such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$. For the second case, e lies to the right of p and e' to the left of p' (w. l. o. g.). Thus, e is inserted at the tail and e' at the head and $e' \ll_{\mathcal{C}_v} e$. If a crossing between e and e' is unavoidable, then w_i is to the left of $w_{i'}$ at v (cf. Fig. 2(b)) and, again, \mathcal{C}_v cannot be split such that $e \in c_{w_i}$ and $e' \in c_{w_{i'}}$. \square

Let $\mathcal{R}_v = (e_1, \dots, e_k)$ be the rotation system of v such that e_1 is the tree edge from v 's parent if v is an inner vertex of \mathcal{T} . If v is the root, then e_1 is the tree edge to v 's first child w_1 . After \mathcal{C}_v is successfully split into pieces c_{w_1}, \dots, c_{w_k} , **PlanarRS** finds for each piece a subsequence of \mathcal{R}_v which contains all forward edges that must be removed from and inserted to c_{w_i} . A *sector* $\tilde{\mathcal{R}}_i$ of child w_i is a subsequence of v 's rotation system such that all edges in $\tilde{\mathcal{R}}_i$ are forward edges incident to an ancestor of v or a descendant of w_i . Further, for each forward edge e there is exactly one sector $\tilde{\mathcal{R}}_i$ with $e \in \tilde{\mathcal{R}}_i$. If the rotation system is planar, then all sectors must properly nest: Consider Fig. 2(c) in which v has five children corresponding to five subtrees. In the rotation system of v , the sector that corresponds to subtree 2 encloses the sector corresponding to subtree 1, and the sector belonging to 3 encloses both. Let $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$ and $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ be two sectors of $\mathcal{R}_v = (e_1, \dots, e_k)$. We say that $\tilde{\mathcal{R}}_i$ and $\tilde{\mathcal{R}}_{i'}$ *clash* if there exist edges $e_r \in \tilde{\mathcal{R}}_i$ and $e_{r'} \in \tilde{\mathcal{R}}_{i'}$ with $p < p' < r < r'$ (see Fig. 2(d)) or $r < r' < q < q'$.

Lemma 2. *In a planar rotation system, no pair of sectors clash.*

Proof. Assume for contradiction that G 's rotation system is planar but the sectors $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$ and $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ clash. Let $e_r \in \tilde{\mathcal{R}}_i$ and $e_{r'} \in \tilde{\mathcal{R}}_{i'}$ be edges with $p < p' < r < r'$. The reasoning for the case $r < r' < q < q'$ is similar. The situation is shown in Fig. 2(d). There is a circle formed by v , e_p , e_r and a simple path between the endpoints of e_p and e_r distinct from v . In a plane drawing of G respecting the rotation system, this circle encloses a region R (shaded in Fig. 2(d)) such that it contains the endpoints of $e_{p'}$ and $e_{r'}$ other than v . As $p < p' < r < r'$, the edge curve of $e_{r'}$ starts outside of R and ends inside which leads to a crossing; a contradiction. \square

Corollary 1. *In a planar rotation system, all pairs of sectors $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$, $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ with $p < p'$, are either disjoint, i. e., $p \leq q < p' \leq q'$, or nesting, i. e., $p < p' < q' < q$.*

If all sectors are disjoint or nesting, we can construct a plane drawing at vertex v given plane drawings of all subgraphs that belong to the subtrees of v 's children. To test if the sectors are nesting, **PlanarRS** uses a stack in which v 's children w_1, \dots, w_l are inserted and removed. Further, for each child w_i , it maintains the boolean variable `removed[w_i]` which stores if w_i has been removed from the stack. **PlanarRS** subsequently processes all edges e of v in order of the rotation system (line 5). In line 6, the child w_i “responsible” for e , is determined, i. e., either e is the tree edge from v to w_i , or e is a forward edge and must be removed from c_{w_i} or e must be inserted at v and is removed at a descendant of w_i . If e is a forward edge, it is appended to sector $\tilde{\mathcal{R}}_i$. If w_i is currently on top of the stack, no further action is needed (line 7). If $w_i \in \mathcal{S}$, all children $w_{i'}$ on the stack are removed until w_i is on top. For all removed children $w_{i'}$, `removed[$w_{i'}$]` is set to `true`. If $w_i \notin \mathcal{S}$ and `removed[w_i]` = `false`, w_i is pushed onto the stack (line 12).

If w_i is not in the stack and has previously been removed, `false` is returned (line 12) as the rotation system is not planar for the following reasons: Child w_i has been removed in a previous iteration when another child $w_{i'}$ further below in the stack needed to be on top. Let $\tilde{\mathcal{R}}_i = (e_p, \dots, e_q)$ be the sector of w_i and $\tilde{\mathcal{R}}_{i'} = (e_{p'}, \dots, e_{q'})$ the sector of $w_{i'}$. $w_{i'}$ has been inserted to \mathcal{S} before w_i and, thus, $p' < p$. In the iteration when w_i is removed from \mathcal{S} , the edge of the iteration is $e_{r'} \in \mathcal{R}_{w_{i'}}$. Further, there is an edge $e_r \in \mathcal{R}_{w_i}$ when w_i would be reinserted with $r' < r$. Altogether we get $p' < p < r' < r$ and, hence, $\tilde{\mathcal{R}}_i$ and $\tilde{\mathcal{R}}_{i'}$ clash. Thus, the rotation system of G is not planar by Lemma 2.

Whenever a child $w_{i'}$ is removed from \mathcal{S} , it must never be inserted again and, hence, $\tilde{\mathcal{R}}_{i'}$ must contain all edges to be processed in $c_{w_{i'}}$. In line 10, **ProcessDeque** is called with sector $\tilde{\mathcal{R}}_{i'}$ and $c_{w_{i'}}$ as parameters. Remember that **ProcessDeque** needs two edges on a path which divide the rotation system into a left and right half. Here, these two edges are the tree edge from the parent of v , if existent, and the tree edge from v to $w_{i'}$. If the return value $\mathcal{C}_{w_{i'}}$ of **ProcessDeque** is \perp , not all edges could be processed in the deque and the rotation system is not planar by Proposition 1. Thus, `false` is returned in line 11.

Otherwise, `PlanarRS` is called recursively on $w_{i'}$ with input deque $\mathcal{C}_{w_{i'}}$ (line 11). After all forward edges of the rotation system are processed, all children remaining in \mathcal{S} are removed (lines 13–16) and `ProcessDeque` and `PlanarRS` are called. If all calls of `PlanarRS` return `true`, the rotation system of each root-to-leaf path is planar and so are the rotation systems at each vertex with more than one child.

Lemma 3. *PlanarRS in Algorithm 2 returns true if and only if the rotation system of its input graph is planar.*

Since `PlanarRS` obeys the SD’s modus operandi, Theorem 1 follows. Each edge is inserted and removed exactly once. Further, in the loop from lines 5–12 each forward edge is processed at most twice during the algorithm and, by using the DFS numbers, it is possible to decide in time $\mathcal{O}(1)$ in which subtree the end vertex of a forward edge lies. Also, each vertex is inserted at most twice to the stack for each edge. The operation that needs more arguing is splitting the deque (line 3) for which also a linear running time can be achieved.

Whenever `PlanarRS` returns `false`, the edges that cause a crossing can be determined: If one of the calls of `ProcessDeque` returns \perp , then an edge could not be removed from the SD since at least one other edge is blocking its way. Hence, these edges must cross. If the SD cannot be split adequately (line 3), then we obtain one of the situations as in Figs. 2(a) or (b) and the edges which prevent the SD from being split adequately are those that cause a crossing. Last, if some of v ’s children w_i were reinserted into the stack (lines 12 and 16), then the corresponding sectors would be classing and, hence, there exist two edges that cross according to the proof of Lemma 2 (see Fig. 2(d)).

5 Conclusion

We characterized planarity by graph layouts in the splittable deque (SD): Although a stack, two stacks, or the deque characterize large classes of planar graphs, they do not capture all. We enhanced the deque by a split-operation and showed that it characterizes planarity. For our proof, we devised a linear-time algorithm operating on the SD to test the planarity of a rotation system. If it is not planar, the operations on the SD indicate crossing edges. Our test also works for graphs with multi-edges. Given a rotation system, it defines the order of order of insertions and removals to the SD. Conversely, given the order of insertions and removals to the SD, we can find a (planar) rotation system. So a planarity testing algorithm can use the SD to find an embedding of a graph.

References

1. Auer, C., Bachmaier, C., Brandenburg, F.J., Brunner, W., Gleißner, A.: Plane drawings of queue and deque graphs. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 68–79. Springer, Heidelberg (2011)

2. Auer, C., Gleißner, A.: Characterizations of deque and queue graphs. In: Kolman, P., Kratochvíl, J. (eds.) WG 2011. LNCS, vol. 6986, pp. 35–46. Springer, Heidelberg (2011)
3. Auer, C., Gleißner, A., Hanauer, K., Vetter, S.: Testing planarity by switching trains. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 557–558. Springer, Heidelberg (2013)
4. Bernhart, F., Kainen, P.: The book thickness of a graph. *J. Combin. Theory, Ser. B* 27(3), 320–331 (1979)
5. Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM J. Algebra. Discr. Meth.* 8(1), 33–58 (1987)
6. Donafee, A., Maple, C.: Planarity testing for graphs represented by a rotation scheme. In: Banissi, E., Börner, K., Chen, C., Clapworthy, G., Maple, C., Lobben, A., Moore, C.J., Roberts, J.C., Ursyn, A., Zhang, J. (eds.) Proc. Seventh International Conference on Information Visualization, IV 2003, pp. 491–497. IEEE Computer Society, Washington, DC (2003)
7. Dujmović, V., Wood, D.R.: On linear layouts of graphs. *Discrete Math. Theor. Comput. Sci.* 6(2), 339–358 (2004)
8. Dujmović, V., Wood, D.R.: Stacks, queues and tracks: Layouts of graph subdivisions. *Discrete Math. Theor. Comput. Sci.* 7(1), 155–202 (2005)
9. de Fraysseix, H., Rosenstiehl, P.: A depth-first-search characterization of planarity. In: *Graph Theory*, Cambridge (1981); *Ann. Discrete Math.*, vol. 13, pp. 75–80. North-Holland, Amsterdam (1982)
10. Heath, L.S., Leighton, F.T., Rosenberg, A.L.: Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discret. Math.* 5(3), 398–412 (1992)
11. Heath, L.S., Pemmaraju, S.V., Trenk, A.N.: Stack and queue layouts of directed acyclic graphs: Part I. *SIAM J. Comput.* 28(4), 1510–1539 (1999)
12. Heath, L.S., Rosenberg, A.L.: Laying out graphs using queues. *SIAM J. Comput.* 21(5), 927–958 (1992)
13. Kosaraju, S.R.: Real-time simulation of concatenable double-ended queues by double-ended queues (preliminary version). In: Proc. 11th Annual ACM Symposium on Theory of Computing, STOC 1979, pp. 346–351. ACM, New York (1979)
14. Rosenstiehl, P., Tarjan, R.E.: Gauss codes, planar hamiltonian graphs, and stack-sortable permutations. *J. of Algorithms* 5, 375–390 (1984)
15. Shih, W.K., Hsu, W.L.: A new planarity test. *Theor. Comput. Sci.* 223(1-2), 179–191 (1999)
16. Wood, D.R.: Queue layouts, tree-width, and three-dimensional graph drawing. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 348–359. Springer, Heidelberg (2002)
17. Yannakakis, M.: Four pages are necessary and sufficient for planar graphs. In: Proc. of the 18th Annual ACM Symposium on Theory of Computing, STOC 1986, pp. 104–108. ACM, New York (1986)
18. Yannakakis, M.: Embedding planar graphs in four pages. *J. Comput. Syst. Sci.* 38(1), 36–67 (1989)

Strip Planarity Testing

Patrizio Angelini¹, Giordano Da Lozzo¹, Giuseppe Di Battista¹, and Fabrizio Frati²

¹ Dipartimento di Ingegneria, Roma Tre University, Italy
{angelini, dalozzo, gdb}@dia.uniroma3.it

² School of Information Technologies, The University of Sydney, Australia
brillo@it.usyd.edu.au

Abstract. In this paper we introduce and study the *strip planarity testing* problem, which takes as an input a planar graph $G(V, E)$ and a function $\gamma : V \rightarrow \{1, 2, \dots, k\}$ and asks whether a planar drawing of G exists such that each edge is monotone in the y -direction and, for any $u, v \in V$ with $\gamma(u) < \gamma(v)$, it holds $y(u) < y(v)$. The problem has strong relationships with some of the most deeply studied variants of the planarity testing problem, such as *clustered planarity*, *upward planarity*, and *level planarity*. We show that the problem is polynomial-time solvable if G has a fixed planar embedding.

1 Introduction

Testing the planarity of a given graph is one of the oldest and most deeply investigated problems in algorithmic graph theory. A celebrated result of Hopcroft and Tarjan [20] states that the planarity testing problem is solvable in linear time.

A number of interesting variants of the planarity testing problem have been considered in the literature [25]. Such variants mainly focus on testing, for a given planar graph G , the existence of a planar drawing of G satisfying certain *constraints*. For example the *partial embedding planarity* problem [1,22] asks whether a plane drawing \mathcal{G} of a given planar graph G exists in which the drawing of a subgraph H of G in \mathcal{G} coincides with a given drawing \mathcal{H} of H . *Clustered planarity testing* [10,23], *upward planarity testing* [5,16,21], *level planarity testing* [24], *embedding constraints planarity testing* [17], *radial level planarity testing* [4], and *clustered level planarity testing* [14] are further examples of problems falling in this category.

In this paper we introduce and study the *strip planarity testing* problem, which is defined as follows. The input of the problem consists of a planar graph $G(V, E)$ and of a function $\gamma : V \rightarrow \{1, 2, \dots, k\}$. The problem asks whether a *strip planar* drawing of (G, γ) exists, i.e. a planar drawing of G such that each edge is monotone in the y -direction and, for any $u, v \in V$ with $\gamma(u) < \gamma(v)$, it holds $y(u) < y(v)$. The name “strip” planarity comes from the fact that, if a strip planar drawing Γ of (G, γ) exists, then k disjoint horizontal strips $\gamma_1, \gamma_2, \dots, \gamma_k$ can be drawn in Γ so that γ_i lies below γ_{i+1} , for $1 \leq i \leq k-1$, and so that γ_i contains a vertex x of G if and only if $\gamma(x) = i$, for $1 \leq i \leq k$. It is not difficult to argue that strips $\gamma_1, \gamma_2, \dots, \gamma_k$ can be given as part of the input, and the problem is to decide whether G can be planarly drawn so that each edge is monotone in the y -direction and each vertex x of G with $\gamma(x) = i$ lies in the strip γ_i . That is, arbitrarily predetermining the placement of the strips does not alter the possibility of constructing a strip planar drawing of (G, γ) .

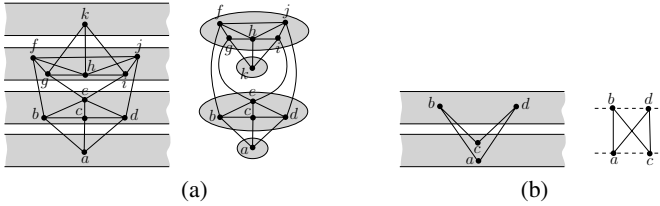


Fig. 1. (a) A negative instance (G, γ) of the strip planarity testing problem whose associated clustered graph $C(G, T)$ is c -planar. (b) A positive instance (G, γ) of the strip planarity testing problem that is not level planar.

Before presenting our result, we discuss the strong relationships of the strip planarity testing problem with three famous graph drawing problems.

Strip Planarity and Clustered Planarity. The c -planarity testing problem takes as an input a *clustered graph* $C(G, T)$, that is a planar graph G together with a rooted tree T , whose leaves are the vertices of G . Each internal node μ of T is called *cluster* and is associated with the set V_μ of vertices of G in the subtree of T rooted at μ . The problem asks whether a c -planar drawing exists, that is a planar drawing of G together with a drawing of each cluster $\mu \in T$ as a simple closed region R_μ so that: (i) if $v \in V_\mu$, then $v \in R_\mu$; (ii) if $V_\nu \subset V_\mu$, then $R_\nu \subset R_\mu$; (iii) if $V_\nu \cap V_\mu = \emptyset$, then $R_\nu \cap R_\mu = \emptyset$; and (iv) each edge of G intersects the border of R_μ at most once. Determining the time complexity of testing the c -planarity of a given clustered graph is a long-standing open problem. See [10,23] for two recent papers on the topic. An instance (G, γ) of the strip planarity testing problem naturally defines a clustered graph $C(G, T)$, where T consists of a root having k children μ_1, \dots, μ_k and, for every $1 \leq j \leq k$, cluster μ_j contains every vertex x of G such that $\gamma(x) = j$. The c -planarity of $C(G, T)$ is a necessary condition for the strip planarity of (G, γ) , since suitably bounding the strips in a strip planar drawing of (G, γ) provides a c -planar drawing of $C(G, T)$. However, the c -planarity of $C(G, T)$ is not sufficient for the strip planarity of (G, γ) (see Fig. 1(a)). It turns out that strip planarity testing coincides with a special case of a problem opened by Cortese et al. [8,9] and related to c -planarity testing. The problem asks whether a graph G can be planarly embedded “inside” an host graph H , which can be thought as having “fat” vertices and edges, with each vertex and edge of G drawn inside a prescribed vertex and a prescribed edge of H , respectively. It is easy to see that the strip planarity testing problem coincides with this problem in the case in which H is a path.

Strip Planarity and Level Planarity. The *level planarity testing* problem takes as an input a planar graph $G(V, E)$ and a function $\gamma : V \rightarrow \{1, 2, \dots, k\}$ and asks whether a planar drawing of G exists such that each edge is monotone in the y -direction and each vertex $u \in V$ is drawn on the horizontal line $y = \gamma(u)$. The level planarity testing (and embedding) problem is known to be solvable in linear time [24], although a sequence of incomplete characterizations by forbidden subgraphs [15,18] (see also [13]) has revealed that the problem is not yet fully understood. The similarity of the level planarity testing problem with the strip planarity testing problem is evident: They have the same input, they both require planar drawings with y -monotone edges, and they both

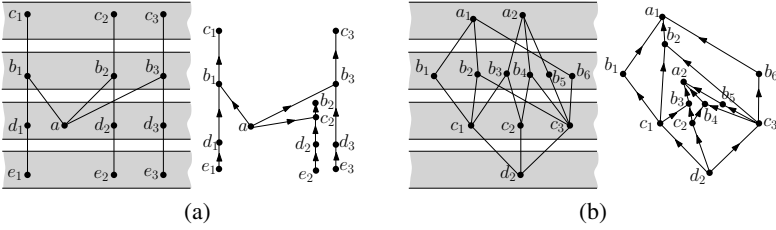


Fig. 2. Two negative instances (G_1, γ_1) (a) and (G_2, γ_2) (b) whose associated directed graphs are upward planar, where G_1 is a tree and G_2 is a subdivision of a triconnected plane graph

constrain the vertices to lie in specific regions of the plane; they only differ for the fact that such regions are horizontal lines in one case, and horizontal strips in the other one. Clearly the level planarity of an instance (G, γ) is a sufficient condition for the strip planarity of (G, γ) , as a level planar drawing is also a strip planar drawing. However, it is easy to construct instances (G, γ) that are strip planar and yet not level planar, even if we require that the instances are *strict*, i.e., no edge (u, v) is such that $\gamma(u) = \gamma(v)$. See Fig. 1(b). Also, the approach of [24] seems to be not applicable to test the strip planarity of a graph. Namely, Jünger et al. [24] visit the instance (G, γ) one level at a time, representing with a PQ-tree [7] the possible orderings of the vertices in level i that are consistent with a level planar embedding of the subgraph of G induced by levels $\{1, 2, \dots, i\}$. However, when visiting an instance (G, γ) of the strip planarity testing problem one strip at a time, PQ-trees seem to be not powerful enough to represent the possible orderings of the vertices in strip i that are consistent with a strip planar embedding of the subgraph of G induced by strips $\{1, 2, \dots, i\}$.

Strip Planarity and Upward Planarity. The *upward planarity testing* problem asks whether a given directed graph \vec{G} admits an *upward planar drawing*, i.e., a drawing which is planar and such that each edge is represented by a curve monotonically increasing in the y -direction, according to its orientation. Testing the upward planarity of a directed graph \vec{G} is an \mathcal{NP} -hard problem [16], however it is polynomial-time solvable, e.g., if \vec{G} has a fixed embedding [5], or if it has a single-source [21]. A strict instance (G, γ) of the strip planarity testing problem naturally defines a directed graph \vec{G} , by directing an edge (u, v) of G from u to v if $\gamma(u) < \gamma(v)$. It is easy to argue that the upward planarity of \vec{G} is a necessary and not sufficient condition for the strip planarity of (G, γ) (see Figs 2(a) and 2(b)). Roughly speaking, in an upward planar drawing different parts of the graph are free to “nest” one into the other, while in a strip planar drawing, such a nesting is only allowed if coherent with the strip assignment.

In this paper, we show that the strip planarity testing problem is polynomial-time solvable for planar graphs with a fixed planar embedding. Our approach consists of performing a sequence of modifications to the input instance (G, γ) (such modifications consist mainly of insertions of graphs inside the faces of G) that ensure that the instance satisfies progressively stronger constraints while not altering its strip planarity. Eventually, the strip planarity of (G, γ) becomes equivalent to the upward planarity of its associated directed graph, which can be tested in polynomial time.

The paper is organized as follows. In Section 2 we give some preliminaries; in Section 3 we prove our result; finally, in Section 4 we conclude and present open problems. For space limitations, proofs are sketched or omitted; refer to [3] for complete proofs.

2 Preliminaries

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* (or *combinatorial 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 combinatorial embedding have the same faces. However, such drawings could still differ for their outer faces. A *plane embedding* of a graph G is a planar embedding of G together with a choice for its outer face. In this paper, we will assume all the considered graphs to have a prescribed plane embedding.

For the sake of simplicity of description, in the following we assume that the considered plane graphs are *2-connected*, unless otherwise specified. We will sketch in the conclusions how to extend our results to simply-connected and even non-connected plane graphs. We now define some concepts related to strip planarity.

An instance (G, γ) of the strip planarity testing problem is *strict* if it contains no intra-strip edge, where an edge (u, v) is *intra-strip* if $\gamma(u) = \gamma(v)$. An instance (G, γ) of strip planarity is *proper* if, for every edge (u, v) of G , it holds $\gamma(v) - 1 \leq \gamma(u) \leq \gamma(v) + 1$. Given any non-proper instance of strip planarity, one can replace every edge (u, v) such that $\gamma(u) = \gamma(v) + j$, for some $j \geq 2$, with a path $(v = u_1, u_2, \dots, u_{j+1} = u)$ such that $\gamma(u_{i+1}) = \gamma(u_i) + 1$, for every $1 \leq i \leq j$, thus obtaining a proper instance (G', γ') of the strip planarity testing problem. It is easy to argue that (G, γ) is strip planar if and only if (G', γ') is strip planar. In the following, we will assume all the considered instances of the strip planarity testing problem to be proper.

Let (G, γ) be an instance of the strip planarity testing problem. A path (u_1, \dots, u_j) in G is *monotone* if $\gamma(u_i) = \gamma(u_{i-1}) + 1$, for every $2 \leq i \leq j$. For any face f in G , we denote by C_f the simple cycle delimiting the border of f . Let f be a face of G , let u be a vertex incident to f , and let v and z be the two neighbors of u on C_f . We say that u is a *local minimum* for f if $\gamma(v) = \gamma(z) = \gamma(u) + 1$, and it is a *local maximum* for f if $\gamma(v) = \gamma(z) = \gamma(u) - 1$. Also, we say that u is a *global minimum* for f (a *global maximum* for f) if $\gamma(w) \geq \gamma(u)$ (resp. $\gamma(w) \leq \gamma(u)$), for every vertex w incident to f . A global minimum u_m and a global maximum u_M for a face f are *consecutive* in f if no global minimum and no global maximum exists in one of the two paths connecting u_m and u_M in C_f . A local minimum u_m and a local maximum u_M for a face f are *visible* if one of the paths P connecting u_m and u_M in C_f is such that, for every vertex u of P , it holds $\gamma(u_m) < \gamma(u) < \gamma(u_M)$.

Definition 1. *An instance (G, γ) of the strip planarity problem is quasi-jagged if it is strict and if, for every face f of G and for any two visible local minimum u_m and local maximum u_M for f , one of the two paths connecting u_m and u_M in C_f is monotone.*

Definition 2. An instance (G, γ) of the strip planarity problem is jagged if it is strict and if, for every face f of G , any local minimum for f is a global minimum for f , and every local maximum for f is a global maximum for f .

3 How to Test Strip Planarity

In this section we show an algorithm to test strip planarity.

3.1 From a General Instance to a Strict Instance

In this section we show how to reduce a general instance of the strip planarity testing problem to an equivalent strict instance.

Lemma 1. Let (G, γ) be an instance of the strip planarity testing problem. Then, there exists a polynomial-time algorithm that either constructs an equivalent strict instance (G^*, γ^*) or decides that (G, γ) is not strip planar.

Consider any intra-strip edge (u, v) in G , if it exists. We distinguish two cases.

In *Case 1*, (u, v) is an edge of a 3-cycle (u, v, z) that contains vertices in its interior in G . Observe that, $\gamma(u) - 1 \leq \gamma(z) \leq \gamma(u) + 1$. Denote by G' the plane subgraph of G induced by the vertices lying outside cycle (u, v, z) together with u, v , and z (this graph might coincide with cycle (u, v, z) if such a cycle delimits the outer face of G); also, denote by G'' the plane subgraph of G induced by the vertices lying inside cycle (u, v, z) together with u, v , and z . Also, let $\gamma'(x) = \gamma(x)$, for every vertex x in G' , and let $\gamma''(x) = \gamma(x)$, for every vertex x in G'' . We have the following:

Claim 1. (G, γ) is strip planar if and only if (G', γ') and (G'', γ'') are both strip planar.

The strip planarity of (G'', γ'') can be tested in linear time as follows.

If $\gamma''(z) = \gamma''(u)$, then (G'', γ'') is strip planar if and only if $\gamma''(x) = \gamma''(u)$ for every vertex x of G'' (such a condition can clearly be tested in linear time). For the necessity, 3-cycle (u, v, z) is entirely drawn in $\gamma''(u)$, hence all the internal vertices of G'' have to be drawn inside $\gamma''(u)$ as well. For the sufficiency, G'' has a plane embedding by assumption, hence any planar y -monotone drawing (e.g. a straight-line drawing where no two vertices have the same y -coordinate) respecting such an embedding and contained in $\gamma''(u)$ is a strip planar drawing of (G'', γ'') .

If $\gamma''(z) = \gamma''(u) - 1$ (the case in which $\gamma''(z) = \gamma''(u) + 1$ is analogous), then we argue as follows: First, a clustered graph $C(G'', T)$ can be defined such that T consists of two clusters μ and ν , respectively containing every vertex x of G'' such that $\gamma''(x) = \gamma''(u) - 1$, and every vertex x of G'' such that $\gamma''(x) = \gamma''(u)$. We show that (G'', γ'') is strip planar if and only if $C(G'', T)$ is c -planar. For the necessity, it suffices to observe that a strip planar drawing of (G'', γ'') is also a c -planar drawing of $C(G'', T)$. For the sufficiency, if $C(G'', T)$ admits a c -planar drawing, then it also admits a c -planar *straight-line* drawing $\Gamma(C)$ in which the regions $R(\mu)$ and $R(\nu)$ representing μ and ν , respectively, are *convex* [2,12]. Assuming w.l.o.g. up to a rotation of $\Gamma(C)$ that $R(\mu)$ and $R(\nu)$ can be separated by a horizontal line, we have that disjoint

horizontal strips can be drawn containing $R(\mu)$ and $R(\nu)$. Slightly perturbing the positions of the vertices so that no two of them have the same y -coordinate ensures that the edges are y -monotone, thus resulting in a strip planar drawing of (G'', γ'') . Finally, the c -planarity of a clustered graph containing two clusters can be decided in linear time, as independently proved by Biedl et al. [6] and by Hong and Nagamochi [19].

In *Case 2*, a 3-cycle (u, v, z) exists that contains no vertices in its interior in G . Then, *contract* (u, v) , that is, identify u and v to be the same vertex w , whose incident edges are all the edges incident to u and v , except for (u, v) ; the clockwise order of the edges incident to w is: All the edges that used to be incident to u in the same clockwise order starting at (u, v) , and then all the edges that used to be incident to v in the same clockwise order starting at (v, u) . Denote by G' the resulting graph. Since G is plane, G' is plane; since G contains no 3-cycle (u, v, z) that contains vertices in its interior, G' is simple. Let $\gamma'(x) = \gamma(x)$, for every vertex $x \neq u, v$ in G , and let $\gamma'(w) = \gamma(u)$. We have the following.

Claim 2. (G', γ') is strip planar if and only if (G, γ) is strip planar.

Claims 1 and 2 imply Lemma 1. Namely, if (G, γ) has no intra-strip edge, there is nothing to prove. Otherwise, (G, γ) has an intra-strip edge (u, v) , hence either Case 1 or Case 2 applies. If Case 2 applies to (G, γ) , then an instance (G', γ') is obtained in linear time containing one less vertex than (G, γ) . By Claim 2, (G', γ') is equivalent to (G, γ) . Otherwise, Case 1 applies to (G, γ) . Then, either the non-strip planarity of (G, γ) is deduced (if (G'', γ'') is not strip planar), or an instance (G', γ') is obtained containing at least one less vertex than (G, γ) (if (G'', γ'') is strip planar). By Claim 1, (G', γ') is equivalent to (G, γ) . The repetition of such an argument either leads to conclude in polynomial time that (G, γ) is not strip planar, or leads to construct in polynomial time a strict instance (G^*, γ^*) of strip planarity equivalent to (G, γ) .

3.2 From a Strict Instance to a Quasi-Jagged Instance

In this section we show how to reduce a strict instance of the strip planarity testing problem to an equivalent quasi-jagged instance. Again, for the sake of simplicity of description, we assume that every considered instance (G, γ) is 2-connected.

Lemma 2. *Let (G, γ) be a strict instance of the strip planarity testing problem. Then, there exists a polynomial-time algorithm that constructs an equivalent quasi-jagged instance (G^*, γ^*) of the strip planarity testing problem.*

Consider any face f of G containing two visible local minimum and maximum u_m and u_M , respectively, such that no path connecting u_m and u_M in C_f is monotone. Insert a monotone path connecting u_m and u_M inside f . Denote by (G^+, γ^+) the resulting instance of the strip planarity testing problem. We have the following claim:

Claim 3. (G^+, γ^+) is strip planar if and only if (G, γ) is strip planar.

Proof Sketch: The necessity is trivial. For the sufficiency, consider any strip planar drawing Γ of (G, γ) . Denote by P the path connecting u_m and u_M along C_f and such

that $\gamma(u_m) < \gamma(v) < \gamma(u_M)$ holds for every internal vertex v of P . Because of the existence of some parts of the graph that “intermingle” with P , it might not be possible to draw a y -monotone curve inside f connecting u_m and u_M in Γ . Thus, a part of Γ has to be horizontally shrunk, so that it moves “far away” from P , thus allowing for the monotone path connecting u_m and u_M to be drawn as a y -monotone curve inside f . This results in a strip planar drawing of (G^+, γ^+) . \square

Claim 3 implies Lemma 2, as proved in the following.

First, the repetition of the above described augmentation leads to a quasi-jagged instance (G^*, γ^*) . In fact, whenever the augmentation is performed, the number of triples (v_m, v_M, g) such that vertices v_m and v_M are visible local minimum and maximum for face g , respectively, and such that both paths connecting v_m and v_M along C_f are not monotone decreases by 1, thus eventually the number of such triples is zero, and the instance is quasi-jagged.

Second, (G^*, γ^*) can be constructed from (G, γ) in polynomial time. Namely, the number of pairs of visible local minima and maxima for a face g of G is polynomial in the number of vertices of g . Hence, the number of triples (v_m, v_M, g) such that vertices v_m and v_M are visible local minimum and maximum for face g , over all faces of G , is polynomial in n . Since a linear number of vertices are introduced in G whenever the augmentation described above is performed, it follows that the construction of (G^*, γ^*) from (G, γ) can be accomplished in polynomial time.

Third, (G^*, γ^*) is an instance of the strip planarity testing problem that is equivalent to (G, γ) . This directly comes from repeated applications of Claim 3.

3.3 From a Quasi-Jagged Instance to a Jagged Instance

In this section we show how to reduce a quasi-jagged instance of the strip planarity testing problem to an equivalent jagged instance. Again, for the sake of simplicity of description, we assume that every considered instance (G, γ) is 2-connected.

Lemma 3. *Let (G, γ) be a quasi-jagged instance of the strip planarity testing problem. Then, there exists a polynomial-time algorithm that constructs an equivalent jagged instance (G^*, γ^*) of the strip planarity testing problem.*

Consider any face f of G that contains some local minimum or maximum which is not a global minimum or maximum for f , respectively. Assume that f contains a local minimum v which is not a global minimum for f . The case in which f contains a local maximum which is not a global maximum for f can be discussed analogously. Denote by u (denote by z) the first global minimum or maximum for f that is encountered when walking along C_f starting at v while keeping f to the left (resp. to the right).

We distinguish two cases, namely the case in which u is a global minimum for f and z is a global maximum for f (Case 1), and the case in which u and z are both global maxima for f (Case 2). The case in which u is a global maximum for f and z is a global minimum for f , and the case in which u and z are both global minima for f can be discussed symmetrically.

In *Case 1*, denote by Q the path connecting u and z in C_f and containing v . Consider the internal vertex v' of Q that is a local minimum for f and that is such that

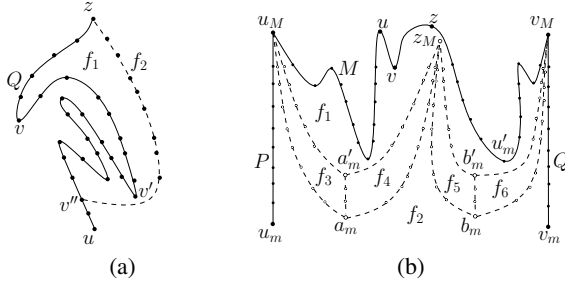


Fig. 3. Augmentation of (G, γ) inside a face f in: (a) Case 1 and (b) Case 2

$\gamma(v') = \min_{u'} \gamma(u')$ among all the internal vertices u' of Q that are local minima for f . Traverse Q starting from u , until a vertex v'' is found with $\gamma(v'') = \gamma(v')$. Notice that, the subpath of Q between u and v'' is monotone. Insert a monotone path connecting v'' and z inside f . See Fig. 3(a). Denote by (G^+, γ^+) the resulting instance of the strip planarity testing problem. We have the following claim:

Claim 4. *Suppose that Case 1 is applied to a quasi-jagged instance (G, γ) to construct an instance (G^+, γ^+) . Then, (G^+, γ^+) is strip planar if and only if (G, γ) is strip planar. Also, (G^+, γ^+) is quasi-jagged.*

Proof Sketch: The necessity is trivial. For the sufficiency, consider any strip planar drawing Γ of (G, γ) . First, Γ is modified so that v'' has y -coordinate smaller than every local minimum of Q different from u . Then, a y -monotone curve can be drawn inside f connecting v'' and z , thus resulting in a strip planar drawing of (G^+, γ^+) . \square

In Case 2, denote by M a maximal path that is part of C_f , whose end-vertices are two global maxima u_M and v_M for f , that contains v in its interior, and that does not contain any global minimum in its interior. By the assumptions of Case 2, such a path exists. Assume, w.l.o.g., that face f is to the right of M when walking along M starting at u_M towards v_M . Possibly $u_M = u$ and/or $v_M = z$. Let u_m (v_m) be the global minimum for f such that u_m and u_M (resp. v_m and v_M) are consecutive global minimum and maximum for f . Possibly, $u_m = v_m$. Denote by P the path connecting u_m and u_M along C_f and not containing v . Also, denote by Q the path connecting v_m and v_M along C_f and not containing v . Since M contains a local minimum among its internal vertices, and since (G, γ) is quasi-jagged, it follows that P and Q are monotone.

Insert the plane graph $A(u_M, v_M, f)$ depicted by white circles and dashed lines in Fig. 3(b) inside f . Consider a local minimum $u'_m \in M$ for f such that $\gamma(u'_m) = \min_{v'_m} \gamma(v'_m)$ among the local minima v'_m for f in M . Set $\gamma(z_M) = \gamma(u_M)$, set $\gamma(a_m) = \gamma(b_m) = \gamma(u_m)$, and set $\gamma(a'_m) = \gamma(b'_m) = \gamma(u'_m)$. The dashed lines connecting a_m and u_M , connecting a'_m and u_M , connecting a_m and z_M , connecting a'_m and z_M , connecting b_m and z_M , connecting b'_m and z_M , connecting b_m and v_M , connecting b'_m and v_M , connecting a_m and a'_m , and connecting b_m and b'_m represent monotone paths. Denote by (G^+, γ^+) the resulting instance of the strip planarity testing problem. We have the following claim:

Claim 5. *Suppose that Case 2 is applied to a quasi-jagged instance (G, γ) to construct an instance (G^+, γ^+) . Then, (G^+, γ^+) is strip planar if and only if (G, γ) is strip planar. Also, (G^+, γ^+) is quasi-jagged.*

Proof Sketch: The necessity is trivial. For the sufficiency, consider any strip planar drawing Γ of (G, γ) . If P is to the left of Q , then a region R is defined as the region delimited by P , by M , by Q , and by the horizontal line delimiting $\gamma(u_m)$ from above. Then, the part of Γ that lies inside R is redrawn so that it lies inside a region $R_Q \subset R$ arbitrarily close to Q . Such a redrawing “frees” space for the drawing of $A(u_M, v_M, f)$ inside f , which results in a strip planar drawing of (G^+, γ^+) . If P is to the right of Q , then M might “wobble” to the right of P and to the left of Q . Thus, we first horizontally shrink a part of Γ that “intermingles” with P and Q , and we then draw $A(u_M, v_M, f)$ using its four monotone paths connecting global minima with global maxima in order to “circumvent” M . This results in a strip planar drawing of (G^+, γ^+) . \square

Claims 4–5 imply Lemma 3, as proved in the following.

First, we prove that the repetition of the above described augmentation leads to a jagged instance (G^*, γ^*) of the strip planarity testing problem. For an instance (G, γ) and for a face g of G , denote by $n(g)$ the number of vertices that are local minima for g but not global minima for g , plus the number of vertices that are local maxima for g but not global maxima for g . Also, let $n(G) = \sum_g n(g)$, where the sum is over all faces g of G . We claim that, when one of the augmentations of Cases 1 and 2 is performed and instance (G, γ) is transformed into an instance (G^+, γ^+) , we have $n(G^+) \leq n(G) - 1$. The claim implies that eventually $n(G^*) = 0$, hence (G^*, γ^*) is jagged.

We prove the claim. When a face f of G is augmented as in Case 1 or in Case 2, for each face $g \neq f$ and for each vertex u incident to g , vertex u is a local minimum, a local maximum, a global minimum, or a global maximum for g in (G^+, γ^+) if and only if it is a local minimum, a local maximum, a global minimum, or a global maximum for g in (G, γ) , respectively. Hence, it suffices to prove that $\sum n(f_i) \leq n(f) - 1$, where the sum is over all the faces f_i that are created from the augmentation inside f .

Suppose that Case 1 is applied to insert a monotone path between vertices v' and z inside f . Such an insertion splits f into two faces, which we denote by f_1 and f_2 , as in Fig. 3(a). Face f_2 is delimited by two monotone paths, hence $n(f_2) = 0$. Every vertex inserted into f is neither a local maximum nor a local minimum for f_1 . As a consequence, no vertex x exists such that x contributes to $n(f_1)$ and x does not contribute to $n(f)$. Further, vertex v' is a global minimum for f_1 , by construction, and it is a local minimum but not a global minimum for f . Hence, v' contributes to $n(f)$ and does not contribute to $n(f_1)$. It follows that $n(f_1) + n(f_2) \leq n(f) - 1$.

Suppose that Case 2 is applied to insert plane graph $A(u_M, v_M, f)$ inside face f . Such an insertion splits f into six faces, which are denoted by f_1, \dots, f_6 , as in Fig. 3(b). Every vertex of $A(u_M, v_M, f)$ incident to a face f_i , for some $1 \leq i \leq 6$, is either a global maximum for f_i , or a global minimum for f_i , or it is neither a local maximum nor a local minimum for f_i . As a consequence, no vertex x exists such that x contributes to some $n(f_i)$ and x does not contribute to $n(f)$. Further, for each vertex x that contributes to $n(f)$, there exists at most one face f_i such that x contributes to $n(f_i)$. Finally, vertex u'_m of M is a global minimum for f_1 , by construction, and it is a local minimum but

not a global minimum for f . Hence, u'_m contributes to $n(f)$ and does not contribute to $n(f_i)$, for any $1 \leq i \leq 6$. It follows that $\sum_{i=1}^6 n(f_i) \leq n(f) - 1$.

Second, (G^*, γ^*) can be constructed from (G, γ) in polynomial time. Namely, the number of local minima (maxima) for a face f that are not global minima (maxima) for f is at most the number of vertices of f . Hence, the number of such minima and maxima over all the faces of G , which is equal to $n(G)$, is linear in n . Since a linear number of vertices are introduced in G whenever the augmentation described above is performed, and since the augmentation is performed at most $n(G)$ times, it follows that the construction of (G^*, γ^*) can be accomplished in polynomial time.

Third, (G^*, γ^*) is an instance of the strip planarity testing problem that is equivalent to (G, γ) . This directly comes from repeated applications of Claims 4 and 5.

3.4 Testing Strip Planarity for Jagged Instances

In this section we show how to test in polynomial time whether a jagged instance (G, γ) of the strip planarity testing problem is strip planar. Recall that the associated directed graph of (G, γ) is the directed plane graph \vec{G} obtained from (G, γ) by orienting each edge (u, v) in G from u to v if and only if $\gamma(v) = \gamma(u) + 1$. We have the following:

Lemma 4. *A jagged instance (G, γ) of the strip planarity testing problem is strip planar if and only if the associated directed graph \vec{G} of (G, γ) is upward planar.*

Proof Sketch: The necessity is trivial. For the sufficiency, we first insert dummy edges in \vec{G} to augment it to a *plane st-digraph* \vec{G}_{st} , which is an upward planar directed graph with exactly one source s and one sink t incident to its outer face [11]. Each face f of \vec{G}_{st} consists of two monotone paths, called *left path* and *right path*, where the left path has f to the right when traversing it from its source to its sink. The inserted dummy edges only connect two sources or two sinks of each face of \vec{G} . Since (G, γ) is jagged, the end-vertices of each dummy edge are in the same strip.

We divide the plane into k horizontal strips. We compute an upward planar drawing of \vec{G}_{st} starting from a y -monotone drawing of the leftmost path of \vec{G}_{st} and adding to the drawing one face at a time, in an order corresponding to any linear extension of the partial order of the faces induced by the directed dual graph of \vec{G}_{st} [11]. When a face is added to the drawing, its left path is already drawn as a y -monotone curve. We draw the right path of f as a y -monotone curve in which each vertex u lies inside strip $\gamma(u)$, hence the rightmost path of the graph in the current drawing is always represented by a y -monotone curve. A strip planar drawing of (G, γ) can be obtained from the drawing of \vec{G}_{st} by removing the dummy edges. \square

We thus obtain the following:

Theorem 1. *The strip planarity testing problem can be solved in polynomial time for instances (G, γ) such that G is a plane graph.*

Proof: By Lemmata 1–3, it is possible to reduce in polynomial time any instance of the strip planarity testing problem to an equivalent jagged instance (G, γ) . By Lemma 4, (G, γ) is strip planar if and only if the associated directed plane graph \vec{G} of (G, γ) is upward planar. Finally, by the results of Bertolazzi et al. [5], the upward planarity of \vec{G} can be tested in polynomial time. \square

4 Conclusions

In this paper, we introduced the strip planarity testing problem and showed how to solve it in polynomial time if the input graph is 2-connected and has a prescribed plane embedding. We now sketch how to remove the 2-connectivity requirement.

Suppose that the input graph (G, γ) is simply-connected (possibly not 2-connected). The algorithmic steps are the same. The transformation of a general instance into a strict instance is exactly the same. The transformation of a strict instance into a quasi-jagged instance has some differences with respect to the 2-connected case. In fact, the visibility between local minima and maxima for a face f of G is redefined with respect to *occurrences* of such minima and maxima along f . Thus, the goal of such a transformation is to create an instance in which, for every face f and for every pair of visible occurrences $\sigma_i(u_m)$ and $\sigma_j(u_M)$ of a local minimum u_m and a local maximum u_M for f , respectively, there is a monotone path between $\sigma_i(u_m)$ and $\sigma_j(u_M)$ in C_f . This is done with the same techniques as in Claim 3. The transformation of a quasi-jagged instance into a jagged instance is almost the same as in the 2-connected case, except that the 2-connected components of G inside a face f have to be suitably squeezed along the monotone paths of f to allow for a drawing of a monotone path between v'' and z or for a drawing of plane graph $A(u_M, v_M, f)$. This is done with the same techniques as in Claims 4 and 5. Finally, the proof of the equivalence between the strip planarity of a jagged instance and the upward planarity of its associated directed graph holds as it is.

Suppose now that the input graph (G, γ) is not connected. Test individually the strip planarity of each connected component of (G, γ) . If one of the tests fails, then (G, γ) is not strip planar. Otherwise, construct a strip planar drawing of each connected component of (G, γ) . Place the drawings of the connected components containing edges incident to the outer face of G side by side. Repeatedly insert connected components in the internal faces of the currently drawn graph (G', γ) as follows. If a connected component (G_i, γ) of (G, γ) has to be placed inside an internal face f of (G', γ) , check whether $\gamma(u_M) \leq \gamma(u_M^f)$ and whether $\gamma(u_m) \geq \gamma(u_m^f)$, where u_M (u_m) is a vertex of (G_i, γ) such that $\gamma(u_M)$ is maximum (resp. $\gamma(u_m)$ is minimum) among the vertices of G_i , and where u_M^f (u_m^f) is a vertex of C_f such that $\gamma(u_M^f)$ is maximum (resp. $\gamma(u_m^f)$ is minimum) among the vertices of C_f . If the test fails, then (G, γ) is not strip planar. Otherwise, using a technique analogous to the one of Claim 3, a strip planar drawing of (G', γ) can be modified so that two consecutive global minimum and maximum for f can be connected by a y -monotone curve \mathcal{C} inside f . Suitably squeezing a strip planar drawing of (G_i, γ) and placing it arbitrarily close to \mathcal{C} provides a strip planar drawing of $(G' \cup G_i, \gamma)$. Repeating such an argument leads either to conclude that (G, γ) is not strip planar, or to construct a strip planar drawing of (G, γ) .

The main question raised by this paper is whether the strip planarity testing problem can be solved in polynomial time or is rather \mathcal{NP} -hard for graphs without a prescribed plane embedding. The problem is intriguing even if the input graph is a tree.

References

1. Angelini, P., Di Battista, G., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. In: Charikar, M. (ed.) SODA 2010, pp. 202–221. ACM (2010)

2. Angelini, P., Frati, F., Kaufmann, M.: Straight-line rectangular drawings of clustered graphs. *Discrete & Computational Geometry* 45(1), 88–140 (2011)
3. Angelini, P., Da Lozzo, G., Di Battista, G., Frati, F.: Strip planarity testing of embedded planar graphs. ArXiv e-prints 1309.0683 (September 2013)
4. Bachmaier, C., Brandenburg, F.J., Forster, M.: Radial level planarity testing and embedding in linear time. *JGAA* 9(1), 53–97 (2005)
5. Bertolazzi, P., Di Battista, G., Liotta, G., Mannino, C.: Upward drawings of triconnected digraphs. *Algorithmica* 12(6), 476–497 (1994)
6. Biedl, T.C., Kaufmann, M., Mutzel, P.: Drawing planar partitions II: HH-drawings. In: Hromkovič, J., Sýkora, O. (eds.) *WG 1998*. LNCS, vol. 1517, pp. 124–136. Springer, Heidelberg (1998)
7. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.* 13(3), 335–379 (1976)
8. Cortese, P.F., Di Battista, G., Patrignani, M., Pizzonia, M.: Clustering cycles into cycles of clusters. *JGAA* 9(3), 391–413 (2005)
9. Cortese, P.F., Di Battista, G., Patrignani, M., Pizzonia, M.: On embedding a cycle in a plane graph. *Discrete Mathematics* 309(7), 1856–1869 (2009)
10. Di Battista, G., Frati, F.: Efficient c-planarity testing for embedded flat clustered graphs with small faces. *JGAA* 13(3), 349–378 (2009)
11. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.* 61, 175–198 (1988)
12. Eades, P., Feng, Q., Lin, X., Nagamochi, H.: Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Algorithmica* 44(1), 1–32 (2006)
13. Estrella-Balderrama, A., Fowler, J.J., Kobourov, S.G.: On the characterization of level planar trees by minimal patterns. In: Eppstein, D., Gansner, E.R. (eds.) *GD 2009*. LNCS, vol. 5849, pp. 69–80. Springer, Heidelberg (2010)
14. Forster, M., Bachmaier, C.: Clustered level planarity. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) *SOFSEM 2004*. LNCS, vol. 2932, pp. 218–228. Springer, Heidelberg (2004)
15. Fowler, J.J., Kobourov, S.G.: Minimum level nonplanar patterns for trees. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) *GD 2007*. LNCS, vol. 4875, pp. 69–75. Springer, Heidelberg (2008)
16. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.* 31(2), 601–625 (2001)
17. Gutwenger, C., Klein, K., Mutzel, P.: Planarity testing and optimal edge insertion with embedding constraints. *JGAA* 12(1), 73–95 (2008)
18. Healy, P., Kuusik, A., Leipert, S.: A characterization of level planar graphs. *Discrete Mathematics* 280(1-3), 51–63 (2004)
19. Hong, S.H., Nagamochi, H.: Two-page book embedding and clustered graph planarity. Tech. Report 2009-004, Dept. of Applied Mathematics & Physics, Kyoto University (2009)
20. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. *J. ACM* 21(4), 549–568 (1974)
21. Hutton, M.D., Lubiw, A.: Upward planarity testing of single-source acyclic digraphs. *SIAM J. Comput.* 25(2), 291–311 (1996)
22. Jelínek, V., Kratochvíl, J., Rutter, I.: A kuratowski-type theorem for planarity of partially embedded graphs. *Comput. Geom. Theory Appl.* 46(4), 466–492 (2013)
23. Jelínková, E., Kára, J., Kratochvíl, J., Pergel, M., Suchý, O., Vyskocil, T.: Clustered planarity: Small clusters in cycles and Eulerian graphs. *JGAA* 13(3), 379–422 (2009)
24. 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)
25. Schaefer, M.: Toward a theory of planarity: Hanani-tutte and planarity variants. In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 162–173. Springer, Heidelberg (2013)

Morphing Planar Graph Drawings Efficiently^{*}

Patrizio Angelini¹, Fabrizio Frati², Maurizio Patrignani¹, and Vincenzo Roselli¹

¹ Engineering Department, Roma Tre University, Italy
{angelini, patrignani, roselli}@dia.uniroma3.it

² School of Information Technologies, The University of Sydney, Australia
brillo@it.usyd.edu.au

Abstract. A morph between two straight-line planar drawings of the same graph is a continuous transformation from the first to the second drawing such that planarity is preserved at all times. Each step of the morph moves each vertex at constant speed along a straight line. Although the existence of a morph between any two drawings was established several decades ago, only recently it has been proved that a polynomial number of steps suffices to morph any two planar straight-line drawings. Namely, at SODA 2013, Alamdari *et al.* [1] proved that any two planar straight-line drawings of a planar graph can be morphed in $O(n^4)$ steps, while $O(n^2)$ steps suffice if we restrict to maximal planar graphs.

In this paper, we improve upon such results, by showing an algorithm to morph any two planar straight-line drawings of a planar graph in $O(n^2)$ steps; further, we show that a morph with $O(n)$ steps exists between any two planar straight-line drawings of a series-parallel graph.

1 Introduction

A *planar morph* between two planar drawings of the same plane graph is a continuous transformation from the first drawing to the second one such that planarity is preserved at all times. The problem of deciding whether a planar morph exists for any two drawings of any graph dates back to 1944, when Cairns [7] proved that any two straight-line drawings of a maximal planar graph can be morphed one into the other while maintaining planarity. In 1981, Grünbaum and Shephard [10] introduced the concept of *linear morph*, that is a continuous transformation in which each vertex moves at uniform speed along a straight-line trajectory. With this further requirement, however, planarity cannot always be maintained for any pair of drawings. Hence, the problem has been subsequently studied in terms of the existence of a sequence of linear morphs, also called *morphing steps*, transforming a drawing into another while maintaining planarity. The first result in this direction is the one of Thomassen [13], who proved that a sequence of morphing steps always exists between any two straight-line drawings of the same plane graph. Further, if the two input drawings are convex, this property is maintained throughout the morph, as well. However, the number of morphing steps used by the algorithm of Thomassen might be exponential in the number of vertices.

^{*} Part of the research was conducted in the framework of ESF project 10-EuroGIGA-OP-003 GraDR “Graph Drawings and Representations”.

Recently, the problem of computing planar morphs gained increasing research attention. The case in which edges are not required to be straight-line segments has been addressed in [11], while morphs between orthogonal graph drawings preserving planarity and orthogonality have been explored in [12]. Morphs preserving more general edge directions have been considered in [6]. Also, the problem of “topological morphing”, in which the planar embedding is allowed to change, has been addressed in [2].

In a paper appeared at SODA 2013, Alamdari *et al.* [1] tackled again the original setting in which edges are straight-line segments and linear morphing steps are required. Alamdari *et al.* presented the first morphing algorithms with a polynomial number of steps in this setting. Namely, they presented an algorithm to morph straight-line planar drawings of maximal plane graphs with $O(n^2)$ steps and of general plane graphs with $O(n^4)$ steps, where n is the number of vertices of the graph.

In this paper we improve upon the result of Alamdari *et al.* [1], providing a more efficient algorithm to morph general plane graphs. Namely, our algorithm uses $O(n^2)$ linear morphing steps. Further, we provide a morphing algorithm with a linear number of steps for a non-trivial class of planar graphs, namely series-parallel graphs. These two main results are summarized in the following theorems.

Theorem 1. *Let Γ_a and Γ_b be two drawings of the same plane series-parallel graph G . There exists a morph $\langle \Gamma_a, \dots, \Gamma_b \rangle$ with $O(n)$ steps transforming Γ_a into Γ_b .*

Theorem 2. *Let Γ_s and Γ_t be two drawings of the same plane graph G . There exists a morph $\langle \Gamma_s, \dots, \Gamma_t \rangle$ with $O(n^2)$ steps transforming Γ_s into Γ_t .*

The rest of the paper is organized as follows. Section 2 contains preliminaries and basic terminology. Section 3 describes an algorithm to morph series-parallel graphs. Section 4 describes an algorithm to morph plane graphs. Section 5 provides geometric details for the morphs described in Sections 3 and 4. Finally, Section 6 contains conclusions and open problems. Because of space limitations, some proofs are omitted or sketched. Full proofs can be found in the extended version of the paper [4].

2 Preliminaries

A *straight-line planar drawing* Γ (in the following simply *drawing*) of a graph $G(V, E)$ maps vertices in V to distinct points of the plane and edges in E to non-intersecting open straight-line segments between their end-vertices. Given a vertex v of a graph G , we denote by $\deg(v)$ the *degree* of v in G , that is, the number of vertices adjacent to v . A planar drawing Γ partitions the plane into connected regions called *faces*. The unbounded face is the *external face*. Also, Γ determines a clockwise order of the edges incident to each vertex. Two planar drawings are *equivalent* if they determine the same clockwise ordering of the incident edges around each vertex and if they have the same external face. A *planar embedding* is an equivalence class of planar drawings. A *plane graph* is a planar graph with a given planar embedding.

A *series-parallel graph* G is a planar graph that does not contain the complete graph on four vertices as a minor. A *plane series-parallel graph* is a graph together with a planar embedding. Let G be a plane biconnected series-parallel graph and let e be an edge

incident to its outer face. Graph G has a unique *decomposition tree* T_e rooted at e having nodes of three types: Q-, S-, and P-nodes. A Q-node represents a single edge, while an S-node (a P-node) μ represents a series (a parallel, respectively) composition of the subgraphs associated to the subtrees of T_e rooted at the children of μ . An embedding of G naturally induces an ordering for the children of each node of T_e .

A (*linear*) *morphing step* $\langle \Gamma_1, \Gamma_2 \rangle$, also referred to as *linear morph*, of two straight-line planar drawings Γ_1 and Γ_2 of a plane graph G is a continuous transformation of Γ_1 into Γ_2 such that all the vertices simultaneously start moving from their positions in Γ_1 and, moving along a straight-line trajectory, simultaneously stop at their positions in Γ_2 so that no crossing occurs between any two edges during the transformation. A *morph* $\langle \Gamma_s, \dots, \Gamma_t \rangle$ of two straight-line planar drawings Γ_s into Γ_t of a plane graph G is a finite sequence of morphing steps that transforms Γ_s into Γ_t . Let u and w be two vertices of G such that edge (u, w) belongs to G and let Γ be a straight-line planar drawing of G . The *contraction* of u onto w results in (i) a graph $G' = G/(u, w)$ not containing u and such that each edge (u, x) of G is replaced by an edge (w, x) in G' , and (ii) a straight-line drawing Γ' of G' such that each vertex different from v is mapped to the same point as in Γ . In the following, the contraction of an edge (u, w) will be only applied if the obtained drawing Γ' is planar. The *uncontraction* of u from w in Γ' yields a straight-line planar drawing Γ'' of G . A morph in which contractions are performed, possibly together with other morphing steps, is a *pseudo-morph*. Let v be a vertex of G and let G' be the graph obtained by removing v and its incident edges from G . Let Γ' be a planar straight-line drawing of G' . The *kernel* of v in Γ' is the set P of points such that straight-line segments can be drawn in Γ' connecting each point $p \in P$ to each neighbor of v in G without intersecting any edge in Γ' .

3 Morphing Series-Parallel Graph Drawings in $O(n)$ Steps

In this section we show an algorithm to compute a pseudo-morph between any two drawings of the same plane series-parallel graph G . In Section 3.1 we assume that G is biconnected, and in Section 3.2 we show how to remove this assumption, thus proving the following theorem.

Theorem 3. *Let Γ_a and Γ_b be two drawings of the same plane series-parallel graph G . There exists a pseudo-morph $\langle \Gamma_a, \dots, \Gamma_b \rangle$ with $O(n)$ steps transforming Γ_a into Γ_b .*

3.1 Biconnected Series-Parallel Graphs

Our approach consists of morphing any drawing Γ of a biconnected plane series-parallel graph G into a “canonical drawing” Γ^* of G in a linear number of steps. As a consequence, any two drawings Γ_1 and Γ_2 of G can be transformed one into the other in a linear number of steps, by morphing Γ_1 to Γ^* and Γ^* to Γ_2 .

A *canonical drawing* Γ^* of a biconnected plane series-parallel graph G is defined as follows. The decomposition tree T_e of G is traversed top-down and a suitable geometric region of the plane is assigned to each node μ of T_e ; such a region will contain the drawing of the series-parallel graph associated with μ . The regions assigned to the nodes

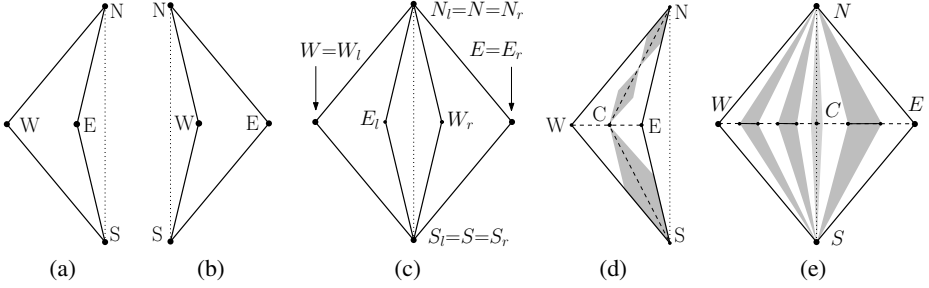


Fig. 1. (a) A left boomerang. (b) A right boomerang. (c) A diamond. (d) Diamonds inside a boomerang. (e) Boomerangs (and a diamond) inside a diamond.

of T_e are similar to those used in [5,3] to construct monotone drawings. Namely, we define three types of regions: Left boomerangs, right boomerangs, and diamonds. A *left boomerang* is a quadrilateral with vertices $N, E, S,$ and W such that E is inside triangle $\triangle(N, S, W)$, where $|NE| = |SE|$ and $|NW| = |SW|$ (see Fig. 1(a)). A *right boomerang* is defined symmetrically, with E playing the role of W , and vice versa (see Fig. 1(b)). A *diamond* is a convex quadrilateral with vertices $N, E, S,$ and W , where $|NW| = |NE| = |SW| = |SE|$. Observe that a diamond contains a left boomerang N_l, E_l, S_l, W_l and a right boomerang N_r, E_r, S_r, W_r , where $S = S_l = S_r, N = N_l = N_r, W = W_l,$ and $E = E_r$ (see Fig. 1(c)).

We assign boomerangs (either left or right, depending on the embedding of G) to S-nodes and diamonds to P- and Q-nodes, as follows.

First, consider the Q-node ρ corresponding to the root edge e of G . Draw edge e as a segment between points $(0, 1)$ and $(0, -1)$. Also, if ρ is adjacent to an S-node μ , then assign to μ the left boomerang $N = (0, 1), E = (-1, 0), S = (0, -1), W = (-2, 0)$ or the right boomerang $N = (0, 1), E = (2, 0), S = (0, -1), W = (1, 0)$, depending on the embedding of G ; if ρ is adjacent to a P-node μ , then associate to μ the diamond $N = (0, 1), E = (+2, 0), S = (0, -1), W = (-2, 0)$.

Then, consider each node μ of $T_e(G)$ according to a top-down traversal.

If μ is an S-node (see Fig. 1(d)), let N, E, S, W be the boomerang associated with it and let α be the angle \widehat{WNE} . We associate diamonds to the children $\mu_1, \mu_2, \dots, \mu_k$ of μ as follows. Consider the midpoint C of segment \overline{WE} . Subdivide \overline{NC} into $\lceil \frac{k}{2} \rceil$ segments with the same length and \overline{CS} into $\lfloor \frac{k}{2} \rfloor$ segments with the same length. Enclose each of such segments $\overline{N_i S_i}$, for $i = 1, \dots, k$, into a diamond N_i, E_i, S_i, W_i , with $\widehat{W_i N_i E_i} = \alpha$, and associate it with child μ_i of μ .

If μ is a P-node (see Fig. 1(e)), let N, E, S, W be the diamond associated with it. Associate boomerangs and diamonds to the children $\mu_1, \mu_2, \dots, \mu_k$ of μ as follows. If a child μ_l of μ is a Q-node, then left boomerangs are associated to μ_1, \dots, μ_{l-1} , right boomerangs are associated to μ_{l+1}, \dots, μ_k , and a diamond is associated to μ_l . Otherwise, right boomerangs are associated to all of $\mu_1, \mu_2, \dots, \mu_k$. We assume that a child μ_l of μ that is a Q-node exists, the description for the case in which no child of μ is a Q-node being similar and simpler. We describe how to associate left boomerangs to the

children $\mu_1, \mu_2, \dots, \mu_{l-1}$ of μ . Consider the midpoint C of segment \overline{WE} and consider $2l$ equidistant points $W = p_1, \dots, p_{2l} = C$ on segment \overline{WC} . Associate each child μ_i , with $i = 1, \dots, l-1$, to the quadrilateral $N_i = N, E_i = p_{2i}, S_i = S, W_i = p_{2i+1}$. Right boomerangs are associated to $\mu_{l+1}, \mu_{l+2}, \dots, \mu_k$ in a symmetric way. Finally, associate μ_l to any diamond such that $N_l = N, S_l = S, W_l$ is any point between C and E_{l-1} , and E_l is any point between C and W_{l+1} .

If μ is a Q-node, let N, E, S, W be the diamond associated with it. Draw the edge corresponding to μ as a straight-line segment between N and S .

Observe that the above described algorithm constructs a drawing of G , that we call the *canonical drawing* of G . We now argue that no two edges e_1 and e_2 intersect in the canonical drawing of G . Consider the lowest common ancestor ν of the Q-nodes τ_1 and τ_2 of T_e representing e_1 and e_2 , respectively. Also, consider the children ν_1 and ν_2 of ν such that the subtree of T_e rooted at ν_i contains τ_i , for $i = 1, 2$. Such children are associated with internally-disjoint regions of the plane. Since the subgraphs G_1 and G_2 of G corresponding to ν_1 and ν_2 , respectively, are entirely drawn inside such regions, it follows that e_1 and e_2 do not intersect except, possibly, at common endpoints.

In order to construct a pseudo-morph of a straight-line planar drawing $\Gamma(G)$ of G into its canonical drawing $\Gamma^*(G)$, we do the following: (i) We perform a contraction of a vertex v of G into a neighbor of v , hence obtaining a drawing $\Gamma(G')$ of a graph G' with $n-1$ vertices; (ii) we inductively construct a pseudo-morph from $\Gamma(G')$ to the canonical drawing $\Gamma^*(G')$ of G' ; and (iii) we uncontract v and perform a sequence of morphing steps to transform $\Gamma^*(G')$ into the canonical drawing $\Gamma^*(G)$ of G .

We describe the three steps in more detail.

Let $T_e(G)$ be the decomposition tree of G rooted at some edge e incident to the outer face of G . Consider a P-node ν such that the subtree of $T_e(G)$ rooted at ν does not contain any other P-node. This implies that all the children of ν , with the exception of at most one Q-node, are S-nodes whose children are Q-nodes. Hence, the series-parallel graph $G(\nu)$ associated to ν is composed of a set of paths connecting its poles s and t . Let p_1 and p_2 be two paths joining s and t and such that their union is a cycle \mathcal{C} not containing other vertices in its interior (see Fig. 2(a)). Such paths exist given that the “rest of the graph” with respect to ν is in the outer face of $G(\nu)$, since the root e of $T_e(G)$ is incident to the outer face of G . Internally triangulate \mathcal{C} by adding dummy edges (dashed edges of Fig. 2). Cycle \mathcal{C} and the added dummy edges yield a drawing of a biconnected outerplane graph O which, hence, has at least two vertices of degree two.

If there exists a vertex v with $\deg(v) = 2$ and $v \neq s, t$ (*Case 1*), then apply the following contraction. Assume that v belongs to p_2 . Since O is internally triangulated, both the neighbors v_1 and v_2 of v belong to p_2 , and they are joined by a dummy edge. We obtain $\Gamma(G')$ from $\Gamma(G)$ by contracting v onto one of its neighbors, while preserving planarity (see Figs. 2(a) and 2(b)). If p_2 contains more than two edges (*Case 1.1*), then p_2 is replaced in G' with a path p'_2 that contains edge (v_1, v_2) and does not contain vertex v . Otherwise, p_2 contains exactly two edges (v, v_1) and (v, v_2) . If there exists edge (v_1, v_2) in G (*Case 1.2*), then $G' = G \setminus \{v\}$. Finally, if edge (v_1, v_2) does not exist in G (*Case 1.3*), then p_2 is replaced in G' with edge (v_1, v_2) . Otherwise, the only two vertices of degree 2 in O are s and t (*Case 2*). In this case, one of the two vertices u_1 and u_2 of O adjacent to s has degree 3, say u_2 (since removing s and its incident

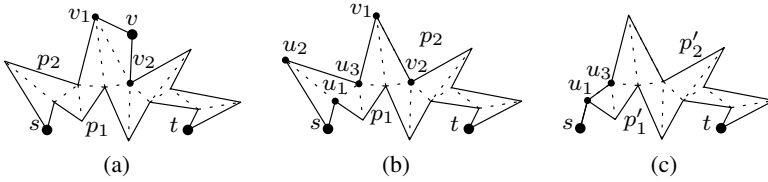


Fig. 2. The internally triangulated cycle \mathcal{C} formed by paths p_1 and p_2 . Dummy edges are drawn as dashed lines. (a–b) Vertex v of degree 2 can be contracted onto v_1 . (b–c) Vertex u_2 of degree 3 can be contracted onto u_1 .

edges from O yields another biconnected outerplane graph with two vertices of degree 2, namely t and one of u_1 and u_2). We obtain $\Gamma(G')$ from $\Gamma(G)$ by contracting u_2 onto u_1 . Let u_3 be the neighbor of u_1 and u_2 different from s . Since the edges incident to u_2 are contained into triangles Δ_{s,u_1,u_2} and Δ_{u_1,u_2,u_3} during the contraction, planarity is preserved (see Figs. 2(b) and 2(c)). Let p'_2 be the path composed of edge (u_1, u_3) and of the subpath of p_2 between u_3 and t , and let p'_1 be the subpath of p_1 between u_1 and t . Note that G' contains edge (u_1, u_3) and does not contain vertex u_2 . In both *Case 1* and *Case 2*, the decomposition tree $T_e(G')$ of G' differs from the decomposition tree $T_e(G)$ of G only “locally” to v . A precise description of the differences between $T_e(G)$ and $T_e(G')$ can be found in the extended version of the paper [4].

Let $\Gamma(G')$ be the drawing of the graph $G' = G \setminus \{v\}$ obtained after the contraction performed in *Case 1* or *Case 2*. Inductively construct a pseudo-morphing from $\Gamma(G')$ to the canonical drawing $\Gamma^*(G')$ of G' in $c \cdot (n - 1)$ steps, where c is a constant. Drawing $\Gamma^*(G)$ can be obtained from $\Gamma^*(G')$ by uncontracting v and by performing a constant number of morphing steps, as described in the following.

Here we only describe how to obtain $\Gamma^*(G)$ from $\Gamma^*(G')$ if *Case 1.1* was applied to contract v into one of its neighbors in p_2 . The other cases can be handled in a similar way (a full description can be found in the extended version of the paper [4]).

Drawings $\Gamma^*(G')$ and $\Gamma^*(G)$ coincide except for the fact that path p_2 in $\Gamma^*(G)$ contains v , while path p'_2 in $\Gamma^*(G')$ does not contain v . Paths p'_2 and p_2 are drawn inside two equal boomerangs in $\Gamma^*(G')$ and in $\Gamma^*(G)$, respectively, however v and some of the vertices of p'_2 need to be moved in order to obtain the drawing of p_2 as in $\Gamma^*(G)$. Namely, the drawing $\Gamma^*(p'_2)$ of p'_2 inside the boomerang N, E, S, W associated to τ_2 in $\Gamma^*(G')$ is composed of edges lying on two straight-line segments \overline{NC} and \overline{SC} , where C is the midpoint of segment \overline{EW} (see Fig. 3(a)). The drawing $\Gamma^*(p_2)$ of p_2 in $\Gamma^*(G)$ also lies inside N, E, S, W and is composed of edges lying on \overline{NC} and \overline{SC} , but vertices lie on different points (see Fig. 3(e)).

With one morphing step, uncontract v from the vertex it had been contracted onto and place it on any point of segment $\overline{v_1v_2}$ (note that edge (v_1, v_2) exists in G' and not in G ; see Fig. 3(b)). Then, in order to redistribute the vertices of p_2 on \overline{NC} and \overline{SC} , perform the following operation. Assume w.l.o.g. that s is on point N and t is on point S in $\Gamma^*(G')$ and in $\Gamma^*(G)$. Consider the vertices $w \in p_2$ and $w' \in p'_2$ that are placed on point C in $\Gamma^*(G)$ and $\Gamma^*(G')$, respectively. Note that either $w = w'$ or $(w, w') \in p_2$. If $w = w'$, either the subpath $p_2(s, w)$ of p_2 between s and w or the subpath $p_2(w, t)$ of p_2 between w and t has the same drawing in $\Gamma^*(G)$ and $\Gamma^*(G')$, say $p_2(w, t)$ has

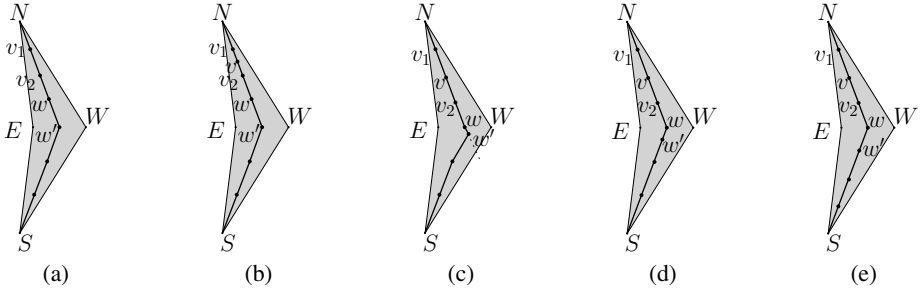


Fig. 3. Construction of $\Gamma^*(G)$ from $\Gamma^*(G')$ when *Case 1.1* applied. (a) $\Gamma^*(p'_2)$. The boomerang associated to τ_2 is light-grey. (b) Vertex v is uncontracted and placed on segment $\overline{v_1v_2}$. (c) Vertices on the path between s and w are placed in their final position, and vertex w' is placed arbitrarily close to C on the elongation of \overline{NC} . (d) Vertex w' is placed on \overline{SC} . (e) Vertices on the path between w and t are placed in their final position, hence obtaining $\Gamma^*(G)$.

such a property. With one morphing step move the vertices of $p_2(s, w)$ on segment \overline{NC} till reaching their positions in $\Gamma^*(G)$. If $w \neq w'$, assume without loss of generality that $w \in p_2(s, w')$. With one morphing step, move the vertices of $p_2(s, w)$ and vertex w' along the line through N and C , so that the vertices of $p_2(s, w)$ reach their positions in $\Gamma^*(G)$ and w' is placed arbitrarily close to C on the elongation of \overline{NC} (see Fig. 3(c)). With a second morphing step, move w' to any point of \overline{SC} between w and its other neighbor in p_2 (see Fig. 3(d)). Finally, with a third morphing step, move the vertices of $p_2(w, t)$ on segment \overline{SC} till reaching their positions in $\Gamma^*(G)$ (see Fig. 3(e)).

3.2 Simply-Connected Series-Parallel Graphs

In this section we show how, by preprocessing the input drawings Γ_a and Γ_b of any series-parallel graph G , the algorithm presented in Section 3.1 can be used to compute a pseudo-morph $M = \langle \Gamma_a, \dots, \Gamma_b \rangle$. The idea is to augment both Γ_a and Γ_b to two drawings Γ'_a and Γ'_b of a biconnected series-parallel graph G' , compute the morph $M' = \langle \Gamma'_a, \dots, \Gamma'_b \rangle$, and obtain M by restricting M' to the vertices and edges of G .

This augmentation is performed on G by repeatedly applying the following lemma.

Lemma 1. *Let v be a cut-vertex of a plane series-parallel graph G with n_b blocks. Let $e_1 = (u, v)$ and $e_2 = (w, v)$ be two consecutive edges in the circular order around v such that e_1 belongs to block b_1 of G and e_2 belongs to block $b_2 \neq b_1$ of G . The graph G^* obtained from G by adding a vertex z and edges (u, z) and (w, z) is a plane series-parallel graph with $n_b - 1$ blocks.*

Observe that, when augmenting G to G^* , both Γ_a and Γ_b can be augmented to two planar straight-line drawings Γ'_a and Γ'_b of G^* by placing vertex z suitably close to v and with direct visibility to vertices u and w , as in the proof of Fáry's Theorem [9]. By repeatedly applying such an augmentation we obtain a biconnected series-parallel graph G' and its drawings Γ'_a and Γ'_b , whose number of vertices and edges is linear in

the size of G . Hence, the algorithm described in Section 3.1 can be applied to obtain a pseudo-morph $\langle \Gamma_a, \dots, \Gamma_b \rangle$, thus proving Theorem 3. We will show in Section 5 how to obtain a morph starting from the pseudo-morph computed in this section.

4 Morphing Plane Graph Drawings in $O(n^2)$ Steps

In this section we prove the following theorem.

Theorem 4. *Let Γ_s and Γ_t be two drawings of the same plane graph G . There exists a pseudo-morph $\langle \Gamma_s, \dots, \Gamma_t \rangle$ with $O(n^2)$ steps transforming Γ_s into Γ_t .*

Preliminary Definitions. Let Γ be a planar straight-line drawing of a plane graph G . A face f of G is *empty* in Γ if it is delimited by a simple cycle. Consider a vertex v of G and let v_1 and v_2 be two of its neighbors. Vertices v_1 and v_2 are *consecutive* neighbors of v if no edge appears between edges (v, v_1) and (v, v_2) in the circular order of the edges around v in Γ . Let v be a vertex with $\deg(v) \leq 5$ such that each face containing v on its boundary is empty. We say that v is *contractible* [1] if, for each two neighbors u_1 and u_2 of v , edge (u_1, u_2) exists in G if and only if u_1 and u_2 are consecutive neighbors of v . We say that v is *quasi-contractible* if, for each two neighbors u_1 and u_2 of v , edge (u_1, u_2) exists in G only if u_1 and u_2 are consecutive neighbors of v . In other words, no edge exists between non-consecutive neighbors of a contractible or quasi-contractible vertex; also, each face incident to a contractible vertex v is delimited by a 3-cycle, while a face incident to a quasi-contractible vertex might have more than three incident vertices. We have the following.

Lemma 2. *Every planar graph contains a quasi-contractible vertex.*

Further, given a neighbor x of v , we say that v is *x -contractible* onto x in Γ if: (i) v is quasi-contractible, and (ii) the contraction of v onto x in Γ results in a straight-line planar drawing Γ' of $G' = G/(v, x)$.

The Algorithm. We describe the main steps of our algorithm to pseudo-morph a drawing Γ_s of a plane graph G into another drawing Γ_t of G .

First, we consider a quasi-contractible vertex v of G , that exists by Lemma 2. Second, we compute a pseudo-morph with $O(n)$ steps of Γ_s into a drawing Γ_s^x of G and a pseudo-morph with $O(n)$ steps of Γ_t into a drawing Γ_t^x of G , such that v is x -contractible onto the same neighbor x both in Γ_s^x and in Γ_t^x . We will describe later how to perform these pseudo-morphs. Third, we contract v onto x both in Γ_s^x and in Γ_t^x , hence obtaining two drawings Γ_s' and Γ_t' of a graph $G' = G/(v, x)$ with $n - 1$ vertices. Fourth, we recursively compute a pseudo-morph transforming Γ_s' into Γ_t' . This completes the description of the algorithm for constructing a pseudo-morphing transforming Γ_s into Γ_t . Observe that the algorithm has $p(n) \in O(n^2)$ steps, thus proving Theorem 4. Namely, as it will be described later, $O(n)$ steps suffice to construct pseudo-morphings of Γ_s and Γ_t into drawings Γ_s^x and Γ_t^x of G , respectively, such that v is x -contractible onto the same neighbor x both in Γ_s^x and in Γ_t^x . Further, two steps are sufficient to contract v onto x in both Γ_s^x and Γ_t^x , obtaining drawings Γ_s'

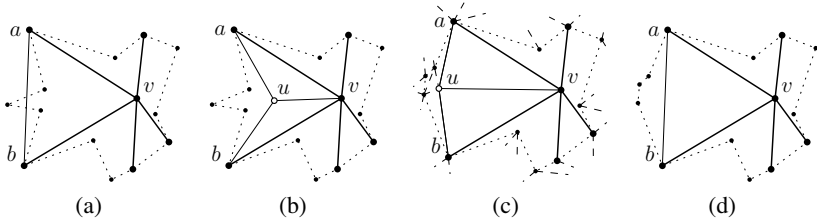


Fig. 4. Vertex v and its neighbors. (a) Vertices a and b do not have direct visibility and the triangle $\langle a, b, v \rangle$ is not empty. (b) A vertex u is added suitably close to v and connected to v , a , and b . (c) The output of CONVEXIFIER on the quadrilateral $\langle a, b, v, u \rangle$. (d) Vertex u and its incident edges can be removed in order to insert edge (a, b) .

and Γ'_t , respectively. Finally, the recursion on Γ'_s and Γ'_t takes $p(n - 1)$ steps. Thus, $p(n) = p(n - 1) + O(n) \in O(n^2)$. We will show in Section 5 how to obtain a morph starting from the pseudo-morph computed in this section.

We remark that our approach is similar to the one proposed by Alamdari *et al.* [1]. In [1] Γ_s and Γ_t are augmented to drawings of the same maximal planar graph with $m \in O(n^2)$ vertices, and a morph with $O(m^2)$ steps is constructed between two drawings of the same m -vertex maximal planar graph. This results in a morphing between Γ_s and Γ_t with $O(n^4)$ steps. Here, we also augment Γ_s and Γ_t to drawings of maximal planar graphs. However, we only require that the two maximal planar graphs coincide in the subgraph induced by the neighbors of v . Since this can be achieved by adding a constant number of vertices to Γ_s and Γ_t , namely one for each of the at most five faces v is incident to, our morphing algorithm has $O(n^2)$ steps.

Making v x -contractible. Let v be a quasi-contractible vertex of G . We show an algorithm to construct a pseudo-morph with $O(n)$ steps transforming any straight-line planar drawing Γ of G into a straight-line planar drawing Γ' of G such that v is x -contractible onto any neighbor x . If v has degree 1, then it is contractible into its unique neighbor in Γ , and there is nothing to prove.

In order to transform Γ into Γ' , we use a support graph S and its drawing Σ , initially set equal to G and Γ , respectively. The goal is to augment S and Σ so that v becomes a contractible vertex of S . In order to do this, we have to add to S an edge between every two consecutive neighbors of v . However, the insertion of these edges might not be possible in Σ , as it might lead to a crossing or to enclose some vertex inside a cycle delimited by v and by two consecutive neighbors of v (see Fig. 4(a)).

Let a and b be two consecutive neighbors of v . If the closed triangle $\langle a, b, v \rangle$ does not contain any vertex other than a , b , and v , then add edge (a, b) to S and to Σ as a straight-line segment. Otherwise, proceed as follows.

Let Σ_u be the drawing of a plane graph S_u obtained by adding a vertex u and the edges (u, v) , (u, a) , and (u, b) to Σ and to S , in such a way that the resulting drawing is straight-line planar and each face containing u on its boundary is empty. As in the proof of Fáry's Theorem [9], a position for u with such properties can be found in Σ , suitably close to v . See Fig. 4(b). Augment Σ_u to the drawing Θ of a maximal plane graph T

by first adding three vertices p , q , and r to Σ_u , so that triangle $\langle p, q, r \rangle$ encloses the rest of the drawing, and then adding dummy edges [8]. If edge (a, b) has been added in this augmentation (this can happen if a and b share a face not having v on its boundary), subdivide (a, b) in Θ (namely, replace (a, b) with edges (a, w) and (w, b) , placing w along the straight-line segment connecting a and b) and triangulate the two faces vertex w is incident to. Next, apply the algorithm described in [1], that we call CONVEXIFIER, to construct a morph of Θ into a drawing Θ' of T in which polygon $\langle a, v, b, u \rangle$ is convex. The input of algorithm CONVEXIFIER consists of a planar straight-line drawing Γ^* of a plane graph G^* and of a set of at most five vertices of G^* inducing a biconnected outerplane graph not containing any other vertex in its interior in Γ^* . The output of algorithm CONVEXIFIER is a sequence of $O(n)$ linear morphing steps transforming Γ^* into a drawing of G^* in which the at most five input vertices bound a convex polygon. Since, by construction, vertices a, v, b, u satisfy all such requirements, we can apply algorithm CONVEXIFIER to Θ and to a, v, b, u , hence obtaining a morph with $O(n)$ steps transforming Θ into the desired drawing Θ' (see Fig. 4(c)). Let Σ'_u be the drawing of S_u obtained by restricting Θ' to vertices and edges of S_u . Since $\langle a, v, b, u \rangle$ is a convex polygon containing no vertex of S_u in its interior, edge (u, v) can be removed from Σ'_u and an edge (a, b) can be introduced in Σ'_u , so that the resulting drawing Σ' is planar and cycle (a, b, v) does not contain any vertex in its interior (see Fig. 4(d)).

Once edge (a, b) has been added to S (either in Σ or after the described procedure transforming Σ into Σ'), if $\deg(v) = 2$ then v is both a -contractible and b -contractible. Otherwise, consider a new pair of consecutive vertices of v not creating an empty triangular face with v , if any, and apply the same operations described before.

Once every pair of consecutive vertices has been handled, vertex v is contractible in S . Let Σ_v be the current drawing of S . Augment Σ_v to the drawing Θ_v of a triangulation T_v (by adding three vertices and a set of edges), contract v onto a neighbor w such that v is w -contractible (one of such neighbors always exists, given that v is contractible), and apply CONVEXIFIER to the resulting drawing Θ'_v and to the neighbors of v to construct a morphing Θ'_v to a drawing Σ'_v in which the polygon defined by such vertices is convex. Drawing Γ' of G in which v is x -contractible for any neighbor x of v is obtained by restricting Σ'_v to the vertices and the edges of G . We can now contract v onto x in Γ' and recur on the obtained graph (with $n - 1$ vertices) and drawing.

It remains to observe that, given a quasi-contractible vertex v , the procedure to construct a pseudo-morph of Γ into Γ' consists of at most $\deg(v) + 1$ executions of CONVEXIFIER, each requiring a linear number of steps [1]. As $\deg(v) \leq 5$, the procedure to pseudo-morph Γ into Γ' has $O(n)$ steps. This concludes the proof of Theorem 4.

5 Transforming a Pseudo-Morph into a Morph

In this section we show how to obtain an actual morph M from a given pseudo-morph \mathcal{M} , by describing how to compute the placement and the motion of any vertex v that has been contracted during \mathcal{M} . By applying this procedure to Theorems 3 and 4, we obtain a proof of Theorems 1 and 2.

Let Γ be a drawing of a graph G and let $\mathcal{M} = \langle \Gamma, \dots, \Gamma^* \rangle$ be a pseudo-morph that consists of the contraction of a vertex v of G onto one of its neighbors x , followed by a pseudo-morph \mathcal{M}' of the graph $G' = G/(v, x)$, and then of the uncontraction of v .

The idea of how to compute M from \mathcal{M} is the same as in [1]: Namely, morph M is obtained by (i) recursively converting \mathcal{M}' into a morph M' ; (ii) modifying M' to a morph M'_v obtained by adding vertex v (and its incident edges) to each drawing of M' , in a suitable position; (iii) replacing the contraction of v onto x , performed in \mathcal{M} , with a linear morph that moves v from its initial position in Γ to its position in the first drawing of M'_v ; and (iv) replacing the uncontraction of v , performed in \mathcal{M} , with a linear morph that moves v from its position in the last drawing of M'_v to its final position in Γ^* . Note that, in order to guarantee the planarity of M when adding v to any drawing of M' in order to obtain M'_v , vertex v must lie inside its kernel. Since vertex x lies in the kernel of v (as x is adjacent to all the neighbors of v in G'), we achieve this property by placing v suitably close to x , as follows.

At any time instant t during M' , there exists an $\epsilon_t > 0$ such that the disk D centered at x with radius ϵ_t does not contain any vertex other than x . Let ϵ be the minimum among the ϵ_t during M' . We place vertex v at a suitable point of a sector S of D according to the following cases. **Case (a): v has degree 1 in G .** Sector S is defined as the intersection of D with the face containing v in G . See Fig. 5(a). **Case (b): v has degree 2 in G .** Sector S is defined as the intersection of D with the face containing v in G and with the halfplane defined by the straight-line passing through x and r , and containing v in Γ . See Fig. 5(b). Otherwise, $\deg(v) \geq 3$ in G' . Let (r, v) and (l, v) be the two edges such that (r, v) , (x, v) , and (l, v) are clockwise consecutive around v in G . Observe that edges (r, x) and (l, x) exist in G' . Assume that x , r , and l are not collinear in any drawing of M' , as otherwise we can slightly perturb such a drawing without compromising the planarity of M' . Let α_i be the angle \widehat{lxr} in any intermediate drawing of M' . **Case (c): $\alpha_i < \pi$.** Sector S is defined as the intersection of D with the wedge delimited by edges (x, r) and (x, l) . See Fig. 5(c). **Case (d): $\alpha_i > \pi$.** Sector S is defined as the intersection of D with the wedge delimited by the elongations of (x, r) and (x, l) emanating from x . See Fig. 5(d). By exploiting the techniques shown in [1], the motion of v can be computed according to the evolution of S over M' , thus obtaining a planar morph M'_v .

Observe that, in the algorithm described in Section 4, the vertex x onto which v has been contracted might be not adjacent to v in G . However, since a contraction has been performed, x is adjacent to v in one of the graphs obtained when augmenting G during the algorithm. Hence, a morph of G can be obtained by applying the above procedure to the pseudo-morph computed on this augmented graph and by restricting it to the vertices and edges of G .

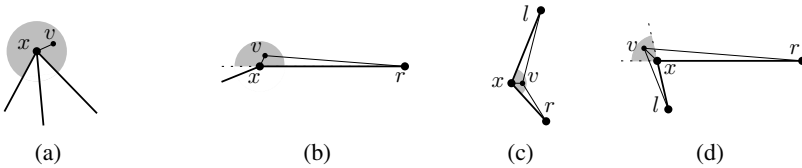


Fig. 5. Sector S (in grey) when: (a) $\deg(v) = 1$, (b) $\deg(v) = 2$, and (c)-(d) $\deg(v) \geq 3$.

6 Conclusions and Open Problems

In this paper we studied the problem of designing efficient algorithms for morphing two planar straight-line drawings of the same graph. We proved that any two planar straight-line drawings of a series-parallel graph can be morphed with $O(n)$ linear morphing steps, and that a planar morph with $O(n^2)$ linear morphing steps exists between any two planar straight-line drawings of any planar graph.

It is a natural open question whether the bounds we presented are optimal or not. We suspect that planar straight-line drawings exist requiring a linear number of steps to be morphed one into the other. However, no super-constant lower bound for the number of morphing steps required to morph planar straight-line drawings is known. It would be interesting to understand whether our techniques can be extended to compute morphs between any two drawings of a *partial planar 3-tree* with a linear number of steps. We recall that, as observed in [1], a linear number of morphing steps suffices to morph any two drawings of a *maximal planar 3-tree*.

References

1. Alamdari, S., Angelini, P., Chan, T.M., Di Battista, G., Frati, F., Lubiw, A., Patrignani, M., Roselli, V., Singla, S., Wilkinson, B.T.: Morphing planar graph drawings with a polynomial number of steps. In: SODA 2013, pp. 1656–1667 (2013)
2. Angelini, P., Cortese, P.F., Di Battista, G., Patrignani, M.: Topological morphing of planar graphs. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 145–156. Springer, Heidelberg (2009)
3. Angelini, P., Didimo, W., Kobourov, S., Mchedlidze, T., Roselli, V., Symvonis, A., Wismath, S.: Monotone drawings of graphs with fixed embedding. *Algorithmica*, 1–25 (2013)
4. Angelini, P., Frati, F., Patrignani, M., Roselli, V.: Morphing planar graph drawings efficiently. CoRR cs.CG (2013), <http://arxiv.org/abs/1308.4291>
5. Angelini, P., Colasante, E., Di Battista, G., Frati, F., Patrignani, M.: Monotone drawings of graphs. *J. of Graph Algorithms and Appl.* 16(1), 5–35 (2012); In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 13–24. Springer, Heidelberg (2011)
6. Biedl, T.C., Lubiw, A., Spriggs, M.J.: Morphing planar graphs while preserving edge directions. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 13–24. Springer, Heidelberg (2006)
7. Cairns, S.S.: Deformations of plane rectilinear complexes. *American Math. Monthly* 51, 247–252 (1944)
8. Chazelle, B.: Triangulating a simple polygon in linear time. *Discrete & Computational Geometry* 6(5), 485–524 (1991)
9. Fáry, I.: On straight line representation of planar graphs. *Acta Univ. Szeged. Sect. Sci. Math.* 11, 229–233 (1948)
10. Grunbaum, B., Shephard, G.: The geometry of planar graphs. Cambridge University Press (1981), <http://dx.doi.org/10.1017/CBO9780511662157.008>
11. Lubiw, A., Petrick, M.: Morphing planar graph drawings with bent edges. *Electronic Notes in Discrete Mathematics* 31, 45–48 (2008)
12. Lubiw, A., Petrick, M., Spriggs, M.: Morphing orthogonal planar graph drawings. In: SODA 2006, pp. 222–230. ACM (2006)
13. Thomassen, C.: Deformations of plane graphs. *J. Comb. Th., Series B* 34, 244–257 (1983)

Graph Drawing through the Lens of a Framework for Analyzing Visualization Methods (Invited Talk, Extended Abstract)

Tamara Munzner

University of British Columbia, Department of Computer Science
Vancouver BC, Canada

tmm@cs.ubc.ca

<http://www.cs.ubc.ca/~tmm>

Abstract. The visualization community has drawn heavily on the algorithmic and systems-building work that has appeared with the graph drawing literature, and in turn has been a fertile source of applications. In the spirit of further promoting the effective transfer of ideas between our two communities, I will discuss a framework for analyzing the design of visualization systems. I will then analyze a range of graph drawing techniques through this lens. In the early stages of a project, this sort of analysis may benefit algorithm developers who seek to identify open problems to attack. In later project stages, it could guide algorithm developers in characterizing how newly developed layout methods connect with the tasks and goals of target users in different application domains.

1 Introduction

Visualization researchers and practitioners have long drawn on the algorithmic work conducted by the graph drawing community, and in turn have helped establish connections between that community and end users for specific application areas. Moreover, the network data that is the focus of the graph drawing community's efforts can be considered as a special case of the broader spectrum of data that is of interest in visualization, and thus its general principles are relevant.

I propose analysis of visualization techniques through *methods*; that is, an enumeration of the design space of techniques in terms of specific sets of choices. This kind of analysis supports thinking systematically about the space of possibilities. It may help a designer in the early stages of developing a new technique to identify gaps in the previous work to address. It can also be used to characterize existing work, in service of matching up which algorithms and techniques are suitable for which real-world problems. Further reading about this analysis framework can be found in an existing book chapter [13] and a forthcoming book [14]. These sources include many more references to the extensive related work that underlies this framework, which I do not directly include here.

In this talk, I begin with a distinction between four levels of visualization design, and continue with a brief discussion of abstraction for data. I introduce the

principles of marks and channels, and discuss the use of space in a visualization context. I continue with further examples of analysis drawn from graph drawing, and then conclude.

2 Levels of Visualization Design

In recent work, I proposed separating the design concerns of visualization into four levels: domain problem, data and task abstraction, visual encoding and interaction technique, and algorithm, as shown in Figure 1 [10]. In that paper, I also discuss the problem of how to validate designs at each of these levels. In this talk I will emphasize techniques, which are at a level just above the algorithm level that is the focus of much of the research from the graph drawing community. I also discuss the abstraction level briefly, to provide background context.

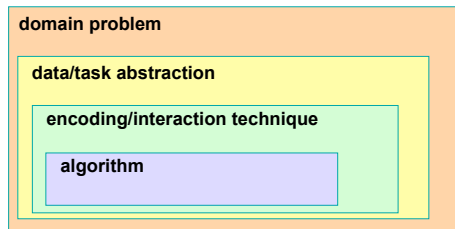


Fig. 1. Four levels of visualization design concerns [10]

My characterization here focuses on one major issue: how is space used? This question is an explicit consideration in visualization, but the motivation is not quite so obvious when considered purely from the perspective of problems that arise in graph drawing. I conjecture that the reason for this difference is that the very common cases in graph drawing, such as force-directed placement with node-link representations, or compound graphs that combine an underlying network with a hierarchy on top of it, are not trivial to analyze. My goal is to encourage more upwards characterization to map from algorithms up to techniques; that is, where algorithms are characterized in terms of the visual encoding and interaction techniques that they support.

When considering the four levels of design, another obvious route of attack is downwards from the top level of a domain problem; that is, to design a visualization system intended to solve some specific problem for a set of target users who have real data and real tasks. This sort of problem-driven work, often called *design studies* in the visualization literature has rich and interesting challenges, many of which are quite different than those that arise from technique-driven work. A detailed discussion of these issues appears elsewhere, in a recent paper on the methodology of design studies [17], and is beyond the scope of this talk.

3 Abstraction for Data

For the purposes of this framework, I will define only two basic types of data abstractions. At the dataset level, there are two major dataset types: *tables* and *networks*. In a simple table, I will call the rows *items* and the columns *attributes*. In a network, I distinguish between two kinds of items: *nodes*, and the *links* between them; either can have attributes. Obviously, the network dataset type is the focus of interest in graph drawing, but I will also present some analyses of table data as part of building up the framework. Attributes also have types. *Categorical* attributes have no implicit ordering, in contrast to *ordered* attributes; these are split into *quantitative* attributes that support full arithmetic operations such as addition or subtraction, in contrast to *ordinal*. For example, type of fruit is a categorical attribute; weight is quantitative; T-shirt size is ordinal.

The common case in visualization of complex, real-world data is that the designer will need to derive additional data beyond the original dataset. This derived data might be new attributes, or even a transformation from one dataset type to another, as with transforming a network into a table or vice versa. One example of a derived quantitative attribute computed from an original network is the Strahler number, a node-based centrality metric. Auber proposed exploiting it for fast interactive rendering of large graphs: by drawing nodes in priority order according to this attribute, a comprehensible skeleton of the network results from drawing only a small fraction of the nodes, in contrast to the poor results from drawing a random sampling [2].

4 Principles of Marks and Channels

I will introduce the idea of breaking down a visual encoding in terms of marks and channels by first considering some easy cases from statistical graphics that show tabular data: bar charts and scatterplots. These plots are straightforward to break down into *marks*, namely geometric primitives that represent items, and visual *channels* that control the appearance of marks. Marks are classified by their dimensionality: points, lines, areas, or volumes. Visual channels include spatial position, color, shape, size, orientation/tilt, and many others. A simple bar chart uses line marks, and encodes one attribute according to vertical spatial position channel. A scatterplot uses point marks, and encodes two attributes: one with the vertical spatial position channel, and one with horizontal position. A third attribute can be added to a scatterplot by encoding with the color channel, and a fourth by encoding with the size channel. The principles of marks and channels can be used to analyze more complex visual encoding techniques beyond these simple statistical graphics.

In addition to marks that represent items or nodes, marks may represent links. Link marks should implicitly convey the idea of relationships between items at a perceptual level. There are two particularly perceptually appropriate ways to do so: containment and connection. Containment uses an area mark to enclose a set of other marks within it; connection uses a line mark to directly connect

two other marks together. I use the terms *connection* and *containment* for link marks, in contrast to *line* and *area* for item marks, to underscore that they communicate relationships between multiple items. A third perceptual way to indicate relationship is *proximity*, where items that are close to each other are implicitly perceived as being more related than those that are far apart. It is not possible to directly use proximity as a mark type, but in the next section I will discuss where it fits within the analysis of space use in visualization.

A crucial aspect of visual channels is that they also have implicit perceptual types, and these can and should be matched with attribute types. Some channels intrinsically convey *how much* in a way that maps well to ordered attributes, such as the spatial position along a common scale, or the length of a line mark, or the size of a point mark. Other channels convey *what* in a way that maps well to categorical attributes, such as what spatial region a mark is within, or what color a mark is, or what shape a mark is. The channels associated with the use of space have the strongest perceptual impact, leading to my choice to emphasize the spatial channels in this talk. The other channels can also be roughly ranked in terms of perceptual impact. In another talk, I discuss the underlying reasons for these rankings and a number of visualization principles that arise from them [11].

5 Using Space

I now discuss in more detail the ways to use space in the design of visual encoding and interaction techniques, emphasizing the different possible uses of spatial channels to control the appearance of marks.

I distinguish between five ways to use space: *use* given data; *express* values; and *separate*, *order*, and *align* regions. In the first case, the spatial layout is given, whereas in the other four the use of space is chosen. Using the data as given is the common case with geographic data. Although there are still many nuances in design considerations, as discussed in the cartographic literature, the fundamental use of space is constrained by this choice. This approach is also the common case when dealing with scalar, vector, or tensor spatial fields, where data is sampled at many points in the field, as in volume graphics and flow visualization. Of course, the existence of spatial data does not dictate its use as the fundamental use of space in a visual encoding; a designer may still choose to derive additional data and use space differently, as discussed below. I will not discuss this case further in this talk, where I focus on choosing the use of space. Although sometimes networks are drawn using given geographic data for node positions, the common case in graph drawing is on making choices about the use of space.

The case of expressing values spatially closely follows the discussion of marks and channels in the previous section: a quantitative attribute is encoded using the spatial position of a mark. Scatterplots are the quintessential example of expressing values in this way. In contrast, the other three uses of space pertain to establishing *regions*. Separating space into distinct regions, where each region shows something different, has major implications for how we perceive the structure of the dataset. Spatial proximity strongly implies grouping at a perceptual

level, and so items in different regions are perceived as being in different categories. The regions themselves can be ordered with respect to each other, for example in a data-driven way according to an ordered attribute. Finally, regions can also be aligned to a shared baseline. Lengths and positions can be compared with higher precision between aligned regions than with unaligned regions, again for fundamental perceptual reasons. A 1D alignment is a list, while aligning in 2D yields a matrix, and in 3D a volumetric grid. In any of these cases, recursive subdivision is possible to accommodate hierarchical attribute structure.

The most extreme form of separation between regions is to divide the display into multiple separate views. There are three major approaches of combining views: showing multiple views side by side, superimposing multiple views on top of each other, and having a single view that changes over time. When superimposing multiple views as layers, they must all have a shared spatial layout. A single changing view is the common case for interactive navigation. Using multiple views side by side is a particularly powerful method because of the principle that “eyes beat memory” [11]. It is easy to compare by moving one’s eyes between side by side views, where the views act as external cognitive supports. It is harder to compare a visible item to the memory of what one saw before, because of the limits of internal working memory.

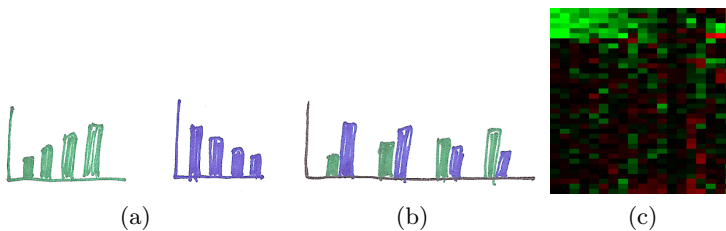


Fig. 2. Three ways to show a multidimensional table. a) Separate bar charts. b) Single interleaved bar chart. c) Heatmap. Figure credit: <http://commons.wikimedia.org/wiki/File:Heatmap.png>

Another seemingly simple example from statistical graphics is nevertheless a good example of the more complex uses of space: a multidimensional table of data with three attributes, one quantitative and two categorical. One categorical attribute is the type of export and there are two possible values: wine or cheese. The other categorical attribute is the name of the city, and there are four possible values. The quantitative attribute is the value in euro of the city’s exports for a type over a year. We might encode this table as two separate bar charts, one for wine and one for cheese, with simple line marks in each, as in Figure 2a. An alternative is one interleaved bar chart where each item of data is depicted with two marks side by side, as in Figure 2b. We now have the vocabulary to analyze this choice in more detail in terms of channels and attributes: with separate charts, we have first separated into two large regions based on one categorical attribute, export type, then separated each of those into four smaller

regions based on the other categorical attribute, city name. These subregions are aligned, and within each a single line mark expresses a value. The subregions are also ordered, for example either alphabetically by the name of the city, or in a data-driven way by the value of quantitative attribute, the exports. With interleaved charts, there is only one level of separation into regions: there are four regions, one for each item. Each region shows information about two attributes, as two side-by-side marks. The value of description at this level of detail is that we can reason about what kinds of information can be easily perceived by the viewer based on the partition into regions. In the first case, with two separate simple charts, the partition into one region for each type of export allows the easy perception of trends for that type. In the second case, having the two marks side by side allows the easy comparison of the export mix for a particular city.

We can now consider a quite different visual encoding of a heatmap display, which shows exactly the same data abstraction: a table with one quantitative attribute indexed by two categorical attributes. In a heatmap, regions are separated and aligned into a 2D matrix, and within each cell of the matrix an area mark is used in conjunction with the color channel to encode a quantitative attribute. The scale of the data is different: there are dozens or hundreds of values for each categorical attributes. In Figure 2c, these are genes and experimental conditions, and the quantitative attribute is the expression level of a gene in a specific condition.

I now turn from table to network datasets. A matrix view of a network is essentially the same as a heatmap, in terms of both data abstraction choice and use of space. The transformation at the data abstraction level is to transform the original network into a table, where a list of the nodes in the graph is used as both of the categorical attributes, and the weighted edge between a pair of nodes is the quantitative attribute. Thus, a cell in the matrix shows the presence or absence of an edge.

In contrast, the most common case in graph drawing is to use link marks to explicitly show links. As the name suggests, all node-link diagrams are an instance of using link connection marks. These diagrams best support tasks that pertain to the topological structure of networks [7], for example path tracing. In tree drawing, containment is also used; for example, all treemap variants are an instance of using link containment marks. These diagrams typically use the size channel to encode attribute values, either just for leaf nodes or recursively for interior nodes as well, and thus support tasks that pertain to understanding those attribute values.

In addition to the five choices for the use of spatial channels, it is useful to consider the orientation of the spatial axes within the layout. The two most common cases in graph drawing are *rectilinear* and *radial*. In 2D, the rectilinear choice yields Cartesian coordinates and the radial choice is polar coordinates. A third choice is to orient all of the axes *parallel* to each other. The limitations of these choices have been investigated in the visualization literature. Rectilinear layouts have the obvious scalability issue that the number of axes is highly constrained. A 2D rectilinear layout allows very high-precision perception of information encoded with

spatial position. While 3D rectilinear layouts are possible, there are many perceptual problems in using three dimensions of space to encode nonspatial data [11]. Four or more rectilinear axes cannot be directly encoded. There has also been empirical work to investigate the strengths and weaknesses of radial layouts, in light of the known limitation that we perceive angles with less accuracy than lengths [5].

The work of McGuffin and Robert is a good example of analyzing many different tree drawing methods according to the efficiency of their use of space [8]. The *information density* of a diagram is as a measure of the amount of information encoded vs. the amount of unused space; there is a tradeoff between encoding as much information as possible, and the potential for visual clutter or other legibility problems. The examples that they analyze can be considered in terms of the methods that I have covered: whether connection or containment link marks are used, whether the layout is rectilinear or radial, how the spatial position channels are used to encode information. The information encoded in these diagrams is the link relationships, the depth in the tree of a node, and the order of siblings. Other analysis considerations are whether some information is encoded redundantly, for example through both spatial position and explicit link marks, and whether any arbitrary information is expressed through the use of spatial ordering. For example, in some trees, sibling order is not specifically defined, yet there is a visible spatial order.

Force-directed placement is a widely adopted approach for visually encoding network datasets. The visual encoding is in some sense straightforward because it is a node-link diagram: point marks represent nodes, and connection marks represent links as lines. However, considering the meaning of spatial position is somewhat tricky, because no meaning is directly encoded; instead it is left free to minimize crossings. Thus, the semantics of proximity are mixed: sometimes it is meaningful, for example when a cluster of nodes placed near to each other truly reflects strong interconnections of links between them. Sometimes it is an arbitrary artifact of the layout algorithm, and two nodes that happen to be nearby in one layout may be quite far in another. There is also an interesting tension between proximity cues and edge length: long edges are more visually salient than short ones that connect nodes close to each other.

A great deal of work has been devoted to developing better algorithms for force-directed placement through multilevel methods; the *sfdp* algorithm is one example [6]. The data abstraction is more complex, namely a compound graph: in addition to the original network dataset, the additional derived data of a cluster hierarchy atop the original network is computed. Although this hierarchy is used within the algorithm, it is not shown explicitly within the drawing. Thus, the fundamental use of space is the same as with simpler versions of force-directed placement; this multilevel approaches is an example of a better algorithm for the same visual encoding technique.

The GrouseFlocks system for the interactive analysis of compound graphs [1] has some instructive similarities and differences from *sfdp*. The data abstraction is the same, a compound graph. However, the visual encoding is different, as shown in Figure 3c. In addition to connection marks for the network links

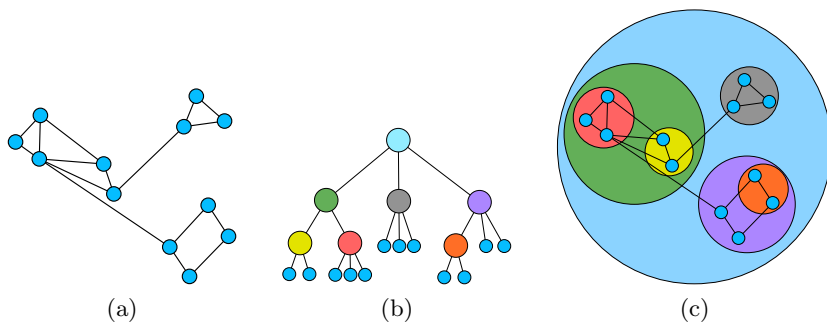


Fig. 3. Compound graph representation in GrouseFlocks [1]. a) Original network. b) Cluster hierarchy. c) Visual encoding in tool, fully expanded.

and point marks for the nodes, the hierarchy links are shown with containment marks. The system features dynamic interaction in order to support large datasets, where typically only an interactively-selected subset of the full compound graph is shown. The user can expand and contract individual metanodes in the hierarchy, which also shows the associated nodes from the base network.

6 Further Analysis Examples

In the talk, I will analyze three more systems within this framework: Cerebral [3, 4], Constellation [12, 15], and Noack’s LinLog energy model [16]. All three are examples of design motivated by explicit prior analysis of the use of space.

The Cerebral system [3, 4] features both multiple side by side views, and superimposed layers within each view. The network layout is designed to mimic the semantics of hand-drawn diagrams of biological networks. I will analyze its visual encoding and design choices pertaining to the use of space in detail.

The Constellation system [12, 15] features a complex multi-level linguistic network that is laid out with spatial position reflecting specific attributes, where edge crossings are resolved using perceptual layers rather than through algorithmically reducing the number of instances.

Noack’s LinLog energy model is designed to reveal clusters in data, by requiring that edges between clusters are longer than those within [16]. Noack specifically indicates that his approach uses the same minimization algorithms as previous work, and frames the energy model in terms of its visual results. I thus consider it a contribution at the visual encoding technique level, even though that exact vocabulary out of the visualization community does not appear in the paper. I note that it was published at a previous Graph Drawing conference, in contrast to the many other examples in this talk that come out of the visualization literature. I encourage more papers like this that can act as bridges between the communities!

7 Conclusions

I have discussed a general framework for systematically analyzing visualization techniques in terms of the methods of using space, and applied it to graph drawing examples in particular. It relies on breaking down visual encodings into marks that are geometric primitives representing either nodes or links, and channels that control their appearance to encode attributes. In this talk I focus on the channels related to the use of space. The simple case is using spatial position to express a quantitative attribute value, but space can also be separated into regions that partition according to a categorical attribute to indicate groups via proximity. These regions can also be ordered and aligned. This framework is a mix of ideas of that are widespread in the visualization literature and those that are new; for a more detailed discussion of the previous work, see the existing chapter [13] and the forthcoming full book [14]. These sources also describe principles in detail, and also the more complete analysis framework from which the subset discussed here was drawn. The full framework includes the nonspatial channels in addition to the spatial ones and a much more detailed discussion of methods for combining multiple views. It covers interactive techniques in addition to visual encoding techniques, particularly in terms of methods for the reduction of the amount of data shown.

This kind of analysis can guide the development of new techniques, or be used to characterize existing ones. While sometimes it is easy to map from a specific algorithm to the visual encoding technique that it supports, for example when the mapping is explicitly discussed in a paper or in work that it directly cites, sometimes this mapping is difficult to reverse-engineer. Algorithm descriptions may not facilitate analysis of the resulting visual encoding, either for the use of space or for other channels. In these cases, the line between technique and algorithm can be blurry: does a new algorithm support an existing technique, or does it result in a new one? Carrying out more such characterization may facilitate the transfer of algorithms from the graph drawing community to the visualization community. It is also important to characterize mappings between the other levels of visualization design [9], but this important question is beyond the scope of this talk.

Of course, characterization according to this sort of framework is only one of many possible ways to analyze graph drawing and visualization approaches. Benchmarks and complexity analysis are a different way to compare approaches, as are user studies in the form of controlled experiments or more qualitative investigation of how people use visualization systems [10].

References

- [1] Archambault, D., Munzner, T., Auber, D.: GrouseFlocks: Steerable exploration of graph hierarchy space. *IEEE Trans. on Visualization and Computer Graphics* 14(4), 900–913 (2008)
- [2] Auber, D.: Using Strahler numbers for real time visual exploration of huge graphs. In: *Intl. Conf. Computer Vision and Graphics*, pp. 56–69 (2002)

- [3] Barsky, A., Gardy, J.L., Hancock, R.E., Munzner, T.: Cerebral: A Cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics* 23(8), 1040–1042 (2007)
- [4] Barsky, A., Munzner, T., Gardy, J., Kincaid, R.: Cerebral: Visualizing multiple experimental conditions on a graph with biological context. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis 2008)* 14(6), 1253–1260 (2008)
- [5] Diehl, S., Beck, F., Burch, M.: Uncovering strengths and weaknesses of radial visualizations - an empirical approach. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis)* 16(6), 935–942 (2010)
- [6] Hu, Y.: Efficient and high quality force-directed graph drawing. *The Mathematica Journal* 10, 37–71 (2005)
- [7] Lee, B., Plaisant, C., Parr, C.S., Fekete, J.D., Henry, N.: Task taxonomy for graph visualization. In: *Proc. AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV)*. ACM Press (2006)
- [8] McGuffin, M.J., Robert, J.M.: Quantifying the space-efficiency of 2D graphical representations of trees. *Information Visualization* 9(2), 115–140 (2010)
- [9] Meyer, M., Sedlmair, M., Munzner, T.: The four-level nested model revisited: Blocks and guidelines. In: *Proc. Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization (BELIV)* (2012)
- [10] Munzner, T.: A nested model for visualization design and validation. *IEEE Trans. Visualization and Computer Graphics (TVCG)* 15(6), 921–928 (2009)
- [11] Munzner, T.: Visualization Principles. *VizBi 2011 Keynote* (2011), <http://www.cs.ubc.ca/~tmm/talks.html#vizbi11>
- [12] Munzner, T.: Constellation: Linguistic semantic networks. In: *Interactive Visualization of Large Graphs and Networks (PhD thesis)*, ch. 5, pp. 105–122. Stanford University Department of Computer Science (2000)
- [13] Munzner, T.: Visualization. In: Shirley, P., Marschner, S. (eds.) *Fundamentals of Graphics*, ch. 27, 3rd edn., vol. ch. 27, pp. 675–707. AK Peters (2009b)
- [14] Munzner, T.: *Visualization Analysis and Design: Principles, Abstractions, and Methods*. AK Peters (to appear, 2014)
- [15] Munzner, T., Guimbretière, F., Robertson, G.: Constellation: A visualization tool for linguistic queries from MindNet. In: *Proc. IEEE Symposium on Information Visualization (InfoVis)*, pp. 132–135 (1999)
- [16] Noack, A.: An energy model for visual graph clustering. In: Liotta, G. (ed.) *GD 2003*. LNCS, vol. 2912, pp. 425–436. Springer, Heidelberg (2004)
- [17] Sedlmair, M., Meyer, M., Munzner, T.: Design study methodology: Reflections from the trenches and the stacks. *IEEE Trans. Visualization and Computer Graphics (TVCG)* 18(12), 2431–2440 (2012)

A Linear-Time Algorithm for Testing Outer-1-Planarity^{*,**}

Seok-Hee Hong¹, Peter Eades¹, Naoki Katoh², Giuseppe Liotta³,
Pascal Schweitzer⁴, and Yusuke Suzuki⁵

¹ University of Sydney, Australia

{seokhee.hong, peter.eades}@sydney.edu.au

² Kyoto University, Japan

naoki@archi.kyoto-u.ac.jp

³ University of Perugia, Italy

liotta@diei.unipg.it

⁴ ETH, Switzerland

pascal@mpi-inf.mpg.de

⁵ Niigata University, Japan

y-suzuki@math.sc.niigata-u.ac.jp

Abstract. A graph is *1-planar* if it can be embedded in the plane with at most one crossing per edge. A graph is *outer-1-planar* if it has an embedding in which every vertex is on the outer face and each edge has at most one crossing. We present a linear time algorithm to test whether a graph is outer-1-planar. The algorithm can be used to produce an outer-1-planar embedding in linear time if it exists.

1 Introduction

A recent research topic in topological graph theory is the study of graphs that are *almost planar* in some sense. Examples of such almost planar graphs are *1-planar* graphs, which can be embedded in a plane with at most one crossing per edge.

Ringel [3] introduced 1-planar graphs in the context of simultaneously coloring vertices and faces of planar graphs. Subsequently, various aspects of 1-planar graphs have been investigated. Borodin [4] gives colouring methods for 1-planar graphs. Pach and Toth [5] prove that a 1-planar graph with n vertices has at most $4n - 8$ edges, which is a tight upper bound. There are a number of structural results on 1-planar graphs [6], and *maximal* 1-planar embeddings [7] (a 1-planar embedding of a graph G is *maximal*, if no edge can be added without violating the 1-planarity of G).

The class of 1-planar graphs is not closed under edge contraction; accordingly, computational problems seem difficult. Korzhik and Mohar proved that testing 1-planarity

* This paper is an extended abstract. For omitted proofs, see the full version of this paper [1].
The problem studied in this paper was initiated at the Port Douglas Workshop on Geometric Graph Theory, June, 2011, held in Australia, organized by Peter Eades and Seok-Hee Hong, supported by IPDF funding from the University of Sydney.

** Independently, another linear time algorithm is reported in [2].

of a graph is NP-complete [8]. On the positive side, it has been shown that the problem of testing maximal 1-planarity of a graph G can be solved in linear time if a *rotation system* (i.e., the circular ordering of edges for each vertex) is given by Eades et al [9].

The existence of a 1-planar embedding does not guarantee the existence of a straight-line 1-planar drawing, as shown by Eggleton [10] and Thomassen [11]. However, recently Hong et al. [12] give a linear time testing algorithm, and a linear time drawing algorithm to construct such a drawing if it exists. Very recently, the more general problem on straight-line drawability of embedded graphs is studied by Nagamochi [13].

Eggleton [10] introduced the investigation of *outer-1-planar graphs*: a graph is outer-1-planar if it has a 1-planar drawing in which every vertex is on the outer face. Examples of outer-1-planar graph drawings are shown in Fig. 1(a), (b), (c) and (d); in Fig. 1(e) a graph that has no outer-1-planar drawing is illustrated. Eggleton describes a number of geometric, topological, and combinatorial properties of outer-1-planar graphs.

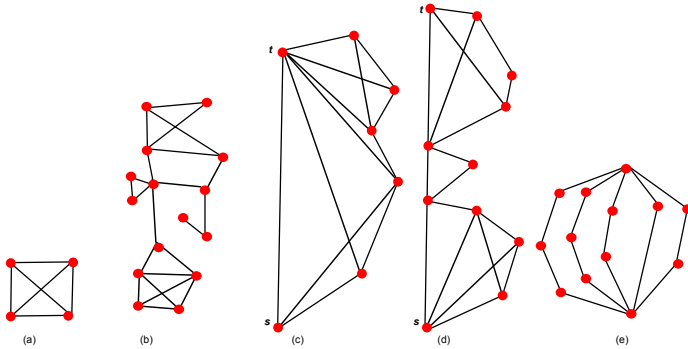


Fig. 1. (a), (b), (c) and (d) are examples of outer-1-planar graph drawings. (a) illustrates the only triconnected outer-1-planar graph. (c) and (d) are examples of graphs that are one-sided outer-1-planar (OSOIP) with respect to (s, t) . (d) illustrates a graph with no outer-1-planar drawing.

In this paper, we investigate algorithmics of outer-1-planar graphs. More specifically, we describe a linear time algorithm to test outer-1-planarity of a given graph G .

Theorem 1. *There is a linear time algorithm to test whether a graph is outer-1-planar. The algorithm produces an outer-1-planar embedding if it exists.*

To prove Theorem 1, we define a sub-class of outer-1-planar graphs as follows. Suppose that G is a graph with vertices s and t . Let $G_{+(s,t)}$ denote the graph obtained by adding the edge (s, t) , if this edge is not already in G . If $G_{+(s,t)}$ has an outer-1-planar embedding in which the edge (s, t) is completely on the outer face, then we say that G is *one-sided-outer-1-planar (OSOIP)* with respect to (s, t) . Examples of such graphs are shown in Fig. 1(c) and (d). For these graphs, we prove the following result.

Theorem 2. *There is a linear time algorithm to test whether a biconnected graph G is one-sided outer-1-planar with respect to a given edge (s, t) of G . The algorithm produces a one-sided outer-1-planar embedding if it exists.*

Section 4 describes an algorithm to test whether a graph is one-sided outer-1-planar, and Section 5 shows how to use one-sided outer-1-planarity to test outer-1-planarity. The adaptation of the algorithms of Sections 4 and 5 to construct an embedding is straightforward, and described in Section 6. In conclusion, Section 7 cites drawing algorithms and discusses future work.

2 Terminology

In this Section we define the terminology used throughout the paper.

A *topological graph* or *embedding* $G = (V, E)$ is a representation of a simple graph in the plane where each vertex is a point and each edge is a Jordan arc between the points representing its endpoints.

Two edges *cross* if they have a point in common, other than their endpoints. The point in common is a *crossing*. To avoid some pathological cases, some standard non-degeneracy conditions apply: (1) two edges intersect in at most one point; (2) an edge does not contain a vertex other than its endpoints; (3) no edge crosses itself; (4) edges must not meet tangentially; (5) no three edges share a crossing point; and (6) no two edges that share an endpoint cross.

A topological graph is *1-planar* if no edge has more than one crossing. A graph is *1-planar* if it has a 1-planar embedding. A graph is *outer-1-planar* if it has a 1-planar embedding in which every vertex is on the outer face. The aim of this paper is to give an algorithm to test whether a graph is outer-1-planar, and to provide an outer-1-planar embedding if it exists.

Our algorithm uses an *SPQR tree* to represent the decomposition of a biconnected graph into triconnected components. We recall some basic terminology of SPQR trees; for details, see [14]. Each node ν in the SPQR tree is associated with a graph called the *skeleton* of ν , denoted by $\sigma(\nu)$. There are four types of nodes ν in the SPQR tree: (1) S-nodes, where $\sigma(\nu)$ is a simple cycle with at least 3 vertices; (2) P-nodes, where $\sigma(\nu)$ consists of two vertices connected by at least 3 edges; (3) Q-nodes, where $\sigma(\nu)$ consists of two vertices connected by a real edges and a virtual edge; and (4) R-nodes, where $\sigma(\nu)$ is a simple triconnected graph. We treat the SPQR tree as a rooted tree by choosing an arbitrary node as its root. Note that every leaf is a Q-node and that the root is not a Q-node.

Let ρ be the parent of an internal node ν . The graph $\sigma(\rho)$ has exactly one *virtual edge* e in common with $\sigma(\nu)$; this is the *parent virtual edge* of $\sigma(\nu)$, and a *child virtual edge* in $\sigma(\rho)$. We denote the graph formed by the union of $\sigma(\nu)$ over all descendants ν of ρ by G_ρ .

If G is an outer-1-planar graph, then $\sigma(\nu)$ and G_ν are outer-1-planar graphs, using the embedding induced from G . If G_ν is a one-sided outer-1-planar (OSO1P) graph with respect to the parent virtual edge (s, t) of ν then we say that ν is a one-sided outer-1-planar (OSO1P) node with respect to (s, t) .

For this paper, we need to define a specific type of S-node. Suppose that μ is an S-node with parent separation pair (u, v) . A tail at u for μ is a Q-node child (that is, a real edge) with parent virtual edge (u, x) for some vertex x .

Further, we need to define a specific type of P-node. A P-node ν is almost one-sided outer-1-planar (AOSOIP) with respect to (the directed edge) (s, t) if G_ν consists of a parallel composition of an OSOIP graph with respect to (s, t) and an S-node μ such that μ has a tail at t and μ is OSOIP with respect to (s, t) . See Fig. 2 for examples.

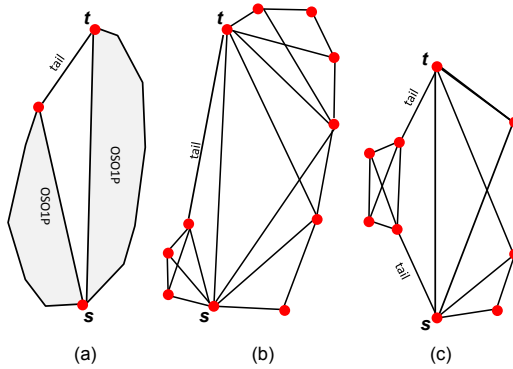


Fig. 2. An AOSOIP graph consists of a parallel composition of an OSOIP graph and an OSOIP S-node with a tail. (a) The general shape of a graph that is AOSOIP with respect to (s, t) . (b) A graph that is AOSOIP with respect to (s, t) . (c) A graph that is AOSOIP with respect to both (s, t) and (t, s) .

3 Structural Results

In this Section we present structural results that support the algorithms defined in the subsequent sections.

First we note that the only triconnected outer-1-planar graph is K_4 , embedded as depicted in Fig. 1(a).

Lemma 1. *If G is outer-1-planar and triconnected, then G is isomorphic to K_4 and every outer-1-planar drawing of G has exactly one crossing.*

Proof. Suppose that G is an outer-1-planar embedding of a triconnected graph; we can assume that G is maximal in the sense that no edge can be added without destroying the property of outer-1-planarity. Eggleton [10] shows that the outer face is a simple cycle γ .

Suppose that (a, b) and (c, d) are a pair of edges, neither on γ , that cross at point p . Suppose that a precedes c in clockwise order around γ . Suppose that there is at least one vertex v that lies between a and c on γ , as shown in Fig. 3.

All edges incident with the vertices between a and c on γ must have both endpoints in the region r bounded by γ , the curve ap and the curve cp . Removing a and c separates v from the remainder of the graph; this contradicts the triconnectivity of G , and we can deduce that there is no such vertex v .

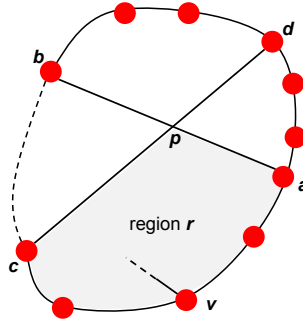


Fig. 3. Here the pair a, c must be a separation pair, since all edges incident with vertices between a and c on the outer face γ must have both endpoints in the region r .

Using the same argument, we can show that the only vertices on γ are a, b, c and d ; thus G is K_4 . □

Secondly, we note that we can restrict our attention to the biconnected case.

Lemma 2. *A graph is outer-1-planar if and only if its biconnected components are outer-1-planar.*

Next we present a simple fact about how an edge can cross a cycle in an outer-1-planar embedding.

Lemma 3. *Suppose that γ is a cycle in an outer-1-planar graph G and G' is an outer-1-planar embedding of G . Suppose that an edge (u, v) is not on γ but crosses an edge of γ in G' . Then either u or v is on γ .*

The next result is a relatively technical but fundamental Lemma about embeddings of paths which share endpoints, illustrated in Fig. 4. A path from a vertex s to a vertex t is *non-trivial* if it contains more than two vertices. If an edge from a path p_1 crosses an edge from a path p_2 then we say that p_1 *crosses* p_2 .

Lemma 4. *Suppose that P is a set of paths between two vertices s and t . Let G be the union of the paths in P , and let G' be an outer-1-planar embedding of G . Then $|P| \leq 5$, and:*

- (a) *If $|P| \geq 3$ and an edge from one non-trivial path $p_1 \in P$ crosses an edge from another non-trivial path $p_2 \in P$ then this crossing occurs between an edge incident with s and an edge incident with t .*
- (b) *If $|P| = 3$ and all paths in P are non-trivial, then there are two paths p_1 and p_2 in P such that there is exactly one crossing between edges of p_1 and edges of p_2 ; furthermore, every edge in the third path is on the outer face.*
- (c) *If $|P| = 3$ and one path in P is trivial and is on the outer face, then there are two paths p_1 and p_2 in P such that there is exactly one crossing between edges of p_1 and edges of p_2 .*

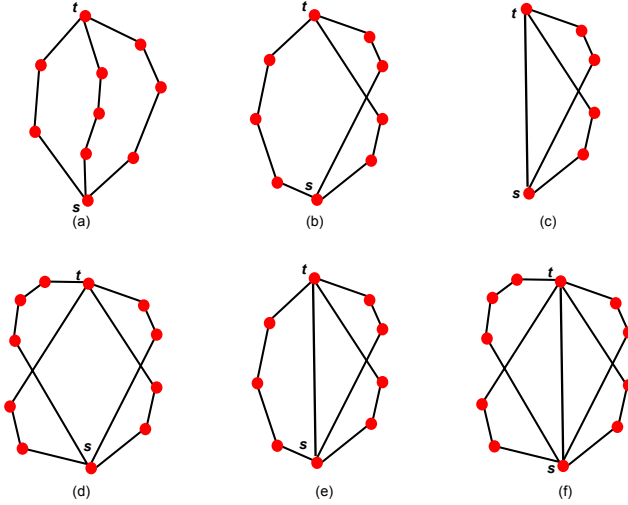


Fig. 4. Embeddings of paths that share endpoints. (a) A planar embedding. (b) An outer-1-planar embedding of 3 non-trivial paths. (c) An outer-1-planar embedding of 3 paths, where one path is trivial. (d) An outer-1-planar embedding of 4 non-trivial paths. (e) An outer-1-planar embedding of 4 paths, where one path is trivial. (f) An outer-1-planar embedding of 5 paths.

- (d) If P contains 4 non-trivial paths, then we can divide P into two pairs of paths $\{p_1, p_2\}$ and $\{p_3, p_4\}$ such that there is exactly one crossing between edges of p_1 and edges of p_2 and exactly one crossing between edges of p_3 and edges of p_4 , and there are no other crossings.
- (e) If $|P| = 4$ and P contains a trivial path, then there are two paths p_1 and p_2 in P such that there is exactly one crossing between edges of p_1 and edges of p_2 , and one non-trivial path in P is on the outer face.
- (f) If $|P| = 5$ then one path in P is trivial, and we can divide the other paths into two pairs of paths $\{p_1, p_2\}$ and $\{p_3, p_4\}$ such that there is exactly one crossing between edges of p_1 and edges of p_2 and exactly one crossing between edges of p_3 and edges of p_4 , and there are no other crossings.

4 Testing OSO1P and AOSO1P

In this Section we describe a linear time algorithm that takes a graph G and vertices s and t of G as input and tests whether G has an OSO1P or AOSO1P embedding with respect to (s, t) .

From Lemma 2, we only need to consider biconnected graphs. Our algorithm computes the SPQR tree T and then works from the leaves of T upward toward the root, computing boolean labels $OSO1P(\nu, s, t)$ and $AOSO1P(\nu, s, t)$ that indicate whether

ν is OSO1P or AOSO1P with respect to (s, t) . The label $OSO1P(\nu, s, t)$ is computed for each node ν of T , and the label $AOSO1P(\nu, s, t)$ is computed for every P-node ν .

Algorithm test-One-Sided-Outer-1-Planar

1. Construct the SPQR tree T of G .
2. Traverse T bottom up, and for each node ν of T with parent virtual edge (s, t) :
 - (a) if ν is a Q-node then return true.
 - (b) elseif ν is an R-node then return $OSO1P(\nu, s, t)$ as described in Section 4.1, using the values $OSO1P(\nu', s', t')$ for each child ν' of ν with child virtual edge (s', t') .
 - (c) elseif ν is a P-node then return $OSO1P(\nu, s, t)$ and $AOSO1P(\nu, s, t)$, as described in Section 4.2, using the values $OSO1P(\nu', s', t')$ for each child ν' of ν with child virtual edge (s', t') .
 - (d) else /* ν is an S-node */ then return $OSO1P(\nu, s, t)$, as described in Section 4.3, using the values $OSO1P(\nu', s', t')$ and $AOSO1P(\nu, s', t')$ for each child ν' of ν with child virtual edge (s', t') .

The time complexity of Step 1 is linear [14,15]. Step 2(a) is trivial and takes constant time for each Q-node. We show below that Steps 2(b), 2(c), and 2(d) each take time proportional to the number of children of the node ν . Summing over all nodes of the SPQR tree T results in linear time for the whole algorithm.

The cases for R-nodes and P-nodes are quite straightforward, and we deal with them first in Sections 4.1 and 4.2. The extension for the computation of the AOSO1P property for P-nodes is again straightforward and described in Section 4.2. The case for S-nodes is a little more involved, and we deal with this in Section 4.3.

4.1 R-nodes

Lemma 1 can be generalised to R-nodes as in the following Lemma.

Lemma 5. *Suppose that ν is an R-node of the SPQR tree of a biconnected graph G ; suppose that (u, v) is its parent virtual edge. Then G_ν is OSO1P with respect to (u, v) if and only if:*

1. $\sigma(\nu)$ is isomorphic to K_4 ; and
2. an edge (u, a) of $\sigma(\nu)$ with $a \neq v$ incident with u represents a child Q-node of ν , an edge (v, b) of $\sigma(\nu)$ with $b \neq u$ represents a child Q-node of ν , and (u, a) crosses (v, b) ; and
3. for every child ν' of ν , ν' is OSO1P with respect to (c, d) , where the parent virtual edge of ν' is (c, d) .

An algorithm to test whether an R-node ν is OSO1P can be derived directly from Lemma 5. It is clear that, given the boolean labels $OSO1P(\nu_i, s, t)$ for each child ν_i of ν , the algorithm runs in time proportional to the number of children of ν .

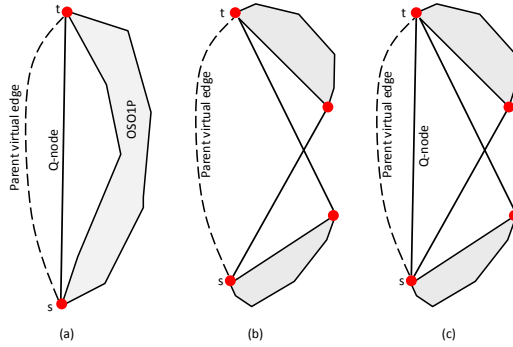


Fig. 5. Possibilities for an OSO1P P-node

4.2 P-nodes

Here we use Lemma 4 to show that a P-node can have at most three children, as depicted in Fig. 5. More specifically, we have the following Lemma:

Lemma 6. *Suppose that ν is a P-node of the SPQR tree of a biconnected graph G ; suppose that (s, t) is its parent virtual edge. Then G_ν is OSO1P with respect to (s, t) if and only if either:*

- (a) ν has two children, of which one is a Q-node (s, t) , and the other is OSO1P with respect to (s, t) ; or
- (b) ν has two children, of which one is an S-node with tail at s which is OSO1P with respect to (s, t) , and the other is an S-node with tail at t which is OSO1P with respect to (s, t) ; or
- (c) ν has three children, of which one is a Q-node (s, t) , one is an S-node with tail at s which is OSO1P with respect to (s, t) , and the other is an OSO1P S-node with tail at t which is OSO1P with respect to (s, t) .

It is straightforward to extend Lemma 6 to test whether a node ν is AOSO1P, using the definition of AOSO1P together with Lemma 6.

Algorithms to test whether a P-node ν is OSO1P or AOSO1P can be derived directly from Lemma 6. It is clear that, given the boolean labels $OSO1P(\nu', s, t)$ for each child ν' of ν , the algorithm runs in time proportional to the size of the skeleton $\sigma(\nu)$ of ν .

4.3 S-nodes

Suppose that ν is an S-node with children $\nu_1, \nu_2, \dots, \nu_k$, where the parent virtual edge of ν_i is (s_{i-1}, s_i) , as shown in Fig. 6(a).

If every child ν_i is OSO1P with respect to (s_{i-1}, s_i) , then clearly ν is OSO1P with respect to (s_0, s_k) ; however, the converse is false. Consider the example shown in Fig. 6(b). Here ν is OSO1P with respect to (s_0, s_k) . However, the child ν_2 is not OSO1P

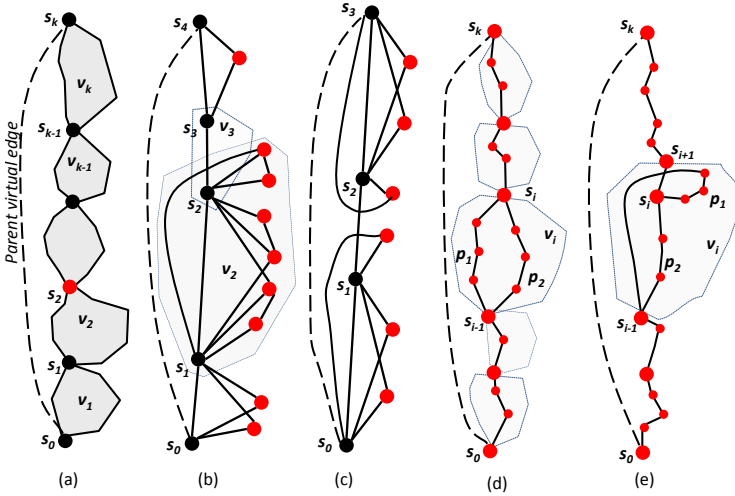


Fig. 6. (a) An S-node. (b) An OSOIP S-node with a child (ν_2) that is not OSOIP. (c) An S-node that satisfies the conditions of Lemma 7 but is not OSOIP. (d) Two paths p_1 and p_2 in the graph G_{ν_i} . (e) The path p_1 crosses the edge (s_i, s_{i+1}) .

with respect to (s_1, s_2) (by Lemma 6). Note that ν_3 is a Q-node, and an edge from the skeleton of ν_2 crosses this edge. In fact the example shown in Fig. 6(b) illustrates the necessary conditions stated in the next lemma.

Lemma 7. *Suppose that ν is an S-node with children $\nu_1, \nu_2, \dots, \nu_k$, where the parent virtual edge of ν_i is (s_{i-1}, s_i) . Suppose that G_ν is OSOIP with respect to (s_0, s_k) . Then for $1 \leq i \leq k$, either:*

- (a) ν_i is OSOIP with respect to (s_{i-1}, s_i) ; or
- (b) $i < k$, ν_i is AOSOIP with respect to (s_i, s_{i-1}) , and ν_{i+1} is a Q-node; or
- (c) $i > 1$, ν_i is AOSOIP with respect to (s_{i-1}, s_i) , and ν_{i-1} is a Q-node.

Lemma 7 gives necessary conditions for an S-node to be OSOIP. However the conditions are not sufficient. Consider, for example, the graph shown in Fig. 6(c). This satisfies the necessary conditions as in Lemma 7, but it is not OSOIP. The problem is that the Q-node represented by the edge (s_1, s_2) has two crossings, one with an edge of the AOSOIP graph at the top and one with an edge of the AOSOIP graph at the bottom. Nevertheless, this situation does not occur when $k = 2$, and we shall show that in this case the conditions of Lemma 7 are sufficient. One can express sufficient conditions for an S-node to be OSOIP in a recursive way, as in the following Lemma. If ν is an S-node with children $\nu_1, \nu_2, \dots, \nu_k$ then we denote the series combination of graphs $G_{\nu_1}, G_{\nu_2}, \dots, G_{\nu_k}$ by $G(\nu_1, \nu_2, \dots, \nu_k)$.

Lemma 8. *Suppose that ν is an S-node with children $\nu_1, \nu_2, \dots, \nu_k$, where the parent virtual edge of ν_i is (s_{i-1}, s_i) . Then G_ν is OSOIP with respect to (s_0, s_k) if and only if either:*

1. G_{ν_1} is OSOIP with respect to (s_0, s_1) and $G(\nu_2, \nu_3, \dots, \nu_k)$ is OSOIP with respect to (s_1, s_k) ; or
2. ν_1 is a Q-node, G_{ν_2} is AOSOIP with respect to (s_1, s_2) , and $G(\nu_3, \nu_4, \dots, \nu_k)$ is OSOIP with respect to (s_2, s_k) ; or
3. G_{ν_1} is AOSOIP with respect to (s_1, s_0) , ν_2 is a Q-node, and $G(\nu_3, \nu_4, \dots, \nu_k)$ is OSOIP with respect to (s_2, s_k) .

Lemma 8 leads to the recursive algorithm for S-nodes; see [1]. The algorithm runs in time proportional to the number of children of ν .

This completes the proof of Theorem 2.

5 Testing Outer-1-Planarity

Once we compute the labels $OSO1P(\nu, s, t)$ and $AOSO1P(\nu, s, t)$ for all internal nodes ν of the SPQR tree, we can test whether the whole graph (that is, the root ρ) is outer-1-planar. This requires separate tests depending on the type of the root node. See Fig. 7.

We can require the root node to be an R-node or a P-node, since if the SPQR tree contains no R-node and no P-node, then the graph is a cycle and thus outerplanar. Both tests for an R-node and a P-node are detailed below.

For R-nodes, we have the following Lemma.

Lemma 9. *Suppose that ρ is an R-node at the root of the SPQR tree. Then G is outer-1-planar if and only if*

1. $\sigma(\rho)$ is isomorphic to K_4 , and
2. at least two children of ρ are Q-nodes, and
3. for every child node ν' of $\sigma(\rho)$ with parent virtual edge (a, b) , $G_{\nu'}$ is OSOIP with respect to (a, b) .

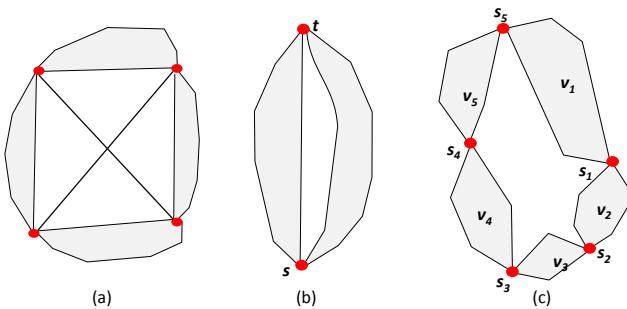


Fig. 7. (a) An R-node at the root. (b) P-node at the root. (c) S-node at the root.

It is clear that one can test the conditions of Lemma 9 in constant time, as long as the labels $OSO1P(\nu', a, b)$ for all children ν' of ρ have already been computed.

For P-nodes, the following result applies.

Lemma 10. *Suppose that ρ is a P-node at the root of the SPQR tree. Then G is outer-1-planar if and only if it is a parallel composition of two OSOIP graphs.*

Using Lemma 4, one can show that the number of children of a P-node ρ at the root is bounded (in fact, at most 5). It follows that the number of ways to partition the children is bounded by a constant. Thus we can define a constant time algorithm to implement Lemma 10 (given that the labels $OSOIP(\nu', a, b)$ for all children ν' of ρ have already been computed).

This completes the proof of Theorem 1.

6 Outer-1-Planar Embedding

One can construct a one-sided outer-1-planar embedding of an input graph G using an extension of the methods in Section 4. The methods for R-nodes and S-nodes described in Lemmas 5 and 6 define crossings; treating these crossings as dummy vertices gives a planar graph G^* . A one-sided outer-1-planar embedding of G is a specific planar embedding of G^* .

Every planar embedding of G^* is defined by an orientation and an ordering for nodes ν in the SPQR tree with respect to the parent separation pair of ν . For P-nodes, R-nodes, and S-nodes, it is possible to “flip” the orientation of ν around its parent separation pair. For P nodes, a left-right order for the children can be chosen. To produce an outer-1-planar embedding we use the same bottom-up strategy as in **Algorithm test-One-Sided-Outer-1-Planar** in Section 4. Throughout the algorithm we maintain an embedding; in particular we keep track of the outside face. At each node ν , we “flip” ν so that all vertices of G_ν lie on the outside face. Also, at each P-node ν , we order the children of ν so that all vertices of G_ν lie on the outside face. This requires linear time manipulation of the SPQR tree, using methods outlined in [14].

7 Conclusion

The algorithm presented in this paper takes a graph G as input and determines whether it has an outer-1-planar embedding. We show that if such an embedding does exist, then we can compute it in linear time.

Given the topological embedding computed by our algorithm, the methods of Eggleton [10] can be used to construct a straight-line drawing. In fact, Eggleton gives conditions that determine whether a given set of points support a straight-line outer-1-planar drawing. Dekhordi et al. [16] show further that every outer-1-planar topological embedding has a straight-line RAC (right-angle crossing) drawing, at the cost of exponential area.

Many algorithms for drawing outerplanar graphs exist, with a number of properties (see [17,18]). It would be interesting to see if these results extend to outer-1-planar graphs.

References

1. Hong, S.H., Eades, P., Katoh, N., Liotta, G., Schweitzer, P., Suzuki, Y.: A linear-time algorithm for testing outer-1-planarity. TR-IT-IVG-2013-01. Technical Report, School of IT, University of Sydney (June 2013)
2. Auer, C., Bachmaier, C., Brandenburg, F.J., Gleißner, A., Hanauer, K., Neuwirth, D., Reislhuber, J.: Recognizing outer 1-planar graphs in linear time. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 107–118. Springer, Heidelberg (2013)
3. Ringel, G.: Ein Sechsfarbenproblem auf der Kugel. Abh. Math. Sem. Univ. Hamburg 29, 107–117 (1965)
4. Borodin, O.V.: Solution of the Ringel problem on vertex-face coloring of planar graphs and coloring of 1-planar graphs. *Metody Diskret. Analiz.* 41, 12–26 (1984)
5. Pach, J., Tóth, G.: Graphs drawn with few crossings per edge. *Combinatorica* 17, 427–439 (1997)
6. Fabrici, I., Madaras, T.: The structure of 1-planar graphs. *Discrete Mathematics* 307, 854–865 (2007)
7. Suzuki, Y.: Re-embeddings of maximum 1-planar graphs. *SIAM J. Discrete Math.* 24, 1527–1540 (2010)
8. Korzhik, V.P., Mohar, B.: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory* 72, 30–71 (2013)
9. Eades, P., Hong, S.-H., Katoh, N., Liotta, G., Schweitzer, P., Suzuki, Y.: Testing maximal 1-planarity of graphs with a rotation system in linear time - (extended abstract). In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 339–345. Springer, Heidelberg (2013)
10. Eggleton, R.: Rectilinear drawings of graphs. *Utilitas Mathematica* 29, 149–172 (1986)
11. Thomassen, C.: Rectilinear drawings of graphs. *Journal of Graph Theory* 12, 335–341 (1988)
12. Hong, S.H., Eades, P., Liotta, G., Poon, S.H.: Fáry's theorem for 1-planar graphs. In: [19], pp. 335–346
13. Nagamochi, H.: Straight-line drawability of embedded graphs. Technical Report 2013-005, Department of Applied Mathematics and Physics, Kyoto University, Japan (2013)
14. Battista, G.D., Tamassia, R.: On-line maintenance of triconnected components with spqr-trees. *Algorithmica* 15, 302–318 (1996)
15. Gutwenger, C., Mutzel, P.: A Linear Time Implementation of SPQR-Trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
16. Dehkordi, H.R., Eades, P.: Every outer-1-plane graph has a right angle crossing drawing. *Int. J. Comput. Geometry Appl.* 22, 543–558 (2012)
17. Frati, F.: Straight-line drawings of outerplanar graphs in $o(dn \log n)$ area. *Comput. Geom.* 45, 524–533 (2012)
18. Knauer, K.B., Micek, P., Walczak, B.: Outerplanar graph drawings with few slopes. In: [19], pp. 323–334
19. Gudmundsson, J., Mestre, J., Viglas, T. (eds.): COCOON 2012. LNCS, vol. 7434. Springer, Heidelberg (2012)

Straight-Line Grid Drawings of 3-Connected 1-Planar Graphs

Md. Jawaherul Alam^{1,*}, Franz J. Brandenburg^{2,**}, and Stephen G. Kobourov^{1,*}

¹ Department of Computer Science, University of Arizona, USA

{mjalam, kobourov}@cs.arizona.edu

² University of Passau, 94030 Passau, Germany

brandenb@informatik.uni-passau.de

Abstract. A graph is 1-planar if it can be drawn in the plane such that each edge is crossed at most once. In general, 1-planar graphs do not admit straight-line drawings. We show that every 3-connected 1-planar graph has a straight-line drawing on an integer grid of quadratic size, with the exception of a single edge on the outer face that has one bend. The drawing can be computed in linear time from any given 1-planar embedding of the graph.

1 Introduction

Since Euler's Königsberg bridge problem dating back to 1736, planar graphs have provided interesting problems in theory and in practice. Using the elaborate techniques of a canonical ordering and Schnyder realizers, every planar graph can be drawn on a grid of quadratic size, and such drawings can be computed in linear time [15, 21]. The area bound is asymptotically optimal, since the nested triangle graphs are planar graphs and require $\Omega(n^2)$ area [10]. The drawing algorithms were refined to improve the area requirement or to admit convex representations, i.e., where each inner face is convex [5, 8] or strictly convex [1].

However, most graphs are nonplanar and recently, there have been many attempts to study larger classes of graphs. Of particular interest are 1-planar graphs, which in a sense are one step beyond planar graphs. They were introduced by Ringel [20] in an attempt to color a planar graph and its dual. Although it is known that a 3-connected planar graph and its dual have a straight-line 1-planar drawing [24] and even on a quadratic grid [13], little is known about general 1-planar graphs. It is NP-hard to recognize 1-planar graphs [16, 18] in general, although there is a linear-time testing algorithm [11] for maximal 1-planar graphs (i.e., where no additional edge can be added without violating 1-planarity) given the the circular ordering of incident edges around each vertex. A 1-planar graph with n vertices has at most $4n - 8$ edges [4, 14, 19] and this upper bound is tight. On the other hand straight-line drawings of 1-planar graphs may have at most $4n - 9$ edges and this bound is tight [9]. Hence not all 1-planar graphs admit straight-line drawings. Unlike planar graphs, maximal 1-planar graphs can be much sparser with only $2.64n$ edges [6].

* Research supported in part by NSF grants CCF-1115971 and DEB 1053573.

** Research supported by the Deutsche Forschungsgemeinschaft, DFG, grant Br 835/18-1.

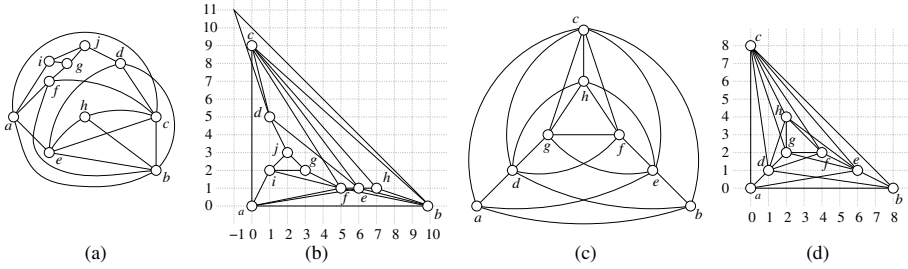


Fig. 1. (a)–(b) A 3-connected 1-planar graph and its straight-line grid drawing (with one bend in one edge), (c)–(d) another 3-connected 1-planar graph and its straight-line grid drawing

Thomassen [23] refers to 1-planar graphs as graphs with *cross index 1* and proved that an embedded 1-planar graph can be turned into a straight-line drawing if and only if it excludes B - and W -configurations; see Fig. 2. These forbidden configurations were first discovered by Eggleton [12] and used by Hong *et al.* [17], who show that the configurations can be detected in linear time if the embedding is given. They also proved that there is a linear time algorithm to convert a 1-planar embedding without B - and W -configurations into a straight-line drawing, but without bounds for the drawing area.

In this paper we settle the straight-line grid drawing problem for 3-connected 1-planar graphs. First we compute a *normal form* for an embedded 1-planar graph with no B -configuration and at most one W -configuration on the outer face. Then, after augmenting the graph with as many planar edges as possible and then deleting the crossing edges, we find a 3-connected planar graph, which is drawn with strictly convex faces using an extension of the algorithm of Chrobak and Kant [8]. Finally the pairs of crossing edges are reinserted into the convex faces. This gives a straight-line drawing on a grid of quadratic size with the exception of a single edge on the outer face, which may need one bend (and this exception is unavoidable); see Fig. 1. In addition, the drawing is obtained in linear time from a given 1-planar embedding.

2 Preliminaries

A *drawing* of a graph G is a mapping of G into the plane such that the vertices are mapped to distinct points and each edge is a Jordan arc between its endpoints. A drawing is *planar* if the Jordan arcs of the edges do not cross and it is *1-planar* if each edge is crossed at most once. Note that crossings between edges incident to the same vertex are not allowed. For example, K_5 and K_6 are 1-planar graphs. An *embedding* of a graph is planar (resp. 1-planar) if it admits a planar (resp. 1-planar) drawing. An embedding specifies the *faces*, which are topologically connected regions. The unbounded face is the *outer face*. A face in a planar graph is specified by a cyclic sequence of edges on its boundary (or equivalently by the cyclic sequence of the endpoints of the edges).

Accordingly, a *1-planar embedding* $\mathcal{E}(G)$ specifies the faces in a 1-planar drawing of G including the outer face. A 1-planar embedding is a witness for 1-planarity. In particular, $\mathcal{E}(G)$ describes the pairs of crossing edges and the faces where the edges

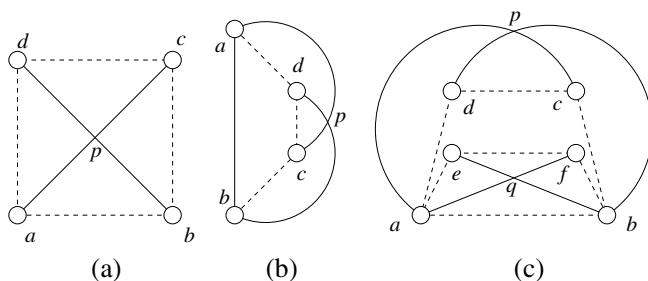


Fig. 2. (a) An augmented X -configuration, (b) an augmented B -configuration, (c) an augmented W -configuration. The graphs induced by the solid edges are called an X -configuration (a), a B -configuration (b), and a W -configuration (c).

cross and has linear size. Each pair of *crossing edges* (a, c) and (b, c) induces a *crossing point* p . Call the segment of an edge between the vertex and the crossing point a *half-edge*. Each half-edge is *impermeable*, analogous to the edges in planar drawings, in the sense that no edge can cross such a half-edge without violating the 1-planarity of the embedding. The non-crossed edges are called *planar*. A *planarization* G^\times is obtained from $\mathcal{E}(G)$ by using the crossing points as regular vertices and replacing each crossing edge by its two half-edges. A 1-planar embedding $\mathcal{E}(G)$ and its planarization share equivalent embeddings, and each face is given by a list of edges and half-edges defining it, or equivalently, by a list of vertices and crossing points of the edges and half edges.

Eggleton [12] raised the problem of recognizing 1-planar graphs with rectilinear drawings. He solved this problem for outer-1-planar graphs (1-planar graphs with all vertices on the outer-cycle) and proposed three forbidden configurations. Thomassen [23] solved Eggleton's problem and characterized the rectilinear 1-planar embeddings by the exclusion of B - and W -configurations; see Fig. 2. Hong *et al.* [17], obtain a similar characterization where the B - and W -configurations are called the ‘‘Bulgari’’ and ‘‘Gucci’’ graphs. They also show that all occurrences of these configurations can be computed in linear time from a given 1-planar embedding.

Definition 1. Consider a 1-planar embedding $\mathcal{E}(G)$:

A B -configuration consists of an edge (a, b) and two edges (a, c) and (b, d) which cross in some point p such that c and d lie in the interior of the triangle (a, b, p) . Here (a, b) is called the *base of the configuration*.

An X -configuration consists of a pair (a, c) and (b, d) of crossing edges which does not form a B -configuration.

A W -configuration consists of two pairs of edges (a, c) , (b, d) and (a, f) , (b, e) which cross in points p and q , such that c, d, e, f lie in the interior of the quadrangle a, p, b, q . Here again the edge (a, b) , if present is the *base*.

Observe that for all these configurations the base edges may be crossed by another edge, whereas the crossing edges are impermeable; see Fig 2.

Thomassen [23] and Hong *et al.* [17] proved that for a 1-planar embedding to admit straight-line drawing, B - and W -configurations must be excluded:

Proposition 1. *A 1-planar embedding $\mathcal{E}(G)$ admits a straight-line drawing with a topologically equivalent embedding if and only if it does not contain a B - or a W -configuration.*

Augment a given 1-planar embedding $\mathcal{E}(G)$ by adding as many edges to $\mathcal{E}(G)$ as possible so that G remains a simple graph and the newly added edges are planar in $\mathcal{E}(G)$. We call such an embedding a *planar-maximal* embedding of G and the operation *planar-maximal augmentation*. (Note that Hong *et al.* [17] color the planar edges of a 1-planar embedding red and call a planar-maximal augmentation a *red augmentation*.) The *planar skeleton* $\mathcal{P}(\mathcal{E}(G))$ consists of the planar edges of a planar-maximal augmentation. It is a planar embedded graph, since all pairs of crossing edges are omitted. Note that the planar augmentation and the planar skeleton are defined for an embedding, not for a graph. A graph may have different embeddings which give rise to different configurations and augmentations. The notion of planar-maximal embedding is different from the notions of maximal 1-planar embeddings and maximal 1-planar graphs, which are such that the addition of any edge violates 1-planarity (or simplicity) [6].

The following claim, proven in many earlier papers [6, 14, 17, 22, 23], shows that a crossing pair of edges induces a K_4 in planar-maximal embedding, since missing edges of a K_4 can be added without inducing new crossings.

Lemma 1. *Let $\mathcal{E}(G)$ be a planar-maximal 1-planar embedding of a graph G and let (a, c) and (b, d) be two crossing edges. Then the four vertices $\{a, b, c, d\}$ induce a K_4 .*

By Lemma 1, for a planar-maximal embedding each X -, B -, and W -configuration is augmented by additional edges. Here we define these augmented configurations.

Definition 2. *Let $\mathcal{E}(G)$ be a planar-maximal 1-planar embedding of a graph G . An augmented X -configuration consists of a K_4 with vertices (a, b, c, d) such that the edges (a, c) and (b, d) cross inside the quadrangle $abcd$. An augmented B -configuration consists of a K_4 with vertices (a, b, c, d) such that the edges (a, c) and (b, d) cross beyond the boundary of the quadrangle $abcd$. An augmented W -configuration consists of two K_4 's (a, b, c, d) and (a, b, e, f) one of which is in an augmented X -configuration and the other in an augmented B -configuration.*

For an augmented X - or augmented B -configuration, the edges not inducing a crossing with other edges in the configuration define a cycle, we call it the skeleton. In each configuration, the edges on the outer-boundary of the embedded configuration and not inducing a crossing with other edges in the configuration are the base edges.

Using the results of Thomassen [23] and Hong *et al.* [17], we can now characterize when a planar-maximal 1-planar embedding of a graph admits a straight-line drawing:

Lemma 2. *Let $\mathcal{E}(G)$ be a planar-maximal 1-planar embedding of a graph G . Then there is a straight-line 1-planar drawing of G with a topologically equivalent embedding as $\mathcal{E}(G)$ if and only if $\mathcal{E}(G)$ does not contain an augmented B -configuration.*

Proof. Assume $\mathcal{E}(G)$ contains an augmented B -configuration. Then it contains a B -configuration and has no straight-line 1-planar drawing by Proposition 1. Conversely, if $\mathcal{E}(G)$ has no straight-line 1-planar drawing then by Proposition 1 it contains at least one B - or W -configuration. Since Γ is a planar-maximal embedding, by Lemma 1 each

crossing edge pair in $\mathcal{E}(G)$ induces a K_4 . Thus the dotted edges in Fig. 2(b)–(c) must be present in any B - or W - configuration, inducing an augmented B -configuration. \square

The *normal form* for an embedded 1-planar graph $\mathcal{E}(G)$ is obtained by first adding the four planar edges to form a K_4 for each pair of crossing edges while routing them closely to the crossing edges and then removing old duplicate edges if necessary. Such an embedding of a 1-planar graph is a normal embedding of it. A *normal planar-maximal augmentation* for an embedded 1-planar graph is obtained by first finding a normal form of the embedding and then by a planar-maximal augmentation.

Lemma 3. *Given a 1-planar embedding $\mathcal{E}(G)$, the normal planar-maximal augmentation of $\mathcal{E}(G)$ can be computed in linear time.*

Proof. First augment each crossing of two edges (a, c) and (b, d) to a K_4 , such that the edges (a, b) , (b, c) , (c, d) , (d, a) are added and in case of a duplicate the former edge is removed. Then all augmented X -configurations are empty and contain no vertices inside their skeletons. Next triangulate all faces which do not contain a half-edge, a crossing edge, or a crossing point. Each step can be done in linear time. \square

3 Characterization of 3-Connected 1-Planar Graphs

Here we characterize 3-connected 1-planar graphs by a normal embedding, where the crossings are augmented to K_4 's such that the resulting augmented X -configurations have vertex-empty skeletons and there is no augmented B -configuration except for at most one augmented W -configuration with a pair of crossing edges in the outer face.

Let $\mathcal{E}(G)$ be a 1-planar embedding of a graph G . Each pair of crossing edges induces a crossing point and the crossing edges and their half-edges are *impermeable* as they cannot be crossed by other edges without violating 1-planarity. An *impermeable path* in $\mathcal{E}(G)$ is an internally-disjoint sequence $P = v_1, p_1, v_2, p_2, \dots, v_n, p_n, v_{n+1}$, where v_1, v_2, \dots, v_{n+1} are (regular) vertices of G , p_1, p_2, \dots, p_n are crossing points in $\mathcal{E}(G)$ and (v_i, p_i) , (p_i, v_{i+1}) for each $i \in \{1, 2, \dots, n\}$ are half edges. If $v_{n+1} = v_0$, then P is an *impermeable cycle*. An impermeable cycle is *separating* when it has vertices both inside and outside of it, since deleting its vertices disconnects G .

Lemma 4. *Let $G = (V, E)$ be a 3-connected 1-planar graph with a planar-maximal 1-planar embedding $\mathcal{E}(G)$. Then the following conditions hold.*

- A. (i) *Two augmented B -configurations or two augmented X -configurations cannot be on the same side of a common base edge.*
- (ii) *Suppose an augmented B -configuration B and an augmented X -configuration X are on the same side of a common base edge (a, b) . Let p and q be the crossing points for X and B , respectively and let $R(X)$ and $R(B)$ be the regions inside the skeletons of X and B . Then all vertices of $V \setminus \{a, b\}$ are inside the impermeable cycle $apbq$ if $R(X) \subset R(B)$; otherwise all vertices of $V \setminus \{a, b\}$ are outside the impermeable cycle $apbq$.*
- B. (i) *If two augmented B -configurations are on opposite sides of a common base edge (a, b) , with crossing points p and q , respectively, then all the vertices of $V \setminus \{a, b\}$ are inside the impermeable cycle $apbq$.*

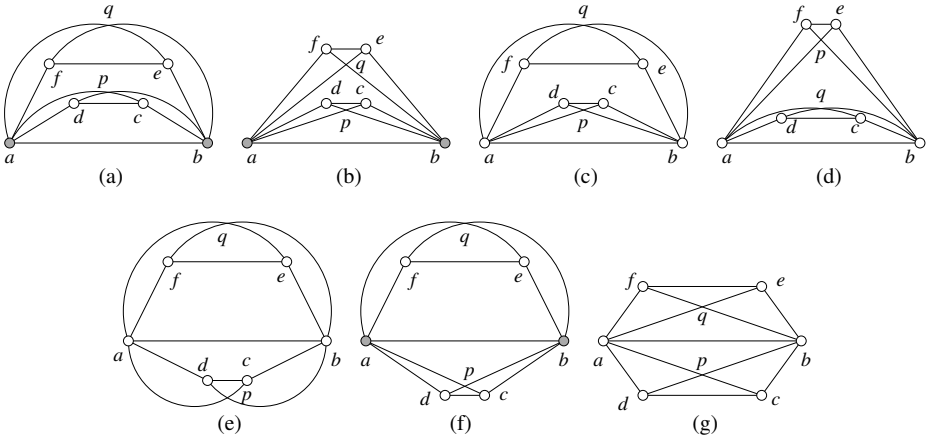


Fig. 3. Illustration for the proof of Lemma 4

- (ii) If two augmented X -configurations are on opposite sides of a common base edge (a, b) , with crossing points p and q , respectively, then all the vertices of $V \setminus \{a, b\}$ are outside the impermeable cycle $apbq$.
- (iii) An augmented B -configuration and an augmented X -configuration cannot share a common base edge from opposite sides.

Proof. Condition A.(i) and B.(iii) hold because each of these configurations induces a separating impermeable $apbq$ cycle in $\mathcal{E}(G)$ with only two (regular) vertices from G , a contradiction with the 3-connectivity of G ; see Fig. 3(a)–(b) and (f). Similarly, if any of the Conditions A.(ii) and B.(i)–(ii) is not satisfied, then the impermeable cycle $apbq$ becomes separating and hence the pair $\{a, b\}$ becomes separation pair of G , again a contradiction with the 3-connectivity of G ; see Fig. 3(c)–(d), (e) and (g). \square

Corollary 1. *Let G be a 3-connected 1-planar graph with a planar-maximal 1-planar embedding $\mathcal{E}(G)$. Then no three crossing edge-pairs in $\mathcal{E}(G)$ share the same base edge.*

Proof. Each crossing edge pair induces either an augmented B - or an augmented X -configuration. This fact along with Lemma 4[A.(i), B.(iii)] yields the corollary. \square

Lemma 5. *Let G be a 3-connected 1-planar graph. Then there is a planar-maximal 1-planar embedding $\mathcal{E}(G^*)$ of a supergraph G^* of G so that $\mathcal{E}(G^*)$ contains at most one augmented W -configuration in the outer face and no other augmented B -configuration, and each augmented X -configuration in $\mathcal{E}(G^*)$ contains no vertex inside its skeleton.*

Proof. Let $\mathcal{E}(G)$ be a 1-planar embedding of G . We claim that by a normal planar-maximal augmentation of $\mathcal{E}(G)$ we get the desired embedding of a supergraph of G . Note that due to the edge-rerouting this operation converts any B -configuration whose base is not shared with another configuration into an X -configuration; see Fig. 4(a). If a base edge is shared by two B -configurations, they are converted into one W -configuration and by Lemma 4 this W -configuration is on the outer face; see Fig. 4(b).

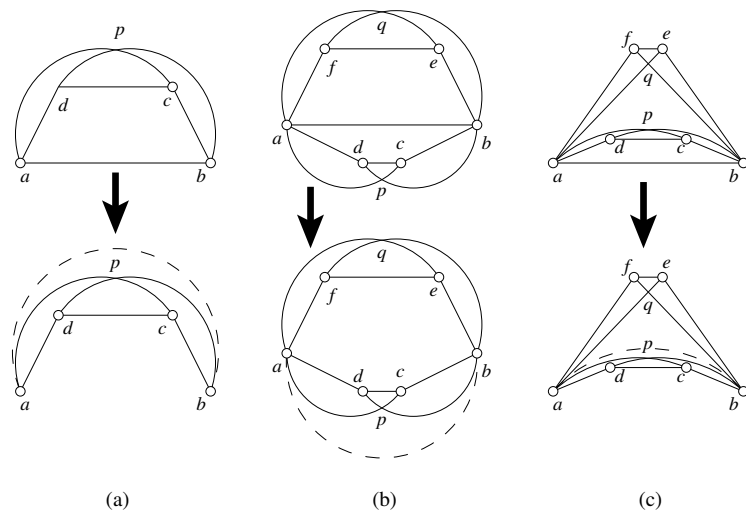


Fig. 4. Illustration for the proof of Lemma 5

By Corollary 1, a base edge cannot be shared by more than two B -configurations. Furthermore this operation does not create any new B -configuration. It also makes the skeleton of any augmented X -configuration vertex-empty; by Lemma 4 a base edge can be shared by at most two augmented X -configurations from opposite sides and if it is shared by two augmented X -configurations, the interior of the induced impermeable cycle is empty; see Fig. 4(c). \square

Lemma 5 together with Proposition 1 implies the following:

Theorem 1. *A 3-connected 1-planar graph admits a straight-line 1-planar drawing except for at most one edge in the outer face.*

4 Grid Drawings

In the previous section we showed that a 3-connected 1-planar graph has a straight-line 1-planar drawing, with the exception of a single edge in the outer face. We now strengthen this result and show that there is straight-line grid drawing with $O(n^2)$ area, which can be constructed in linear time from a given 1-planar embedding.

The algorithm takes an embedding $\mathcal{E}(G)$ and computes a normal planar-maximal augmentation. Consider the planar skeleton $\mathcal{P}(\mathcal{E}(G))$ for the embedding. If there is an augmented W -configuration and a crossing in the outer face, one crossing edge on the outer face is kept and the other crossing edge is treated separately. Thus the outer face of $\mathcal{P}(\mathcal{E}(G))$ is a triangle and the inner faces are triangles or quadrangles. Each quadrangle comes from an augmented X -configuration. It must be drawn strictly convex, such that the crossing edges can be re-inserted. This is achieved by an extension of the convex grid drawing algorithm of Chrobak and Kant [8], which itself is an extension of the shifting method of de Fraysseix, Pach and Pollack [15]. Since the faces are at most

quadrangles, we can avoid three collinear vertices and the degeneration to a triangle by an extra unit shift. Note that our drawing algorithm achieves an area of $(2n-2) \times (2n-3)$, while the general algorithms for strictly convex grid drawings [1, 7] require larger area, since strictly convex drawings of n -gons need $\Omega(n^3)$ area [2]. Barany and Rote give a strictly convex grid drawing of a planar graph on a $14n \times 14n$ grid if the faces are at most 4-gons, and on a $2n \times 2n$ grid if, in addition, the outer face is a triangle. However, their approach is quite complex and does not immediately yield these bounds. It is also not clear how to use this approach for planar graphs in our 1-planar graph setting, in particular when we have an unavoidable W -configuration in the outer face.

The algorithm of Chrobak and Kant and in particular the computation of a canonical decomposition presumes a 3-connected planar graph. Thus the planar skeleton of a 3-connected 1-planar graph must be 3-connected, which holds except for the K_4 , when it is embedded as an augmented X -configuration. This results parallels the fact that the planarization of a 3-connected 1-planar graph is 3-connected [14].

Lemma 6. *Let G be a graph with a planar-maximal 1-planar embedding $\mathcal{E}(G)$ such that it has no augmented B -configuration and each augmented X -configuration in $\mathcal{E}(G)$ has no vertex inside its skeleton. Then the planar skeleton $\mathcal{P}(\mathcal{E}(G))$ is 3-connected.*

We will prove Lemma 6 by showing that there is no separation pair in $\mathcal{P}(\mathcal{E}(G))$. First we obtain a planar graph H from G as follows. Let (a, c) and (b, d) be a pair of crossing edges that form an augmented X -configuration X in Γ . We then delete the two edges (a, c) , (b, d) ; add a vertex u and the edges (a, u) , (b, u) , (c, u) , (d, u) to triangulate the face $abcd$. Call v a *cross-vertex* and call this operation *cross-vertex insertion* on X . We then obtain H from G by cross-vertex insertion on each augmented X -configuration. Call H a *planarization* of G and denote the set of all the cross-vertices by U . Then $\mathcal{P}(\mathcal{E}(G)) = H \setminus U$. Before proving Lemma 6 we consider several properties of H , the planarization of the 1-planar graph.

Lemma 7. *Let $G = (V, E)$ be a graph with a planar-maximal 1-planar embedding $\mathcal{E}(G)$ such that $\mathcal{E}(G)$ contains no augmented B -configuration and each augmented X -configuration in $\mathcal{E}(G)$ contains no vertex inside its skeleton. Let H be a planarization of G , where U is the set of cross-vertices. Then the following conditions hold.*

- (a) H is a maximal planar graph (except if H is the K_4 in an X -configuration)
- (b) Each vertex of U has degree 4.
- (c) U is an independent set of H .
- (d) There is no separating triangle of H containing any vertex from U .
- (e) There is no separating 4-cycle of H containing two vertices from U .

Proof. For convenience, we call each vertex in $V - U$ a *regular vertex*.

- (a) Since H is a planar graph, we only show that each face of H is a triangle. Each crossing edge pair in Γ induces an augmented X -configuration whose skeleton has no vertex in its interior. Hence each face of H containing a crossing vertex is a triangle. Again, Hong *et al.* [17] showed that in a planar-maximal 1-planar embedding a face with no crossing vertices is a triangle. Thus H is a maximal planar graph.

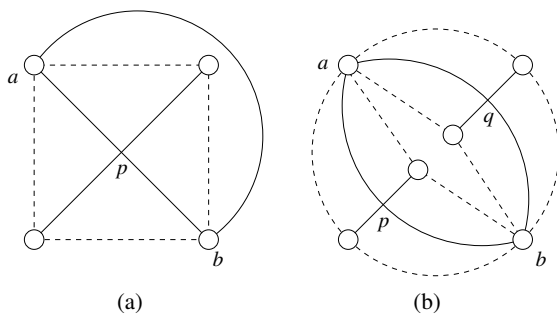


Fig. 5. Illustration for the proof of Lemma 6

- (b)–(c) These two conditions follow from the fact that the neighborhood of each crossing vertex consists of exactly four regular vertices that form the skeleton of the corresponding augmented X-configuration.
- (d) For a contradiction suppose a vertex $u \in U$ participates in a separating triangle T of H . Since the neighborhood of u forms the skeleton of the corresponding augmented X-configuration X , the other two vertices, say a and b , in T are regular vertices. The edge (a, b) cannot form a base edge for X , since if it did, then the interior of the separating triangle T would be contained in the interior of the skeleton for X and hence would be empty. Assume therefore that a and b are not consecutive on the skeleton of X . In this case the edge (a, b) is a crossing edge in G and hence has been deleted when constructing H ; see Fig. 5(a).
- (e) Suppose two vertices $u, v \in U$ participate in a separating 4-cycle T of H . Due to Condition (c), assume that $T = abuv$, where a, b are regular vertices. If the two vertices a, b are adjacent in H , assume without loss of generality that the edge (a, b) is drawn inside the interior of T . Then the interior of at least one of the two triangles abu and abv is non-empty, inducing a separating triangle in H , a contradiction with Condition (d). We thus assume that the two vertices a and b are not adjacent in H . Then for both the augmented X-configurations X and Y , corresponding to the two crossing vertices u and v , the two vertices u and v are not consecutive on their skeleton. This implies that the crossing edge (a, b) participates in two different augmented X-configurations in Γ , again a contradiction; see Fig. 5(b). \square

We are now ready to prove Lemma 6.

Proof (Lemma 6). Assume for a contradiction that $\mathcal{P}(\mathcal{E}(G))$ is not 3-connected. Then there exists some separation pair $\{a, b\}$ in $\mathcal{P}(\mathcal{E}(G))$. Let H be the planarization of G , where U is the set of cross-vertices. Then $S = U \cup \{a, b\}$ is a separating set for H . Take a minimal separating set $S' \subset S$ such that no proper subset of S' is a separating set. Since H is a maximal planar graph (from Lemma 7(a)), S' forms a separating cycle [3]. The 3-connectivity of the maximal planar graph H implies $|S'| \geq 3$. Again since S' contains at most two regular vertices a, b and no two cross-vertices can be adjacent in H (Lemma 7(c)), $|S'| < 5$. Hence S' is a separating triangle or a separating 4-cycle with at most two regular vertices; we get a contradiction with Lemma 7(d)–(e). \square

Finally, we describe our algorithm for straight-line grid drawings. This drawing algorithm is based on an extension of the algorithm of Chrobak and Kant [8] for computing a convex drawing of a planar 3-connected graph. For convenience we refer to this algorithm as the CK-algorithm and we begin with a brief overview. Let $G = (V, E)$ be an embedded 3-connected graph and let (u, v) be an edge on the outer-cycle of G . The CK-algorithm starts by computing a *canonical decomposition* of G , which is an ordered partition V_1, V_2, \dots, V_t of V such that the following conditions hold:

- (i) For each $k \in \{1, 2, \dots, t\}$, the graph G_k induced by the vertices $V_1 \cup \dots \cup V_k$ is 2-connected and its outer-cycle C_k contains the edge (u, v) .
- (ii) G_1 is a cycle, V_t is a singleton $\{z\}$, where $z \notin \{u, v\}$ is on the outer-cycle of G .
- (iii) For each $k \in \{2, \dots, t-1\}$ the following conditions hold:
 - If V_k is a singleton $\{z\}$, then z is on the outer face of G_{k-1} and has at least one neighbor in $G - G_k$.
 - If V_k is a chain $\{z_1, \dots, z_l\}$, each z_i has at least one neighbor in $G - G_k$, z_1, z_l have one neighbor each on C_{k-1} and no other z_i has neighbors on G_{k-1} .

For each $k \in \{1, 2, \dots, t\}$, we say that the vertices that belong to V_k have *rank* k . We call a vertex of G_k *saturated* if it has no neighbor in $G - G_k$. The CK-algorithm starts by drawing the edge (u, v) with a horizontal line-segment of unit length. Then for $k = 1, 2, \dots, t$, it incrementally completes the drawing of G_k . Let $C_{k-1} = \{(u = w_1, \dots, w_p, \dots, w_q, \dots, w_r = v)\}$ with $1 \leq p < q \leq r$ where w_p and w_q are the leftmost and the rightmost neighbor of vertices in V_k . Then the vertices of V_k are placed above the vertices w_p, \dots, w_q . Assume that $V_k = \{z_1, \dots, z_l\}$. Then z_1 is placed on the vertical line containing w_p if w_p is saturated in G_k ; otherwise it is placed on the vertical line one unit to the right of w_p . On the other hand, z_l is placed on the negative diagonal line (i.e., with -45° slope) containing w_q . If v_k is a singleton then $z = z_1 = z_l$ is placed at the intersection of these two lines. Otherwise (after necessary shifting of w_q and other vertices), the vertices z_1, \dots, z_l are placed on consecutive vertical lines one unit apart from each other. In order to make sure that this shifting operation does not disturb planarity or convexity, each vertex v is associated with an “under-set” $U(v)$ and whenever v is shifted, all vertices in $U(v)$ are also shifted along with v . Thus the edges between vertices of any $U(v)$ are in a sense *rigid*.

Theorem 2. *Given a 1-planar embedding $\mathcal{E}(G)$ of a 3-connected graph G , a straight-line drawing on the $(2n - 2) \times (2n - 3)$ grid can be computed in linear time. Only one edge on the outer face may require one bend.*

Proof. Assume that $\mathcal{E}(G)$ is a normal planar-maximal embedding; otherwise we compute one by a normal planar-maximal augmentation in linear time by Lemma 3. Consider the planar skeleton $\mathcal{P}(\mathcal{E}(G))$. If there is no unavoidable W-configuration on the outer face of the maximal planar augmentation, then the outer-cycle of $\mathcal{P}(\mathcal{E}(G))$ is a triangle. Otherwise we add one of the crossing edges in the outer face to $\mathcal{P}(\mathcal{E}(G))$ to make the outer-cycle a triangle. The other crossing edge is treated separately. By Lemma 6, $\mathcal{P}(\mathcal{E}(G))$ is 3-connected, its outer face is a triangle (a, b, c) and the inner faces are triangles or quadrangles, where the latter result from augmented X-configurations and are in one-to-one correspondence to pairs of crossing edges.

We wish to obtain a planar straight-line grid drawing of $\mathcal{P}(\mathcal{E}(G))$ such that all quadrangles are strictly convex. Although the CK-algorithm draws any 3-connected planar graph of n vertices on a grid of size $(n - 1) \times (n - 1)$ with convex faces, the faces are not necessarily strictly convex [8]. Hence we must modify the algorithm so that all quadrangles are strictly convex. Note that by the assignment of the under-sets, the CK-algorithm guarantees that once a face is drawn strictly convex, it would remain strictly convex after any subsequent shifting of vertices.

For $\mathcal{P}(\mathcal{E}(G))$ each V_k is either a single vertex or a pair with an edge, since the faces are at most quadrangles. If V_k is an edge (z_1, z_2) then, by the definition of the canonical decomposition, exactly one quadrangle face $w_p z_1 z_2 w_q$ is formed and by construction this face is drawn convex. We thus assume that V_k contains a single vertex, say v . Let $C_{k-1} = \{(u = w_1, \dots, w_p, \dots, w_q, \dots, w_r = v)\}$ with $1 \leq p < q \leq r$ where w_p and w_q are the leftmost and the rightmost neighbors of vertices in V_k . Then the new faces created by the insertion of v are all drawn strictly convex unless there is some quadrangle $vw_{p'} w_{p'+1}$ where $p < p' < q$ and $w_{p'-1}, w_{p'}, w_{p'+1}$ are collinear in the drawing of G_{k-1} . In this case $w_{p'}$ must be saturated in G_{k-1} and this occurs in the CK-algorithm only when the line containing $w_{p'-1}, w_{p'}, w_{p'+1}$ is a vertical line or a negative diagonal (with -45° slope). In the former case, w_{p-1} should have also been saturated in G_{k-1} , which is not possible since v is its neighbor. It is thus sufficient to ensure that no saturated vertex of G_k is in the negative diagonal of both its left and right neighbors on C_k . We do this by the following extension of the CK-algorithm.

Suppose v is placed above w_q with slope -45 , w_q was placed above its rightmost lower neighbor $w'_{q'}$ with slope -45 , and there is a quadrangle $(v, w_q, w'_{q'}, u)$ for some vertex u with higher rank to be placed later. Then shift $w'_{q'}$ by one extra unit to the right when v or u is placed. This implies a bend at w_q and a strictly convex angle above w_q .

The CK-algorithm starts by placing the first two vertices one unit away and it requires a unit shift to the right for each following vertex. On the other hand, a 1-planar graph has at most $n - 2$ pairs of crossing edges. Hence, there are $g \leq n - 3$ augmented X-configurations, each of which induces a quadrangle in the planar skeleton. Thus the width and height are $n - 1 + g$, which is bounded by $2n - 4$. The vertices a, b, c of the outer triangle are placed at the grid points $(0, 0), (0, n - 1 + g), (n - 1 + g, 0)$.

If the graph had an unavoidable W -configuration in the outer face, we need a post-processing phase to draw the extra edge (b, d) , which induces a crossing with the edge (a, c) . Since a is the leftmost lower neighbor of d when d is placed and d is not saturated, d is placed at $(1, j)$ for some $j < n - 2 + g$. Shift b one unit to the right, insert a bend at $(-1, n + g)$, one diagonal unit left above c and route (b, d) via the bend point. \square

5 Conclusion and Future Work

We showed that 3-connected 1-planar graphs can be embedded on $O(n) \times O(n)$ integer grid, so that edges are drawn as straight-line segments (except for at most one edge on the outerface that requires a bend). Moreover, the algorithm is simple and runs in linear time given a 1-planar embedding. Note that even a path may require exponential area for a given fixed 1-planar embedding, e.g., [17]. Recognition of 1-planar graphs is NP-hard [18]. How hard is the recognition of planar-maximal 1-planar graphs?

References

1. Bárány, I., Rote, G.: Strictly convex drawings of planar graphs. *Documenta Mathematica* 11, 369–391 (2006)
2. Bárány, I., Tokushige, N.: The minimum area of convex lattice n -gons. *Combinatorica* 24(2), 171–185 (2004)
3. Baybars, I.: On k -path hamiltonian maximal planar graphs. *Discrete Mathematics* 40(1), 119–121 (1982)
4. Bodendiek, R., Schumacher, H., Wagner, K.: Bemerkungen zu einem Sechsfarbenproblem von G. Ringel. *Abh. aus dem Math. Seminar der Univ. Hamburg* 53, 41–52 (1983)
5. Bonichon, N., Felsner, S., Mosbah, M.: Convex drawings of 3-connected plane graphs. *Algorithmica* 47(4), 399–420 (2007)
6. Brandenburg, F.J., Eppstein, D., Gleißner, A., Goodrich, M.T., Hanauer, K., Reislhuber, J.: On the density of maximal 1-planar graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 327–338. Springer, Heidelberg (2013)
7. Chrobak, M., Goodrich, M.T., Tamassia, R.: Convex drawings of graphs in two and three dimensions. In: *Symposium on Computational Geometry*, pp. 319–328 (1996)
8. Chrobak, M., Kant, G.: Convex grid drawings of 3-connected planar graphs. *International Journal of Computational Geometry and Applications* 7(3), 211–223 (1997)
9. Didimo, W.: Density of straight-line 1-planar graph drawings. *Information Processing Letter* 113(7), 236–240 (2013)
10. Dolev, D., Leighton, T., Trickey, H.: Planar embedding of planar graphs. In: *Advances in Computing Research*, pp. 147–161 (1984)
11. Eades, P., Hong, S.H., Katoh, N., Liotta, G., Schweitzer, P., Suzuki, Y.: Testing maximal 1-planarity of graphs with a rotation system in linear time. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 339–345. Springer, Heidelberg (2013)
12. Eggleton, R.B.: Rectilinear drawings of graphs. *Utilitas Math.* 29, 149–172 (1986)
13. Erten, C., Kobourov, S.G.: Simultaneous embedding of a planar graph and its dual on the grid. *Theory of Computing Systems* 38(3), 313–327 (2005)
14. Fabrici, I., Madaras, T.: The structure of 1-planar graphs. *Discrete Mathematics* 307(7-8), 854–865 (2007)
15. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. *Combinatorica* 10(1), 41–51 (1990)
16. Grigoriev, A., Bodlaender, H.L.: Algorithms for graphs embeddable with few crossings per edge. *Algorithmica* 49(1), 1–11 (2007)
17. Hong, S.-H., Eades, P., Liotta, G., Poon, S.-H.: Fáry’s theorem for 1-planar graphs. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 335–346. Springer, Heidelberg (2012)
18. Korzhik, V.P., Mohar, B.: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory* 72(1), 30–71 (2013)
19. Pach, J., Tóth, G.: Graphs drawn with a few crossings per edge. *Combinatorica* 17, 427–439 (1997)
20. Ringel, G.: Ein Sechsfarbenproblem auf der Kugel. *Abh. aus dem Math. Seminar der Univ. Hamburg* 29, 107–117 (1965)
21. Schnyder, W.: Embedding planar graphs on the grid. In: Johnson, D.S. (ed.) *Symposium on Discrete Algorithms*, pp. 138–148 (1990)
22. Suzuki, Y.: Re-embeddings of maximum 1-planar graphs. *SIAM Journal of Discrete Mathematics* 24(4), 1527–1540 (2010)
23. Thomassen, C.: Rectilinear drawings of graphs. *Journal of Graph Theory* 12(3), 335–341 (1988)
24. Tutte, W.T.: How to draw a graph. *Proc. London Math. Society* 13(52), 743–768 (1963)

New Bounds on the Maximum Number of Edges in k -Quasi-Planar Graphs

Andrew Suk^{1,*} and Bartosz Walczak^{2,3,**}

¹ Massachusetts Institute of Technology, Cambridge, MA
asuk@math.mit.edu

² École Polytechnique Fédérale de Lausanne, Switzerland
bartosz.walczak@epfl.ch

³ Theoretical Computer Science Department, Faculty of Mathematics
and Computer Science, Jagiellonian University, Kraków, Poland
walczak@tcs.uj.edu.pl

Abstract. A topological graph is k -quasi-planar if it does not contain k pairwise crossing edges. An old conjecture states that for every fixed k , the maximum number of edges in a k -quasi-planar graph on n vertices is $O(n)$. Fox and Pach showed that every k -quasi-planar graph with n vertices and no pair of edges intersecting in more than $O(1)$ points has at most $n(\log n)^{O(\log k)}$ edges. We improve this upper bound to $2^{\alpha(n)^c} n \log n$, where $\alpha(n)$ denotes the inverse Ackermann function, and c depends only on k . We also show that every k -quasi-planar graph with n vertices and every two edges have at most one point in common has at most $O(n \log n)$ edges. This improves the previously known upper bound of $2^{\alpha(n)^c} n \log n$ obtained by Fox, Pach, and Suk.

1 Introduction

A *topological graph* is a graph drawn in the plane so that its vertices are represented by points and its edges are represented by curves connecting the corresponding points. The curves are always *simple*, that is, they do not have self-intersections. The curves are allowed to intersect each other, but they cannot pass through vertices except for their endpoints. Furthermore, the edges are not allowed to have tangencies, that is, if two edges share an interior point, then they must properly cross at that point. We only consider graphs without parallel edges or loops. Two edges of a topological graph *cross* if their interiors share a point. A topological graph is *simple* if any two of its edges have at most one point in common, which can be either a common endpoint or a crossing.

It follows from Euler's polyhedral formula that every topological graph on n vertices and with no two crossing edges has at most $3n - 6$ edges. A graph

* Supported by an NSF Postdoctoral Fellowship and by Swiss National Science Foundation Grant 200021-125287/1.

** Supported by Swiss National Science Foundation Grant 200020-144531 and by Polish MNiSW Grant 884/N-ESF-EuroGIGA/10/2011/0 within the ESF EuroGIGA project GraDR.

is called k -quasi-planar if it can be drawn as a topological graph with no k pairwise crossing edges. Hence, a graph is 2-quasi-planar if and only if it is planar. According to an old conjecture (see Problem 1 in Section 9.6 of [4]), for any fixed $k \geq 2$ there exists a constant c_k such that every k -quasi-planar graph on n vertices has at most $c_k n$ edges. Agarwal, Aronov, Pach, Pollack, and Sharir [2] were the first to prove this conjecture for *simple* 3-quasi-planar graphs. Later, Pach, Radoičić, and Tóth [14] generalized the result to *all* 3-quasi-planar graphs. Ackerman [1] proved the conjecture for $k = 4$.

For larger values of k , several authors have proved upper bounds on the maximum number of edges in k -quasi-planar graphs under various conditions on how the edges are drawn. These include but are not limited to [5,7,8,15,19]. In 2008, Fox and Pach [7] showed that every k -quasi-planar graph with n vertices and no pair of edges intersecting in more than t points has at most $n(\log n)^{c \log k}$ edges, where c depends only on t . In this paper, we improve the exponent of the polylogarithmic factor from $O(\log k)$ to $1 + o(1)$ for fixed t .

Theorem 1. *Every k -quasi-planar graph with n vertices and no pair of edges intersecting in more than t points has at most $2^{\alpha(n)^c} n \log n$ edges, where $\alpha(n)$ denotes the inverse of the Ackermann function, and c depends only on k and t .*

Recall that the *Ackermann function* $A(n)$ is defined as follows. Let $A_1(n) = 2n$, and $A_k(n) = A_{k-1}(A_k(n-1))$ for $k \geq 2$. In particular, we have $A_2(n) = 2^n$, and $A_3(n)$ is an exponential tower of n two's. Now let $A(n) = A_n(n)$, and let $\alpha(n)$ be defined as $\alpha(n) = \min\{k \geq 1: A(k) \geq n\}$. This function grows much slower than the inverse of any primitive recursive function.

For *simple* topological graphs, Fox, Pach, and Suk [8] showed that every k -quasi-planar simple topological graph on n vertices has at most $2^{\alpha(n)^c} n \log n$ edges, where c depends only on k . We establish the following improvement.

Theorem 2. *Every k -quasi-planar simple topological graph on n vertices has at most $c_k n \log n$ edges, where c_k depends only on k .*

We start the proofs of both theorems with a reduction to the case of topological graphs containing an edge that intersects every other edge. This reduction introduces the $O(\log n)$ factor for the bound on the number of edges. Then, the proof of Theorem 1 follows the approaches of Valtr [19] and Fox, Pach, and Suk [8], using a result on generalized Davenport-Schinzel sequences, which we recall in Section 3. Although the proofs in [19] and [8] heavily depend on the assumption that any two edges have at most one point in common, we are able to remove this condition by establishing some technical lemmas in Section 4. In Section 5, we finish the proof of Theorem 1. The proof of Theorem 2, which relies on a recent coloring result due to Lason, Micek, Pawlik, and Walczak [10], is given in Section 6.

2 Initial Reduction

We call a collection C of curves in the plane *decomposable* if there is a partition $C = C_1 \cup \dots \cup C_w$ such that each C_i contains a curve intersecting all other curves

in C_i , and for $i \neq j$, no curve in C_i crosses nor shares an endpoint with a curve in C_j .

Lemma 1 (Fox, Pach, Suk [8]). *There is an absolute constant $c > 0$ such that every collection C of $m \geq 2$ curves such that any two of them intersect in at most t points has a decomposable subcollection of size at least $\frac{cm}{t \log m}$.*

In the proofs of both Theorem 1 and Theorem 2, we establish a (near) linear upper bound on the number of edges under the additional assumption that the graph has an edge intersecting every other edge. Once this is achieved, we use the following lemma to infer an upper bound for the general case.

Lemma 2 (implicit in [8]). *Let G be a topological graph on n vertices such that no two edges have more than t points in common. Suppose that for some constant β , every subgraph G' of G containing an edge that intersects every other edge of G' has at most $\beta|V(G')|$ edges. Then G has at most $c_t \beta n \log n$ edges, where c_t depends only on t .*

Proof. By Lemma 1, there is a decomposable subset $E' \subset E(G)$ such that $|E'| \geq c'_t |E(G)| / \log |E(G)|$, where c'_t depends only on t . Hence there is a partition $E' = E_1 \cup \dots \cup E_w$, such that each E_i has an edge e_i that intersects every other edge in E_i , and for $i \neq j$, the edges in E_i are disjoint from the edges in E_j . Let V_i denote the set of vertices that are the endpoints of the edges in E_i , and let $n_i = |V_i|$. By the assumption, we have $|E_i| \leq \beta n_i$ for $1 \leq i \leq w$. Hence

$$\frac{c'_t |E(G)|}{\log |E(G)|} \leq |E'| \leq \sum_{i=1}^w \beta n_i \leq \beta n.$$

Since $|E(G)| \leq n^2$, we obtain $|E(G)| \leq 2(c'_t)^{-1} \beta n \log n$. □

3 Generalized Davenport-Schinzel Sequences

A sequence $S = (s_1, \dots, s_m)$ is called l -regular if any l consecutive terms of S are pairwise different. For integers $l, m \geq 2$, the sequence $S = (s_1, \dots, s_{lm})$ is said to be of *type up*(l, m) if the first l terms are pairwise different and $s_i = s_{i+l} = \dots = s_{i+(m-1)l}$ for $1 \leq i \leq l$. In particular, every sequence of type up(l, m) is l -regular. For convenience, we will index the elements of an up(l, m) sequence as

$$S = (s_{1,1}, \dots, s_{l,1}, s_{1,2}, \dots, s_{l,2}, \dots, s_{1,m}, \dots, s_{l,m}),$$

where $s_{1,1}, \dots, s_{l,1}$ are pairwise different and $s_{i,1} = \dots = s_{i,m}$ for $1 \leq i \leq l$.

Theorem 3 (Klazar [9]). *For $l \geq 2$ and $m \geq 3$, every l -regular sequence over an n -element alphabet that does not contain a subsequence of type up(l, m) has length at most*

$$n \cdot l \cdot 2^{(lm-3)} \cdot (10l)^{10\alpha(n)^{lm}}.$$

For more results on generalized Davenport-Schinzel sequences, see [13,16,17].

4 Intersection Pattern of Curves

In this section, we will prove several technical lemmas on the intersection pattern of curves in the plane. We will always assume that no two curves are tangent, and that if two curves share an interior point, then they must properly cross at that point.

Lemma 3. *Let λ_1 and λ_2 be disjoint simple closed curves. Let C be a collection of m curves with one endpoint on λ_1 , the other endpoint on λ_2 , and no other common points with λ_1 or λ_2 . If no k members of C pairwise cross, then C contains $\lceil m/(k-1)^2 \rceil$ pairwise disjoint members.*

Proof. Let G be the intersection graph of C . Since G does not contain a clique of size k , by Turán’s theorem, $|E(G)| \leq (1 - 1/(k-1))m^2/2$. Hence there is a curve $a \in C$ and a subset $S \subset C$, such that $|S| \geq m/(k-1) - 1$ and a is disjoint from every curve in S . We order the elements in $S \cup \{a\}$ as $a_0, a_1, \dots, a_{|S|}$ in clockwise order as their endpoints appear on λ_1 , starting with $a_0 = a$. Now we define the partial order \prec on the pairs in S so that $a_i \prec a_j$ if $i < j$ and a_i is disjoint from a_j . A simple geometric observation shows that \prec is indeed a partial order. Since S does not contain k pairwise crossing members, by Dilworth’s theorem [6], $S \cup \{a\}$ contains $\lceil m/(k-1)^2 \rceil$ pairwise disjoint members. \square

A collection of curves with a common endpoint v is called a *fan* with *apex* v . Let $C = \{a_1, \dots, a_m\}$ be a fan with apex v , and $\gamma = \gamma_1 \cup \dots \cup \gamma_m$ be a curve with endpoints p and q partitioned into m subcurves $\gamma_1, \dots, \gamma_m$ that appear in order along γ from p to q . We say that C is *grounded* by $\gamma_1 \cup \dots \cup \gamma_m$, if

- (i) γ does not contain v ,
- (ii) each a_i has its other endpoint on γ_i .

We say that C is *well-grounded* by $\gamma_1 \cup \dots \cup \gamma_m$ if C is grounded by $\gamma_1 \cup \dots \cup \gamma_m$ and each a_i intersects γ only within γ_i . Note that both notions depend on a particular partition $\gamma = \gamma_1 \cup \dots \cup \gamma_m$.

Lemma 4. *Let $C = \{a_1, \dots, a_m\}$ be a fan grounded by a curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$. If each a_i intersects γ in at most t points, then there is a subfan $C' = \{a_{i_1}, \dots, a_{i_r}\} \subset C$ with $i_1 < \dots < i_r$ and $r = \lfloor \log_{t+1} m \rfloor$ that is grounded by a subcurve $\gamma' = \gamma'_1 \cup \dots \cup \gamma'_r \subset \gamma$. Moreover,*

- (i) $\gamma'_j \supset \gamma_{i_j}$ for $1 \leq j \leq r$,
- (ii) a_{i_j} intersects γ' only within $\gamma'_1 \cup \dots \cup \gamma'_j$ for $1 \leq j \leq r$.

Proof. We proceed by induction on m . The base case $m \leq t$ is trivial. Now assume that $m \geq t + 1$ and the statement holds up to $m - 1$. Since a_1 intersects γ in at most t points, there exists an integer j such that a_1 is disjoint from $\gamma_j \cup \gamma_{j+1} \cup \dots \cup \gamma_{j+\lfloor m/(t+1) \rfloor - 1}$. By the induction hypothesis applied to $\{a_j, a_{j+1}, \dots, a_{j+\lfloor m/(t+1) \rfloor - 1}\}$ and the curve $\gamma_j \cup \gamma_{j+1} \cup \dots \cup \gamma_{j+\lfloor m/(t+1) \rfloor - 1}$, we obtain a subfan $C^* = \{a_{i_2}, \dots, a_{i_r}\}$ of $r - 1 = \lfloor \log_{t+1} \lfloor m/(t+1) \rfloor \rfloor = \lfloor \log_{t+1} m \rfloor - 1$ curves, and a subcurve $\gamma^* = \gamma'_2 \cup \dots \cup \gamma'_r \subset \gamma_j \cup \gamma_{j+1} \cup \dots \cup \gamma_{j+\lfloor m/(t+1) \rfloor - 1}$ with

the desired properties. Let γ'_1 be the subcurve of γ obtained by extending the endpoint of γ_1 to the endpoint of γ^* along γ so that $\gamma'_1 \supset \gamma_1$. Set $\gamma' = \gamma'_1 \cup \gamma^*$. Hence the collection of curves $C' = \{a_1\} \cup C^*$ and γ' have the desired properties. \square

Lemma 5. *Let $C = \{a_1, \dots, a_m\}$ be a fan grounded by a curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$. If each a_i intersects γ in at most t points, then there is a subfan $C' = \{a_{i_1}, \dots, a_{i_r}\} \subset C$ with $i_1 < \dots < i_r$ and $r = \lfloor \log_{t+1} \log_{t+1} m \rfloor$ that is well-grounded by a subcurve $\gamma' = \gamma'_1 \cup \dots \cup \gamma'_r \subset \gamma$. Moreover, $\gamma'_j \supset \gamma_{i_j}$ for $1 \leq j \leq r$.*

Proof. We apply Lemma 4 to C and $\gamma = \gamma_1 \cup \dots \cup \gamma_m$ to obtain a subcollection $C^* = \{a_{j_1}, a_{j_2}, \dots, a_{j_{m^*}}\}$ of $m^* = \lfloor \log_{t+1} m \rfloor$ curves, and a subcurve $\gamma^* = \gamma_1^* \cup \dots \cup \gamma_{m^*}^* \subset \gamma$ with the properties listed in Lemma 4. Then we apply Lemma 4 again to C^* and γ^* with the elements in C^* in reverse order. By the second property of Lemma 4, the resulting subcollection $C' = \{a_{i_1}, \dots, a_{i_r}\}$ of $r = \lfloor \log_{t+1} \log_{t+1} m \rfloor$ curves is well-grounded by a subcurve $\gamma' = \gamma'_1 \cup \dots \cup \gamma'_r \subset \gamma$, and by the first property we have $\gamma'_j \supset \gamma_{i_j}$ for $1 \leq j \leq r$. \square

We say that fans C_1, \dots, C_l are *simultaneously grounded* (simultaneously well-grounded) by a curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$ to emphasize that they are grounded (well-grounded) by γ with the same partition $\gamma = \gamma_1 \cup \dots \cup \gamma_m$.

Lemma 6. *Let C_1, \dots, C_l be l fans with $C_i = \{a_{i,1}, \dots, a_{i,m}\}$, simultaneously grounded by a curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$. If each $a_{i,j}$ intersects γ in at most t points, then there are indices $j_1 < \dots < j_r$ with $r = \lfloor \log_{t+1}^{(2l)} m \rfloor$ ($2l$ -times iterated logarithm of m) and a subcurve $\gamma' = \gamma'_1 \cup \dots \cup \gamma'_r \subset \gamma$ such that*

- (i) *the subfans $C'_i = \{a_{i,j_1}, \dots, a_{i,j_r}\} \subset C_i$ for $1 \leq i \leq l$ are simultaneously well-grounded by $\gamma'_1 \cup \dots \cup \gamma'_r$,*
- (ii) *$\gamma'_s \supset \gamma_{j_s}$ for $1 \leq s \leq r$.*

Proof. We proceed by induction on l . The base case $l = 1$ follows from Lemma 5. Now assume the statement holds up to $l - 1$. We apply Lemma 5 to the fan $C_1 = \{a_{1,1}, \dots, a_{1,m}\}$ and the curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$, to obtain a subfan $C_1^* = \{a_{1,w_1}, \dots, a_{1,w_s}\} \subset C_1$ with $w_1 < \dots < w_s$ and $s = \lfloor \log_{t+1} \log_{t+1} m \rfloor$ that is well-grounded by a subcurve $\gamma^* = \gamma_1^* \cup \dots \cup \gamma_s^* \subset \gamma$ and satisfies $\gamma_i^* \supset \gamma_{w_i}$ for $1 \leq i \leq s$. For $2 \leq i \leq l$, let $C_i^* = \{a_{i,w_1}, \dots, a_{i,w_s}\} \subset C_i$. Now we apply the induction hypothesis on the collection of $l - 1$ fans C_2^*, \dots, C_l^* that are simultaneously grounded by the curve $\gamma^* = \gamma_1^* \cup \dots \cup \gamma_s^*$. Hence we obtain indices $j_1 < \dots < j_r$ with $r = \lfloor \log_{t+1}^{(2l-2)} s \rfloor = \lfloor \log_{t+1}^{(2l)} m \rfloor$ and a subcurve $\gamma' = \gamma'_1 \cup \dots \cup \gamma'_r \subset \gamma^*$ such that each subfan $C'_i = \{a_{i,j_1}, \dots, a_{i,j_r}\} \subset C_i$ with $2 \leq i \leq l$ is well-grounded by $\gamma'_1 \cup \dots \cup \gamma'_r$, and moreover $\gamma'_z \supset \gamma_z^* \supset \gamma_z$ for $1 \leq z \leq r$. By setting $C'_1 = \{a_{1,j_1}, \dots, a_{1,j_r}\} \subset C_1^*$, the collection C'_1, \dots, C'_l is simultaneously well-grounded by the subcurve $\gamma' = \gamma'_1 \cup \dots \cup \gamma'_r \subset \gamma$. \square

Let $C = \{a_1, \dots, a_m\}$ be a fan with apex v grounded by a curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$ with endpoints p and q . We say that a_i is *left-sided* (*right-sided*) if moving along a_i from v until we reach γ for the first time, and then turning left (right)

onto the curve γ , we reach the endpoint q (p). We say that C_i is *one-sided*, if the curves in C_i are either all left-sided or all right-sided.

Lemma 7. *Let C_1, \dots, C_l be l fans with $C_i = \{a_{i,1}, \dots, a_{i,m}\}$, simultaneously grounded by a curve γ . Then there are indices $j_1 < \dots < j_r$ with $r = \lceil m/2^l \rceil$ such that the subfans $C'_i = \{a_{i,j_1}, \dots, a_{i,j_r}\} \subset C_i$ for $1 \leq i \leq l$ are one-sided.*

Proof. We proceed by induction on l . The base case $l = 1$ is trivial since at least half of the curves in $C_1 = \{a_{1,1}, \dots, a_{1,m}\}$ form a one-sided subset. For the inductive step, assume that the statement holds up to $l - 1$. Let $C_1^* = \{a_{1,w_1}, \dots, a_{1,w_{\lceil m/2 \rceil}}\}$ with $w_1 < \dots < w_{\lceil m/2 \rceil}$ be a subset of $\lceil m/2 \rceil$ curves that is one-sided. For $i \geq 2$, set $C_i^* = \{a_{i,w_1}, \dots, a_{i,w_{\lceil m/2 \rceil}}\}$. Then apply the induction hypothesis on the $l - 1$ fans C_2^*, \dots, C_l^* , to obtain indices $j_1 < \dots < j_r$ with $r = \lceil \lceil m/2 \rceil / 2^{l-1} \rceil = \lceil m/2^l \rceil$ such that the subfans $C'_i = \{a_{i,j_1}, \dots, a_{i,j_r}\} \subset C_i^*$ for $2 \leq i \leq l$ are one-sided. By setting $C'_1 = \{a_{1,j_1}, \dots, a_{1,j_r}\} \subset C_1^*$, the collection C'_1, \dots, C'_l have the desired properties. \square

Since at least half of the fans obtained from Lemma 7 are either left-sided or right-sided, we have the following corollary.

Corollary 1. *Let C_1, \dots, C_{2l} be $2l$ fans with $C_i = \{a_{i,1}, \dots, a_{i,m}\}$, simultaneously grounded by a curve γ . Then there are indices $i_1 < \dots < i_l$ and $j_1 < \dots < j_r$ with $r = \lceil m/2^{2l} \rceil$ such that the subfans $C'_{i_w} = \{a_{i_w,j_1}, \dots, a_{i_w,j_r}\} \subset C_{i_w}$ for $1 \leq w \leq l$ are all left-sided or all right-sided.*

By combining Lemma 6 and Corollary 1, we easily obtain the following lemma which will be used in Section 5.

Lemma 8. *Let C_1, \dots, C_{2l} be $2l$ fans with $C_i = \{a_{i,1}, \dots, a_{i,m}\}$, simultaneously grounded by a curve $\gamma = \gamma_1 \cup \dots \cup \gamma_m$. Then there are indices $i_1 < \dots < i_l$ and $j_1 < \dots < j_r$ with $r = \lceil \log_{i+1}^{(4l)} m \rceil / 2^{2l}$ and a subcurve $\gamma^* = \gamma_{i_1}^* \cup \dots \cup \gamma_r^* \subset \gamma$ such that*

- (i) *the subfans $C'_{i_w} = \{a_{i_w,j_1}, \dots, a_{i_w,j_r}\} \subset C_{i_w}$ for $1 \leq w \leq l$ are simultaneously well-grounded by $\gamma_{i_1}^* \cup \dots \cup \gamma_r^*$,*
- (ii) *$\gamma_s^* \supset \gamma_{j_s}$ for $1 \leq s \leq r$,*
- (iii) *the subfans $C'_{i_1}, \dots, C'_{i_l}$ are all left-sided or all right-sided.*

5 Proof of Theorem 1

By Lemma 2 and the fact that the function $\alpha(n)$ is non-decreasing, it is enough to prove that every k -quasi-planar topological graph on n vertices such that no two edges have more than t points in common and there is an edge that intersects every other edge has at most $2^{\alpha(n)^c} n$ edges, where c depends only on k and t .

Let G be a k -quasi-planar graph on n vertices with no two edges intersecting in more than t points. Let $e_0 = pq$ be an edge that intersects every other edge of G . Let $V_0 = V(G) \setminus \{p, q\}$ and E_0 be the set of edges with both endpoints in

V_0 . Hence we have $|E_0| > |E(G)| - 2n$. Assume without loss of generality that no two elements of E_0 cross e_0 at the same point.

By a well-known fact (see e.g. Theorem 2.2.1 in [3]), there is a bipartition $V_0 = V_1 \cup V_2$ such that at least half of the edges in E_0 connect a vertex in V_1 to a vertex in V_2 . Let E_1 be the set of these edges. For each vertex $v_i \in V_1$, consider the graph G_i whose each vertex corresponds to the subcurve γ of an edge $e \in E_1$ such that

- (i) e is incident to v_i ,
- (ii) the endpoints of $\gamma \subset e$ are v_i and the first intersection point in $e \cap e_0$ as moving from v_i along e .

Two vertices are adjacent in G_i if the corresponding subcurves cross. Each graph G_i is isomorphic to the intersection graph of a collection of curves with one endpoint on a simple closed curve λ_1 and the other endpoint on a simple closed curve λ_2 and with no other points in common with λ_1 or λ_2 . To see this, enlarge the point v_i and the curve e_0 a little, making them simple closed curves λ_1 and λ_2 , and shorten the curves γ appropriately, so as to preserve all crossings between them. Since no k of these curves pairwise intersect, by Lemma 3, G_i contains an independent set of size $\lceil |V(G_i)| / (k - 1)^2 \rceil$. We keep all edges corresponding to the elements of this independent set, and discard all other edges incident to v_i . After repeating this process for all vertices in V_1 , we are left with at least $\lceil |E_1| / (k - 1)^2 \rceil$ edges, forming a set E_2 . We continue this process on the vertices in V_2 and the edges in E_2 . After repeating this process for all vertices in V_2 , we are left with at least $\lceil |E_2| / (k - 1)^2 \rceil$ edges, forming a set E' . Thus $|E(G)| < 2(k - 1)^4 |E'| + 2n$. Now, for any two edges $e_1, e_2 \in E'$ that share an endpoint, the subcurves $\gamma_1 \subset e_1$ and $\gamma_2 \subset e_2$ described above must be disjoint.

For each edge $e \in E'$, fix an arbitrary intersection point $s \in e \cap e_0$ to be the *main intersection point* of e and e_0 . Let $e_1, \dots, e_{|E'|}$ denote the edges in E' listed in the order their main intersection points appear on e_0 from p to q , and let $s_1, \dots, s_{|E'|}$ denote these points respectively. We label the endpoints of each e_i as p_i and q_i , as follows. As we move along e_0 from p to q until we arrive at s_i , then we turn left and move along e_i , we finally reach p_i , while as we turn right at s_i and move along e_i , we finally reach q_i . We define sequences $S_1 = (p_1, \dots, p_{|E'|})$ and $S_2 = (q_1, \dots, q_{|E'|})$. They are sequences of length $|E'|$ over the $(n - 2)$ -element alphabet V_0 .

Lemma 9 (Valtr [19]). *For $2l \geq 1$, at least one of the sequences S_1, S_2 defined above contains a $2l$ -regular subsequence of length at least $\lceil |E'| / (8l) \rceil$.*

The proof of Lemma 9 can be copied almost verbatim from the proof of Lemma 5 in [19]. Indeed, the only fact about the sequences S_1 and S_2 it uses is that the edges $e_{j_1}, e_{j_1+1}, \dots, e_{j_2}$ are spanned by the vertices p_{j_1}, \dots, p_{j_2} and q_{j_1}, \dots, q_{j_2} , for any $j_1 < j_2$.

For the rest of this section, we set $l = 2^{k^2+2k}$ and m to be such that $(\log_{t+1}^{(4l)} m) / 2^{2l} = 3 \cdot 2^k - 4$.

Lemma 10. *Neither of the sequences S_1 and S_2 has a subsequence of type $\text{up}(2l, m)$.*

Proof. By symmetry, it suffices to show that S_1 does not contain a subsequence of type $\text{up}(2l, m)$. We will prove that the existence of such a subsequence would imply that G has k pairwise crossing edges. Let

$$S = (s_{1,1}, \dots, s_{2l,1}, s_{1,2}, \dots, s_{2l,2}, \dots, s_{1,m}, \dots, s_{2l,m})$$

be a subsequence of S_1 of type $\text{up}(2l, m)$ such that the first $2l$ terms are pairwise distinct and $s_{i,1} = \dots = s_{i,m} = v_i$ for $1 \leq i \leq 2l$. For $1 \leq j \leq m$, let $a_{i,j}$ be the subcurve of the edge corresponding to the entry $s_{i,j}$ in S_1 between the vertex v_i and the main intersection point with e_0 . Let $C_i = \{a_{i,1}, \dots, a_{i,m}\}$ for $1 \leq i \leq 2l$. Hence C_1, \dots, C_{2l} are $2l$ fans with apices v_1, \dots, v_{2l} respectively. Clearly, there is a partition $e_0 = \gamma_1 \cup \dots \cup \gamma_m$ such that C_1, \dots, C_{2l} are simultaneously grounded by $\gamma_1 \cup \dots \cup \gamma_m$.

We apply Lemma 8 to the fans C_1, \dots, C_{2l} that are simultaneously grounded by $\gamma_1 \cup \dots \cup \gamma_m$ to obtain indices $i_1 < \dots < i_l$ and $j_1 < \dots < j_r$ with $r = (\log_{l+1}^{(4l)} m) / 2^{2l} = 3 \cdot 2^k - 4$ and a subcurve $\gamma^* = \gamma_1^* \cup \dots \cup \gamma_r^* \subset e_0$ such that

- (i) the subfans $C'_{i_w} = \{a_{i_w, j_1}, \dots, a_{i_w, j_r}\} \subset C_{i_w}$ for $1 \leq w \leq l$ are simultaneously well-grounded by $\gamma_1^* \cup \dots \cup \gamma_r^*$,
- (ii) $\gamma_z^* \supset \gamma_{j_z}$ for $1 \leq z \leq r$,
- (iii) the subfans $C'_{i_1}, \dots, C'_{i_l}$ are all left-sided or all right-sided.

We will only consider the case that $C'_{i_1}, \dots, C'_{i_l}$ are left-sided, the other case being symmetric.

Now for $1 \leq w \leq l$ and $1 \leq z \leq r$, we define the subcurve $a_{w,z}^* \subset a_{i_w, j_z}$ whose endpoints are v_{i_w} and the first point from $a_{i_w, j_z} \cap \gamma^*$ as moving from v_{i_w} along a_{i_w, j_z} . Hence the interior of $a_{w,z}^*$ is disjoint from γ^* . Let $A_w^* = \{a_{w,1}^*, \dots, a_{w,r}^*\}$ for $1 \leq w \leq l$. Note that any two curves in A_w^* do not cross by construction, and all curves in A_w^* enter γ^* from the same side. For simplicity, we will call this the *left side* of γ^* and we will relabel the apices of the fans A_1^*, \dots, A_l^* from v_{i_1}, \dots, v_{i_l} to v_1, \dots, v_l . To finally reach a contradiction, we prove the following.

Claim 1. *For $l = 2^{k^2+2k}$ and $r = 3 \cdot 2^k - 4$, among the l fans A_1^*, \dots, A_l^* with the properties above, there are k pairwise crossing curves.*

The proof follows the argument of Lemma 4.3 in [8]. We proceed by induction on k . The base case $k = 1$ is trivial. For the inductive step, assume the statement holds up to $k - 1$. For simplicity, we let $a_{i,j}^* = a_{i,j'}^*$ for all $j \in \mathbb{Z}$, where $j' \in \{1, \dots, r\}$ is such that $j \equiv j' \pmod{r}$. Consider the fan A_1^* , which is of size r . By construction of A_1^* , the arrangement $A_1^* \cup \{\gamma^*\}$ partitions the plane into r regions. By the pigeonhole principle, there is a subset $V' \subset \{v_1, \dots, v_l\}$ of size

$$|V'| = \frac{l-1}{r} = \frac{2^{k^2+2k}-1}{3 \cdot 2^k - 4},$$

such that all the vertices in V' lie in the same region. Let $j_0 \in \{1, \dots, r\}$ be an integer such that V' lies in the region bounded by a_{1,j_0}^* , a_{1,j_0+1}^* , and γ^* .

Let $v_i \in V'$ and $1 < j_1 < r$, and consider the curve $a_{i,j_0+j_1}^*$. Recall that $a_{i,j_0+j_1}^*$ is disjoint from $\gamma_{j_0}^* \cup \gamma_{j_0+1}^*$ and thus intersects $a_{1,j_0}^* \cup a_{1,j_0+1}^*$. Let $a \subset a_{i,j_0+j_1}^*$

be the maximal subcurve with an endpoint on γ^* whose interior is disjoint from $a_{1,j_0}^* \cup a_{1,j_0+1}^*$. If a intersects a_{1,j_0+1}^* (i.e. the second endpoint of a lies on a_{1,j_0+1}^*), then v_i and the left side of $\gamma_{j_0+2}^* \cup \dots \cup \gamma_{j_0+j_1-1}^*$ lie in different connected components of $\mathbb{R}^2 \setminus (a_{1,j_0+1}^* \cup \gamma^* \cup a)$. Likewise, if a intersects a_{1,j_0}^* , then v_i and the left-side of $\gamma_{j_0+j_1+1}^* \cup \dots \cup \gamma_{j_0+r-1}^*$ lie in different connected components of $\mathbb{R}^2 \setminus (a_{1,j_0}^* \cup \gamma^* \cup a)$.

If a intersects a_{1,j_0+1}^* , then all curves $a_{i,j_0+2}^*, \dots, a_{i,j_0+j_1-1}^*$ must also cross a_{1,j_0+1}^* . Indeed, they connect v_i with the left-side of $\gamma_{j_0+2}^* \cup \dots \cup \gamma_{j_0+j_1-1}^*$, but their interiors are disjoint from γ^* and $a_{i,j_0+j_1}^*$. Likewise, if a intersects a_{1,j_0}^* , then all curves $a_{i,j_0+j_1+1}^*, \dots, a_{i,j_0+r-1}^*$ must also cross a_{1,j_0}^* . Therefore, we have the following.

Claim 2. *For half of the vertices $v_i \in V'$, the curves emanating from v_i satisfy one of the following:*

- (i) $a_{i,j_0+2}^*, a_{i,j_0+3}^*, \dots, a_{i,j_0+r/2}^*$ all cross a_{1,j_0+1}^* ,
- (ii) $a_{i,j_0+r/2+1}^*, a_{i,j_0+r/2+2}^*, \dots, a_{i,j_0+r-1}^*$ all cross a_{1,j_0}^* .

We keep all curves satisfying Claim 2, and discard all other curves. Since $r/2-2 = 3 \cdot 2^{k-1} - 4$ and

$$\frac{|V'|}{2} \geq \frac{l-1}{2r} = \frac{2^{k^2+2k}-1}{6 \cdot 2^k - 8} \geq 2^{(k-1)^2+2(k-1)},$$

by Claim 2, we can apply the induction hypothesis on these remaining curves which all cross a_{1,j_0+1}^* or a_{1,j_0}^* . Hence we have found k pairwise crossing edges, and this completes the proof of Claim 1 and thus Lemma 10. □

We are now ready to prove Theorem 1.

Proof (Theorem 1). By Lemma 9 we know that, say, S_1 contains a $2l$ -regular subsequence of length $\lceil |E'|/(8l) \rceil$. By Theorem 3 and Lemma 10, this subsequence has length at most

$$n \cdot 2l \cdot 2^{(2lm-3)} \cdot (20l)^{10\alpha(n)^{2lm}}.$$

Therefore, we have

$$\left\lceil \frac{|E'|}{8l} \right\rceil \leq n \cdot 2l \cdot 2^{(2lm-3)} \cdot (20l)^{10\alpha(n)^{2lm}},$$

which implies

$$|E'| \leq 8n \cdot 2l^2 \cdot 2^{(2lm-3)} \cdot (20l)^{10\alpha(n)^{2lm}}.$$

Since $l = 2^{k^2+2k}$ and m depends only on k and t , for sufficiently large c (depending only on k and t) and $\alpha(n) \geq 2$, we have

$$|E(G)| < 2(k-1)^4 |E'| + 2n \leq 2^{\alpha(n)^c} n,$$

which completes the proof of Theorem 1. □

6 Proof of Theorem 2

A family of curves in the plane is *simple* if any two of them share at most one point. A family C of curves is K_k -free if the intersection graph of C is K_k -free, that is, no k curves in C pairwise intersect. By $\chi(C)$ we denote the chromatic number of the intersection graph of C , that is, the minimum number of colors that suffice to color the curves in C so that no two intersecting curves receive the same color.

Let ℓ be a horizontal line in the plane. Our proof of Theorem 2 is based on the following result, proved in [10] in a more general setting, for simple K_k -free families of compact arc-connected sets in the plane whose intersections with a line ℓ are non-empty segments.

Theorem 4 (Lasoń, Micek, Pawlik, Walczak [10]). *Every simple K_k -free family of curves C all intersecting ℓ at exactly one point satisfies $\chi(C) \leq a_k$, where a_k depends only on k .*

Special cases of Theorem 4 have been proved by McGuinness [11] for $k = 3$ and by Suk [18] for y -monotone curves and any k . We will also use the following graph-theoretical result.

Lemma 11 (McGuinness [12]). *Let G be a graph, \prec be a total ordering of $V(G)$, and $a, b \geq 0$. For $u, v \in V(G)$, let $G(u, v)$ denote the subgraph of G induced by the vertices strictly between u and v in \prec . If $\chi(G) > 2^{a+b+1}$, then there is an induced subgraph H of G such that $\chi(H) > 2^a$ and $\chi(G(u, v)) \geq 2^b$ for any $uv \in E(H)$.*

Let β be a segment in ℓ . We will consider curves crossing β at exactly one point, always assuming that this intersection point is distinct from the endpoints of β . Any such curve γ is partitioned by β into two subcurves: γ^+ that enters β from above and γ^- that enters β from below, both including the intersection point of β and γ .

Lemma 12. *Let C be a simple K_k -free family of curves all crossing β at exactly one point. If $\gamma_1^+ \cap \gamma_2^+ = \emptyset$ and $\gamma_1^- \cap \gamma_2^- = \emptyset$ for any $\gamma_1, \gamma_2 \in C$, then $\chi(C) \leq 2^{3k-6}$.*

Proof. We proceed by induction on k . The base case $k = 2$ is trivial. For the induction step, assume $k \geq 3$ and the statement holds up to $k-1$. Assume for the sake of contradiction that $\chi(C) > 2^{3k-6}$. Let \prec be the ordering of C according to the left-to-right order of the intersection points with β . Apply Lemma 11 with $a = 0$ and $b = 3k - 7$. It follows that there are two intersecting curves $\delta_1, \delta_2 \in C$ such that $\chi(C(\delta_1, \delta_2)) \geq 2^{3k-7}$, where $C(\delta_1, \delta_2) = \{\gamma \in C : \delta_1 \prec \gamma \prec \delta_2\}$. The curves β, δ_1 and δ_2 together partition the plane into two regions R^+ and R^- so that for $\gamma \in C(\delta_1, \delta_2)$, γ^+ enters β from the side of R^+ , while γ^- enters β from the side of R^- . Take any $\gamma_1, \gamma_2 \in C(\delta_1, \delta_2)$ that intersect at a point p . It follows from the assumptions of the lemma that $p \in \gamma_1^+ \cap \gamma_2^-$ or $p \in \gamma_1^- \cap \gamma_2^+$. If $p \in R^+$, then one of γ_1^-, γ_2^- (whichever contains p) must intersect δ_1 or δ_2 . Similarly,

if $p \in R^-$, then one of γ_1^+, γ_2^+ must intersect δ_1 or δ_2 . In both cases, one of γ_1, γ_2 intersects δ_1 or δ_2 . Let C_1 and C_2 consist of those members of $C(\delta_1, \delta_2)$ that intersect δ_1 and δ_2 , respectively. Clearly, both C_1 and C_2 are K_{k-1} -free, and thus the induction hypothesis yields $\chi(C_1) \leq 2^{3k-9}$ and $\chi(C_2) \leq 2^{3k-9}$. Moreover, $\chi(C(\delta_1, \delta_2) \setminus (C_1 \cup C_2)) \leq 1$ as $C(\delta_1, \delta_2) \setminus (C_1 \cup C_2)$ is independent by the assumption $\gamma_1^+ \cap \gamma_2^+ = \emptyset$ and $\gamma_1^- \cap \gamma_2^- = \emptyset$ for any $\gamma_1, \gamma_2 \in C$. To conclude, $\chi(C(\delta_1, \delta_2)) \leq 2 \cdot 2^{3k-9} + 1 < 2^{3k-7}$, which is a contradiction. \square

Now we prove the following theorem, which can also be generalized to simple K_k -free families of compact arc-connected sets in the plane whose intersections with a segment β are non-empty subsegments.

Theorem 5. *Every simple K_k -free family of curves C all crossing β at exactly one point satisfies $\chi(C) \leq b_k$, where b_k depends only on k .*

Proof. Assume without loss of generality that no curve in C passes through the endpoints of β . One can transform the family $C^+ = \{\gamma^+ : \gamma \in C\}$ into a family $\tilde{C}^+ = \{\tilde{\gamma}^+ : \gamma \in C\}$ so that

- \tilde{C}^+ is simple,
- each $\tilde{\gamma}^+$ is entirely contained in the upper half-plane delimited by ℓ ,
- $\tilde{\gamma}_1^+$ and $\tilde{\gamma}_2^+$ intersect if and only if γ_1^+ and γ_2^+ intersect.

Similarly, one can transform the family $C^- = \{\gamma^- : \gamma \in C\}$ into a family $\tilde{C}^- = \{\tilde{\gamma}^- : \gamma \in C\}$ so that

- \tilde{C}^- is simple,
- each $\tilde{\gamma}^-$ is entirely contained in the lower half-plane delimited by ℓ ,
- $\tilde{\gamma}_1^-$ and $\tilde{\gamma}_2^-$ intersect if and only if γ_1^- and γ_2^- intersect.

The curves $\tilde{\gamma}^+$ and $\tilde{\gamma}^-$ are respectively the upper and lower parts of the curve $\tilde{\gamma} = \tilde{\gamma}^+ \cup \tilde{\gamma}^-$ intersecting ℓ at exactly one point. The family $\tilde{C} = \{\tilde{\gamma} : \gamma \in C\}$ is clearly simple and K_k -free. Therefore, by Theorem 4, $\chi(\tilde{C}) \leq a_k$. Fix a proper a_k -coloring ϕ of \tilde{C} and consider the set C_i consisting of those $\gamma \in C$ for which $\phi(\tilde{\gamma}) = i$. It follows that $\gamma_1^+ \cap \gamma_2^+ = \emptyset$ and $\gamma_1^- \cap \gamma_2^- = \emptyset$ for any $\gamma_1, \gamma_2 \in C_i$. Therefore, by Lemma 12, $\chi(C_i) \leq 2^{3k-6}$. Summing up over all colors used by ϕ we obtain $\chi(C) \leq 2^{3k-6} a_k$. \square

Proof (Theorem 2). By Lemma 2, it is enough to prove that every k -quasi-planar simple topological graph on n vertices that contains an edge intersecting every other edge has at most $c_k n$ edges, where c_k depends only on k .

Let G be a k -quasi-planar simple topological graph on n vertices, and let pq be an edge that intersects every other edge. Remove all edges with an endpoint at p or q except the edge pq . Shorten each curve representing a remaining edge by a tiny bit at both endpoints, so that curves sharing an endpoint become disjoint, while all crossings are preserved. The resulting set of curves C is simple and K_k -free and contains a curve γ crossing every other curve in C . Therefore, $C \setminus \{\gamma\}$ is K_{k-1} -free and $|C \setminus \{\gamma\}| > |E(G)| - 2n$. Since C can be transformed into an equivalent set of curves so that γ becomes the horizontal segment β , Theorem 5

yields $\chi(C \setminus \{\gamma\}) \leq b_{k-1}$. Consequently, $C \setminus \{\gamma\}$ contains an independent set S of size

$$|S| \geq \frac{|C \setminus \{\gamma\}|}{b_{k-1}} > \frac{|E(G)| - 2n}{b_{k-1}}.$$

The edges of G corresponding to the curves in S form a planar subgraph of G , which implies $|S| < 3n$. The two inequalities give $|E(G)| < (3b_{k-1} + 2)n$. \square

References

1. Ackerman, E.: On the maximum number of edges in topological graphs with no four pairwise crossing edges. *Discrete Comput. Geom.* 41, 365–375 (2009)
2. Agarwal, P.K., Aronov, B., Pach, J., Pollack, R., Sharir, M.: Quasi-planar graphs have a linear number of edges. *Combinatorica* 17, 1–9 (1997)
3. Alon, N., Spencer, J.: *The Probabilistic Method*, 3rd edn. Wiley Interscience (2008)
4. Brass, P., Moser, W., Pach, J.: *Research Problems in Discrete Geometry*. Springer (2005)
5. Capovleas, V., Pach, J.: A Turán-type theorem on chords of a convex polygon. *J. Combin. Theory Ser. B* 56, 9–15 (1992)
6. Dilworth, R.P.: A decomposition theorem for partially ordered sets. *Ann. Math.* 51, 161–166 (1950)
7. Fox, J., Pach, J.: Coloring K_k -free intersection graphs of geometric objects in the plane. *European J. Combin.* 33, 853–866 (2012)
8. Fox, J., Pach, J., Suk, A.: The number of edges in k -quasi-planar graphs. *SIAM J. Discrete Math.* 27, 550–561 (2013)
9. Klazar, M.: A general upper bound in extremal theory of sequences, *Comment. Math. Univ. Carolin.* 33, 737–746 (1992)
10. Lasoń, M., Micek, P., Pawlik, A., Walczak, B.: Coloring intersection graphs of arc-connected sets in the plane (manuscript)
11. McGuinness, S.: Colouring arcwise connected sets in the plane I. *Graphs Combin.* 16, 429–439 (2000)
12. McGuinness, S.: On bounding the chromatic number of L-graphs. *Discrete Math.* 154, 179–187 (1996)
13. Nivasch, G.: Improved bounds and new techniques for Davenport-Schinzel sequences and their generalizations. *J. Assoc. Comput. Machin.* 57, 1–44 (2010)
14. Pach, J., Radoičić, R., Tóth, G.: Relaxing planarity for topological graphs. In: Gyóri, E., Katona, G.O.H., Lovász, L., Fleiner, T. (eds.) *More Sets, Graphs and Numbers*, Bolyai Soc. Math. Stud., vol. 15, pp. 285–300. Springer (2006)
15. Pach, J., Shahrokhi, F., Szegedy, M.: Applications of the crossing number. *J. Graph Theory* 22, 239–243 (1996)
16. Pettie, S.: Generalized Davenport-Schinzel sequences and their 0-1 matrix counterparts. *J. Combin. Theory Ser. A* 118, 1863–1895 (2011)
17. Pettie, S.: On the structure and composition of forbidden sequences, with geometric applications. In: *27th ACM Symposium on Computational Geometry*, pp. 370–379. ACM (2011)
18. Suk, A.: Coloring intersection graphs of x -monotone curves in the plane. *Combinatorica* (to appear)
19. Valtr, P.: On geometric graphs with no k pairwise parallel edges. *Discrete Comput. Geom.* 19, 461–469 (1998)

Recognizing Outer 1-Planar Graphs in Linear Time^{*,**}

Christopher Auer, Christian Bachmaier, Franz J. Brandenburg,
Andreas Gleißner, Kathrin Hanauer, Daniel Neuwirth, and Josef Reislhuber

University of Passau, 94030 Passau, Germany
{`auerc,bachmaier,brandenb,gleissner,hanauer,`
`neuwirth,reislhuber`}@fim.uni-passau.de

Abstract. A graph is outer 1-planar (*o1p*) if it can be drawn in the plane such that all vertices are on the outer face and each edge is crossed at most once. *o1p* graphs generalize outerplanar graphs, which can be recognized in linear time and specialize 1-planar graphs, whose recognition is \mathcal{NP} -hard.

Our main result is a linear-time algorithm that first tests whether a graph G is *o1p*, and then computes an embedding. Moreover, the algorithm can augment G to a maximal *o1p* graph. If G is not *o1p*, then it includes one of six minors (see Fig. 3), which are also detected by the recognition algorithm. Hence, the algorithm returns a positive or negative witness for *o1p*.

1 Introduction

Planar graphs are one of the most studied areas in graph theory and an important class in graph drawing. Outerplanar graphs are in turn an important subfamily of planar graphs. Here, all vertices are on the outer face and edges do not cross. Every outerplanar graph has at least two vertices of degree two, which is used for a recognition in linear time [16].

There were several attempts to generalize planarity to graphs that are “almost” planar in some sense. Such attempts are important as many graphs are not planar. One generalization is 1-planar graphs, which were introduced by Ringel [17] in an approach to color a planar graph and its dual. A graph is 1-planar if it can be drawn in the plane such that each edge is crossed at most once and incident edges do not cross. 1-planar graphs are a hot topic in graph drawing, see also [1, 4–6, 8, 9, 13, 15].

The combination of 1-planarity and outerplanarity leads to *o1p* graphs, which are graphs with an embedding in the plane with all vertices on the outer face and at most one crossing per edge. They were introduced by Eggleton [10] who called

* This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG) grant Br835/18-1.

** A linear-time algorithm for testing outer 1-planarity was independently obtained by Hong et al. and appears in these proceedings [14].

them outerplanar graphs with edge crossing number one. He showed that edges of maximal *o1p* graphs do not cross in the outer face and each face is incident to at most one crossing, from which he concluded that every *o1p* graph has an *o1p* drawing with straight-line edges and convex (inner) faces. Thomassen [18] generalized Eggleton's result and characterized the class of 1-planar graphs which admit straight-line drawings by the exclusion of so-called B- and W-configurations in embeddings. These configurations were rediscovered by Hong et al. [13], who also provide a linear-time drawing algorithm that starts from a given embedding.

From the algorithmic perspective there is a big step from zero to some crossings. It is well-known that planar graphs can be recognized in linear time, and there are linear-time algorithms to construct an embedding and drawings, e. g., straight-line drawings and visibility representations in quadratic area. On the contrary, dealing with crossings generally leads to \mathcal{NP} -hard problems. It is \mathcal{NP} -hard to recognize 1-planar graphs [15], even if the graph is given with a rotation system, which determines the cyclic ordering of the edges at each vertex [2]. 1-planarity remains \mathcal{NP} -hard even if the treewidth is bounded [3]. There also is no efficient algorithm to compute the crossing number of a graph [12] and to compute the number of crossings induced by the insertion of an edge into a planar graph [5]. However, there is a linear-time recognition algorithm of Eades et al. [8] for maximal 1-planar graphs, which needs a given rotation system.

In this paper we study *o1p* graphs. Our main result is a linear-time recognition algorithm for *o1p* graphs. This is the first efficient algorithm for a test of 1-planarity that takes solely a graph as input. Our recognition algorithm is based on SPQR-trees. It analyzes its nodes and then either computes an *o1p* embedding or detects one of six minors. If the graph is *o1p*, it can be augmented to a maximal *o1p* graph. In a maximal *o1p* graph, adding a new edge violates its defining property. From the structure of a maximal *o1p* graph we derive that every *o1p* graph is planar. Thus, they are subgraphs of planar graphs with a Hamiltonian cycle, and the SPQR-tree reveals a treewidth of at most three.

2 Preliminaries

We consider simple, undirected graphs $G = (V, E)$ with n vertices and m edges. The graphs are biconnected, otherwise, the components are treated separately. A *drawing* of a graph is a mapping of G into the plane such that the vertices are mapped to distinct points and each edge is a Jordan arc between its endpoints. A drawing is *planar* if the Jordan arcs of the edges do not cross and it is *1-planar* if each edge is crossed at most once. Accordingly, a graph is planar (1-planar) if it has a planar (1-planar) drawing. Crossings of edges with the same endpoint, i. e., *incident* edges, are excluded. A planar drawing of a graph partitions the plane into *faces*. A face is specified by a cyclic sequence of edges that forms its boundary. The set of all faces forms the *embedding* of the graph. In 1-planar drawings, every crossing divides an edge into two *edge segments*. An uncrossed edge consists of one segment. Therefore, a face of a *1-planar embedding* is specified by a cyclic list of edge segments.

A graph G is *outerplanar* if it has a planar drawing with all vertices on one distinguished face. This face is referred to as the *outer face* and corresponds to the unbounded, external face in a drawing on the plane. G is *maximal* outerplanar if no further edge can be added without violating outerplanarity. Then, the edges on the outer face form a Hamiltonian cycle. A graph G is *outer 1-planar*, *o1p* for short, if it has a drawing with all vertices on the outer face and such that each edge is crossed at most once. G is *maximal o1p* if the addition of any edge violates outer 1-planarity.

In an *o1p* embedding, an edge is either *crossing* or *plane* (*non-crossing*). We say that it is *inner*, if none of its segments is part of the boundary of the outer face. Analogously, an edge is *outer*, if it is entirely part of this boundary. Observe that a crossed edge cannot be outer. If the embedding is maximal, we can classify every edge as *outer* or *inner*.

Maximal outerplanar graphs have a unique embedding. This does no longer hold for maximal *o1p* graphs. Consider a graph with 6 vertices and 11 edges consisting of two K_4 s. If the left K_4 is fixed, the right can be flipped. In order to gain more insight into the structure of an *o1p* graph G , we consider its *SPQR-tree* \mathcal{T} . SPQR-trees were first introduced by Di Battista and Tamassia [7] and provide a description of how the graph is composed. Fig. 2(a) depicts an example graph along with its SPQR-tree in Fig. 2(b). In the definition we adopt here, the SPQR-tree is unrooted. The nodes of \mathcal{T} either represent a series composition (S), a parallel composition (P), a single edge (Q), or a triconnected component (R). Associated with each node μ of \mathcal{T} is a graph that is homeomorphic to a subgraph of G and called the *skeleton* $\text{skel}(\mu)$ of μ . In its original definition, every edge $e = \{u, v\}$ of a skeleton, except for one of each Q-node, is a *virtual edge*, i. e., an edge that represents the subgraph of G which connects u and v . This subgraph is also referred to as the *expansion graph* $\text{expg}(e)$ of e . For every virtual edge e in the skeleton of a node μ , there is another node ν that refines the structure of $\text{expg}(e)$. We say that ν is the *refining* node $\text{refn}(e)$ of e . This link is represented by an edge between μ and ν in \mathcal{T} and we say that μ and ν are *adjacent* in \mathcal{T} . Therefore, every leaf of \mathcal{T} is a Q-node. For simplification, we represent edges of the graph directly in the skeleton of an S-, P-, or R-node, so that we can neglect Q-nodes. We also call these edges *non-virtual*. Observe that all nodes are always as large as possible, so neither two S-nodes nor two P-nodes may be adjacent. For a more detailed introduction to SPQR-trees, the reader is referred to [7].

3 Recognition

There are linear-time algorithms for the recognition of (maximal) outerplanar graphs, that use the fact that there are at least two vertices of degree two. A single K_4 implies that this property no longer holds for *o1p* graphs. In contrast, the recognition of 1-planar graphs is \mathcal{NP} -hard [15], even if the graphs are given with a rotation system [2].

Algorithm 1. *o1p* Recognition

```

1: procedure TESTOUTER1PLANARITY( $G$ )
2:   if  $G$  is not planar then return  $\perp$ 
3:    $\mathcal{T} \leftarrow$  SPQR-tree of  $G$ 
4:   for all R- and P-nodes  $\mu \in \mathcal{T}$  do
5:     if  $\mu$  is R-node then
6:       if  $\text{skel}(\mu) \neq K_4$  or contains vertex incident to  $> 2$  virtual edges then
7:         return  $\perp$  ▷ Lemma 1, Corollary 1
8:       for all neighbors  $\nu$  of  $\mu$  do
9:         if  $\nu$  is S-node or R-node then insert plane edge ▷ Proposition 1
10:      else if  $\mu$  is P-node then
11:        if  $\text{skel}(\mu)$  contains  $> 4$  virtual edges then return  $\perp$  ▷ Corollary 1
12:        else if  $\mu$  has only virtual edges then insert plane edge ▷ Lemma 4
13:      compute mapping  $\mathcal{C}$ 
14:       $\mathbb{P}_F \leftarrow$  {fixable P-nodes} ;  $\mathbb{P}_N \leftarrow$  {P-nodes with crossings, but none fixable}
15:      while  $\mathbb{P}_F \cup \mathbb{P}_N \neq \emptyset$  do
16:        while  $\mathbb{P}_F \neq \emptyset$  do
17:          remove next P-node  $\pi$  from  $\mathbb{P}_F$  with fixable S-nodes  $\sigma_1, \sigma_2$ 
18:           $z \leftarrow$  FIXCROSSINGATPNODE( $G, \mathcal{T}, \pi, \sigma_1, \sigma_2$ )
19:          if  $z = \perp$  then return  $\perp$ 
20:          for all  $\pi' \in z$  do update  $\mathcal{C}$ 
21:            if  $\pi'$  is fixable then move  $\pi'$  from  $\mathbb{P}_N$  to  $\mathbb{P}_F$ 
22:          if  $\mathbb{P}_N \neq \emptyset$  then ▷ Lemma 5
23:            choose any element  $\pi$  of  $\mathbb{P}_N$  with S-nodes  $\sigma_1, \sigma_2$  conformant to  $\mathcal{C}$ 
24:             $z \leftarrow$  FIXCROSSINGATPNODE( $G, \mathcal{T}, \pi, \sigma_1, \sigma_2$ )
25:            for all  $\pi' \in z$  do update  $\mathcal{C}$ 
26:              if  $\pi'$  is fixable then move  $\pi'$  from  $\mathbb{P}_N$  to  $\mathbb{P}_F$ 
27:          for all S-/P-/R-nodes  $\mu \in \mathcal{T}$  do fix embedding
28:      return 2-clique-sum of skeleton embeddings

```

Theorem 1. *There is a linear-time algorithm to test whether a biconnected graph G is *o1p* and, if so, returns an embedding.*

We prove this theorem by first establishing a number of necessary conditions for a graph to have an *o1p* embedding. At the same time, we implement a linear-time algorithm (Algorithm 1) that checks these conditions and, if positive, constructs an *o1p* embedding of the input graph. The algorithm starts by ensuring that the input graph is planar (cf. Corollary 4) and computes its SPQR-tree. Both subroutines take $\mathcal{O}(n)$ time [11]. Observe that, although the graph will be augmented during the next steps, it remains *o1p* and therefore also planar. Consequently, the number of nodes in \mathcal{T} always is in $\mathcal{O}(n)$. In a second step, we show that the conditions are not only necessary, but also sufficient.

We start with two observations regarding *o1p* embeddings. For maximal 1-planar embeddings, a well-known fact is that every crossing induces a K_4 . This holds in an even tightened form for *o1p* embeddings:

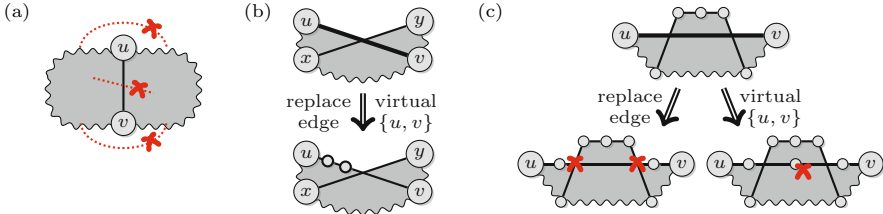


Fig. 1. (a): Proposition 2, (b) and (c): Proposition 3

Proposition 1 ([6]). *Let $\{a, b\}$ and $\{c, d\}$ be a pair of crossing edges in an $o1p$ embedding of a maximal $o1p$ graph. Then the vertices $a, b, c,$ and d form a K_4 and the edges $\{a, b\}, \{b, c\}, \{c, d\},$ and $\{d, a\}$ are plane.*

Consider a plane, inner edge $\{u, v\}$ in an $o1p$ embedding of a graph G . Then, $\{u, v\}$ “partitions” the embedding and the deletion of u and v disconnects G (cf. Fig. 1(a)).

Proposition 2. *Every plane, inner edge in an $o1p$ embedding connects a separation pair.*

Let \mathcal{T} be the SPQR-tree of an $o1p$ graph G .

Lemma 1. *The skeleton of every R-node is a K_4 .*

Proof. Recall that outerplanar graphs are series-parallel. Hence, the SPQR-tree of an outerplanar graph has no R-nodes. Let μ be an R-node in \mathcal{T} . Then, $\text{skel}(\mu)$ must be embedded such that at least two edges cross, e.g., edges $\{a, b\}$ and $\{c, d\}$. By Proposition 1, the vertices $a, b, c,$ and d form a K_4 .

There must be an embedding of $\text{skel}(\mu)$ such that all vertices are on the boundary of the same face. Suppose $\text{skel}(\mu)$ has more than four vertices. Then, at least one of $\{a, b\}, \{b, c\}, \{c, d\},$ and $\{d, a\}$ is an inner edge. By Proposition 1, all of them are plane. As an inner edge cannot be virtual, by Proposition 2, $\text{skel}(\mu)$ has a separation pair, so $\text{skel}(\mu)$ is not triconnected, a contradiction. \square

Instead of considering the possible embeddings of G on the whole, we study those of the skeletons of the nodes in \mathcal{T} . As G is $o1p$, there must be an embedding of every skeleton of \mathcal{T} such that the 2-clique-sums over all skeletons result in an $o1p$ embedding of G . In short, a 2-clique-sum combines two graphs by selecting an edge (2-clique) in each one and glueing them together at those edges. The selected edges are removed from the new graph. If the input graphs were embedded, the embedding is inherited for the 2-clique-sum such that in each case the other graph takes the position of the removed edge.

Consequently, we need to derive properties of $o1p$ embeddings of skeletons. Like in usual $o1p$ embeddings, there must be a face (the outer face) such that all vertices lie on its boundary. However, as virtual edges represent entire subgraphs, they demand special attention. Observe that the expansion graph of every virtual edge contains, besides the separation pair, at least one more vertex. Consider

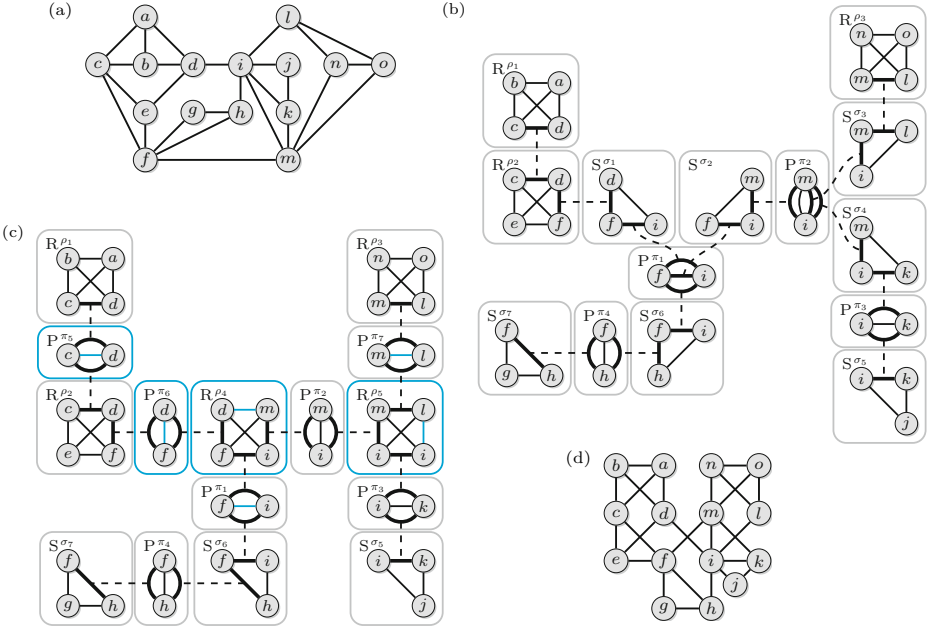


Fig. 2. Input graph (a), its SPQR-tree (b), the SPQR-tree after the algorithm (c) (new edges and nodes colored), and the found *o1p* embedding (d).

the virtual edge $\{u, v\}$ in Fig. 1(b). The crossing edge $\{x, y\}$ partitions $\{u, v\}$ into two segments, hence, $\text{expg}(\{u, v\})$ must be embedded such that it replaces the edge segment of $\{u, v\}$ that lies on the outer face. Suppose a virtual edge e is embedded such that it has at least two crossings. Then there is either an edge in $\text{expg}(e)$ that is crossed more than once or at least one vertex is enclosed between two crossings and hence does not lie on the outer face (cf. Fig. 1(c)).

Proposition 3. *Every virtual edge may consist of at most two edge segments and the embedding must be such that at least one segment is part of the boundary of the outer face.*

Observe that in contrast to the *o1p* embedding of the whole graph, we must allow the crossing of two virtual edges with a common end vertex in the embedding of a skeleton. We qualify the virtual edges that must always be embedded plane.

Lemma 2. *Let μ be a node of \mathcal{T} and let $e = \{u, v\}$ be a virtual edge in $\text{skel}(\mu)$. If both u and v have degree > 1 in $\text{expg}(e)$, then e must be embedded plane.*

Proof. Suppose e is embedded such that it crosses another edge e' , which can be virtual or not. In either case, e' may be crossed at most once. As $\text{skel}(\mu)$ is biconnected and $\text{expg}(e)$ contains at least one additional vertex, in $\text{expg}(e)$, either all edges incident to u or all edges incident to v must be crossed in order

to have all vertices lie on the outer face. If both u and v have degree > 1 in $\text{expg}(e)$, e' has at least two crossings. \square

Note that unlike planar embeddings, neither the skeleton of an S-node nor that of an R-node has a unique *o1p* embedding. However, Lemma 2 limits the number of possible *o1p* embeddings for skeletons considerably:

Lemma 3. *Let μ be a node in \mathcal{T} . Then for every virtual edge $e = \{u, v\}$ in $\text{skel}(\mu)$ holds:*

If $\text{refn}(e)$ is a P- or an R-node, then e must be embedded plane in $\text{skel}(\mu)$.

If $\text{refn}(e)$ is an S-node whose skeleton is the cycle graph $(u, c_1, c_2, \dots, c_k, v, u)$, then e must be embedded such that the segment incident to u (v) lies on the outer face if the edge $\{u, c_1\}$ ($\{c_k, v\}$) is virtual.

Proof. If $\text{refn}(e)$ is a P- or an R-node, both u and v have degree > 1 in $\text{expg}(e)$, hence by Lemma 2, e must be embedded plane. Suppose $\text{refn}(e)$ is an S-node whose skeleton is the cycle graph $(u, c_1, c_2, \dots, c_k, v, u)$. As the embedding must be such that all vertices lie on the outer face, only the edges $\{u, c_1\}$ or $\{c_k, v\}$ may be crossed. Recall that by the structure of an SPQR-tree, if $\{u, c_1\}$ ($\{c_k, v\}$) is virtual, then $\text{refn}(\{u, c_1\})$ ($\text{refn}(\{c_k, v\})$) is either a P- or an R-node, so $\{u, c_1\}$ ($\{c_k, v\}$) must be embedded plane. \square

Lemma 1, Proposition 3, and Lemma 3 allow us to draw the following conclusion:

Corollary 1. *Every virtual edge in an S-node must be embedded plane.*

The skeleton of every R-node contains at most four virtual edges, which must be embedded plane, and no vertex may be incident to more than two virtual edges.

The skeleton of a P-node may have at most 4 virtual edges.

The conditions for R-nodes are easily checked by Algorithm 1 in time $\mathcal{O}(1)$ per R-node. Additionally, if an R-node is adjacent to another R-node or an S-node, then one of the edges of the K_4 is not present. For an example, see the R-nodes ρ_1 and ρ_2 in Fig. 2(b). By Proposition 1, however, the edge may be inserted and is always plane. Observe that this introduces a new P-node π_5 in Fig. 2(c). As an R-node may have at most four neighbors and as the SPQR-tree can be updated in $\mathcal{O}(1)$ time, this modification takes constant time, too.

The following lemma allows us to insert a non-virtual edge in every P-node, if none is present. In Fig. 2(b), this would apply, e. g., to π_1 .

Lemma 4. *Let u, v be the vertices in the skeleton of a P-node without non-virtual edges. Then, the insertion of $\{u, v\}$ does not violate outer 1-planarity and $\{u, v\}$ is plane for every *o1p* embedding of G .*

Proof. Let π be a P-node with separation pair u, v , that is connected by virtual edges only. According to the definition of SPQR-trees, every skeleton of a P-node has at least three edges. Hence, π is adjacent to at least three other nodes. Subsequently, at least two virtual edges must be refined by S-nodes and are embedded with a crossing. This results in a crossing of two non-virtual edges in G that are, by Lemma 3, incident to u and v , respectively. By Proposition 1, the edge $\{u, v\}$ can always be inserted and is plane. \square

Again, Algorithm 1 can check these two conditions and augment the graph for a P-node in time $\mathcal{O}(1)$, which results in a running time of $\mathcal{O}(n)$ for ll. 4 – 12.

Consider a P-node π with vertices u, v . If $\text{skel}(\pi)$ has at most two virtual edges, they can be embedded without a crossing and such that both completely lie on the outer face. Suppose $\text{skel}(\pi)$ has at least three virtual edges. In consequence of Proposition 3, two of them must cross each other. In Fig. 2(b), this holds for π_1 and π_2 . We say that a P-node π *claims* a non-virtual edge e , and express this by defining the mapping $\mathcal{C}(e) = \pi$, if e is crossed in every embedding of $\text{skel}(\pi)$ that conforms with Lemma 3. Observe that \mathcal{C} is uniquely defined, since G is *o1p* and thus, no edge may be crossed more than once. We say that an embedding of the skeleton of a P-node is *admissible* if it conforms with Lemma 3 and does not imply the crossing of non-virtual edges claimed by other P-nodes. In Fig. 2(b), e. g., π_1 has two admissible embeddings, but both imply crossing the edge $\{f, m\}$, either by $\{d, i\}$ or by $\{h, i\}$. Hence, π_1 claims $\{f, m\}$. Computing \mathcal{C} involves checking the embeddings of all P-nodes. As every P-node has at most four virtual edges, there are at most $\binom{4}{2} \cdot 2 = 12$ embeddings. Hence, the total time needed for this step is in $\mathcal{O}(n)$.

If every admissible embedding of $\text{skel}(\pi)$ yields the same set of edges that are crossed, then π is called *fixable*. Let e, e' be two virtual edges that are embedded crossing each other. Observe that in this case, two S-nodes, namely $\text{refn}(e)$ and $\text{refn}(e')$, are “crossing”. By Proposition 1, the crossing can be augmented to a K_4 . The insertion of these additional edges transforms the crossing S-nodes into an R-node that represents the K_4 . In Fig. 2(b), this happens to π_1, σ_1 , and σ_2 . If the skeleton of an S-node previously had exactly three vertices, it is now completely contained in the K_4 . Otherwise, its number of vertices is reduced by exactly 1, i. e., the vertex u or v , respectively. Note that completing the K_4 may affect the number of admissible embeddings and hence the fixability of other P-nodes if there was an admissible embedding of their skeletons that implied crossing one of e or e' . Algorithm 2 checks whether the virtual edges may cross each other and fixes the embedding of π . The next lemma enables us to also proceed when there is no fixable P-node.

Lemma 5. *Let π be a non-fixable P-node. If \mathcal{T} has no fixable P-nodes, then every admissible embedding of $\text{skel}(\pi)$ maintains at least one admissible embedding for every other P-node.*

Proof. Consider the fixing procedure of an embedding for a P-node π and S-nodes σ' and σ'' . Let e' and e'' be the non-virtual edges that are crossed thereby. This affects the number of admissible embeddings for the skeletons of at most two other P-nodes π' and π'' , that are adjacent to σ' and σ'' , respectively. Observe that $\pi' \neq \pi''$, as \mathcal{T} is a tree, and that every non-virtual edge is represented in the skeleton of exactly one node of \mathcal{T} .

Consider π' . W. l. o. g., let e' be the non-virtual edge in $\text{skel}(\sigma')$ that is crossed after the fixing. Then, the number of admissible embeddings of $\text{skel}(\pi')$ is reduced by exactly those that implied crossing e' , too. However, π' did not claim e' , so there is at least one other admissible embedding of $\text{skel}(\pi')$. The same argument holds for π'' and e'' . \square

Algorithm 2. Fix Embedding of P-node with two crossing S-nodes

```

1: procedure FIXCROSSINGATPNODE( $G, \mathcal{T}$ , P-Node  $\pi$ , S-Node  $\sigma_1$ , S-Node  $\sigma_2$ )
2:   Let  $u, v$  be the separation pair of  $\pi$ ,
3:   Let  $(u, c_1, \dots, c_k, v, u)$  be the cycle in  $\text{skel}(\sigma_1)$ .
4:   Let  $(u, d_1, \dots, d_l, v, u)$  be the cycle in  $\text{skel}(\sigma_2)$ .
5:   if  $\{c_k, v\}$  virtual or  $\{u, d_1\}$  virtual then
6:     if  $\{u, c_1\}$  virtual or  $\{d_l, v\}$  virtual then return  $\perp$ 
7:     else swap roles of  $\sigma_1, \sigma_2$ 
8:    $\mathbb{P}_d \leftarrow \emptyset$  ▷ possibly affected P-nodes
9:   if  $k > 1$  then insert edge  $\{u, c_k\}$  in  $G$ , update  $\mathcal{T}$ 
10:    if  $\{c_{k-1}, c_k\}$  virtual then add its associated P-node to  $\mathbb{P}_d$ 
11:    else if  $\{u, c_k\}$  virtual then add its associated P-node to  $\mathbb{P}_d$ 
12:    if  $l > 1$  then insert edge  $\{v, d_1\}$  in  $G$ , update  $\mathcal{T}$ 
13:    if  $\{d_1, d_2\}$  virtual then add its associated P-node to  $\mathbb{P}_d$ 
14:    else if  $\{v, d_1\}$  virtual then add its associated P-node to  $\mathbb{P}_d$ 
15:    insert edge  $\{c_k, d_1\}$ , update  $\mathcal{T}$ 
16:    if  $\pi$  has two (other) virtual edges then add  $\pi$  to  $\mathbb{P}_d$ 
17:  return  $\mathbb{P}_d$ 

```

Hence, by applying Lemma 5, we can step by step fix all embeddings of the skeletons of P-nodes with at least three virtual edges. Afterwards, every P-node has exactly two virtual edges and one non-virtual (cf. Fig. 2(c)). In Algorithm 1, this corresponds to ll. 15 – 26. FIXCROSSINGATPNODE takes $\mathcal{O}(1)$ time per call and there are embeddings of at most $\mathcal{O}(n)$ P-nodes to fix. Hence, the time for this part is $\mathcal{O}(n)$. The algorithm concludes by selecting an admissible embedding for all P- and R-nodes. All remaining S-nodes are embedded as plane cycles. The embedding for G is obtained via the 2-clique-sums of all skeleton embeddings (cf. Fig. 2(d)). Consequently, Algorithm 1 has a running time of $\mathcal{O}(n)$.

It remains to show that all conditions presented so far are also sufficient for a graph to be *o1p*. Every skeleton is, taken by itself, embedded *o1p*. Consider the 2-clique-sum of two skeleton embeddings. This operation glues both graphs together at two virtual edges. After the augmentation of Algorithm 1, every virtual edge is embedded such that it lies on the outer face. Hence, in the resulting embedding, every vertex still lies on the outer face and every edge is crossed at most once. With this, the outer 1-planarity of the whole embedding follows by structural induction.

Lemma 6. *A graph G is o1p if and only if it is a subgraph of a graph H with SPQR-tree \mathcal{T} such that R-nodes and S-nodes are adjacent to P-nodes only, every skeleton of an R-node is a K_4 , and every skeleton of a P-node has exactly one non-virtual and two virtual edges.*

This concludes the proof of Theorem 1. Additionally, if a graph is *o1p*, Algorithm 1 also provides an *o1p* embedding. With some extra effort, we can augment G to maximality. Consider the supergraph H constructed from G by Algorithm 1 and its SPQR-tree. It may have S-nodes with four or more vertices. As all re-

maining S-nodes are embedded plane, we can insert a plane edge between two non-adjacent vertices, which splits the S-node into two smaller S-nodes with an intermediate P-node. This procedure can be repeated until all S-nodes are triangles. Next, consider a P-node that is adjacent to exactly two S-nodes, e. g., π_4 in Fig. 2(c). Then, we can insert a crossing edge ($\{g, i\}$ in the example) that augments the subgraph to a K_4 . As a result, the nodes π_4 , σ_6 , and σ_7 are replaced by a new R-node. We denote by H^+ this supergraph of H . Its SPQR-tree consists of R-nodes, of which each corresponds to a K_4 and S-nodes, of which each corresponds to a triangle. R- and S-nodes are only connected via P-nodes, which in turn have exactly two virtual edges and one non-virtual. Consider an embedding of H^+ . It has a tree-like structure that consists of K_4 s and triangles (K_3 s) that share an edge if and only if their corresponding R- and S-nodes are connected via a P-node. As no P-node is adjacent to two S-nodes, triangles can only share an edge with K_4 s. Suppose H^+ was not maximal. If we were able to insert an inner, plane edge, this would correspond to inserting a P-node into the SPQR-tree of H^+ . However, no two P-nodes may be adjacent. Inserting an inner, crossed edge is equal to augmenting two triangles to a K_4 , which is impossible, too, as no P-node is adjacent to two S-nodes. Finally, consider adding an edge to the outer face. As every crossing has been augmented to a K_4 , the boundary of the outer face consists of a plane Hamiltonian cycle. Hence, every additional edge would shield at least one vertex from the outer face. Consequently, we can easily extend Algorithm 1 such that it maximizes the input graph. Additionally, we receive another characterization:

Lemma 7. *A graph G is maximal $o1p$ if and only if the conditions for H in Lemma 6 hold and in its SPQR-tree, no P-node is adjacent to more than one S-node and the skeleton of every S-node is a cycle of length three.*

The argument above also implies that every embedded maximal $o1p$ graph is maximal for all $o1p$ embeddings.

Corollary 2. *A graph G is maximal $o1p$ if it has a maximal $o1p$ embedding.*

Note that the embedding of a maximal $o1p$ graph is fixed if and only if that of the skeleton of every R-node is. This, in turn, is the case iff it contains at least two incident virtual edges.

Corollary 3. *The embedding of a maximal $o1p$ graph is unique up to inversion if and only if the skeleton of every R-node of its SPQR-tree contains a vertex that is incident to exactly two virtual edges.*

Another consequence of Lemma 7 is, that every maximal $o1p$ graph is composed of triangles and K_4 s. Changing the embedding of the K_4 s, we obtain:

Corollary 4. *Every $o1p$ graph is planar and has treewidth at most three.*

Observe that if the step that augments a P-node with two adjacent triangle S-nodes to a K_4 is omitted, we obtain a plane-maximal $o1p$ graph, i. e., every additional edge either violates outer 1-planarity or introduces a new crossing. Equivalently, we can also adjust Algorithm 1 to test (plane) $o1p$ maximality.

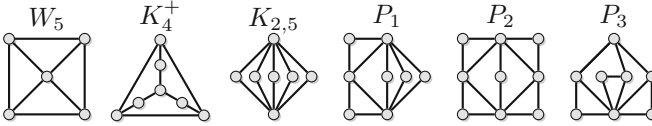


Fig. 3. Set M of minors of non- $o1p$ graphs

Corollary 5. *There is a linear-time algorithm to test whether a graph is maximal (plane-maximal) $o1p$ or to augment an $o1p$ graph to a maximal (plane-maximal) $o1p$ graph.*

From the recognition algorithm, we can immediately derive minors of non- $o1p$ graphs: If the algorithm returns \perp , the graph at hand contains at least one of the $o1p$ minors M as depicted in Fig. 3.

Theorem 2. *If a graph is not $o1p$, it contains at least one graph in M as a minor. Further, M is minimal and every graph in M is not $o1p$ while removing or contracting an edge makes it $o1p$.*

Note that a graph might still be $o1p$ even if it contains a graph in M as a minor, as outer 1-planar graphs are not closed under taking minors. The first minor W_5 is the wheel with five vertices, which is the smallest triconnected graph that is not $o1p$ (Lemma 1). W_5 occurs in ll. 2 and 6 of Algorithm 1. If \perp is returned in l. 2, then the graph contains a K_5 or $K_{3,3}$ as minor and W_5 is a minor of both. In l. 6, the first of the two checks implies W_5 : If the R-node contains more than four vertices, \perp is returned and the whole graph contains a planar triconnected component with at least four vertices, which always contains a W_5 as a minor. If the second condition in l. 6 is true, then the R-node at hand is a K_4 that contains a vertex v incident to three virtual edges. As the expansion graph of a virtual edge has a path with two edges as minor, we obtain K_4^+ in Fig. 3, where vertex v is in the center. If a P-node has at least five virtual edges (l. 11), then the $K_{2,5}$ is the minor. The remaining minors P_1 , P_2 , and P_3 can occur when fixing the embedding of a P-node with two crossing S-nodes. Consider l. 6 in Algorithm 2. If $\{u, d_1\}$ and $\{u, c_1\}$ are virtual, then u is incident to virtual edges in both S-nodes σ_1 and σ_2 . If u is incident to at least one other virtual edge in π in whose expansion graph, u has at least degree two, then π has no admissible embedding and we obtain P_3 as minor. By a complete case differentiation, P_1 and P_2 can also be identified as minors.

4 Conclusion

We have designed a linear-time recognition algorithm for $o1p$ that uses the SPQR-tree and returns a witness in terms of an $o1p$ embedding or detects one of six minors, respectively.

Are there other classes of 1-planar graphs which can be recognized efficiently?

References

1. Alam, M.J., Brandenburg, F.J., Kobourov, S.G.: Straight-line drawings of 3-connected 1-planar graphs. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 83–94. Springer, Heidelberg (2013)
2. Auer, C., Brandenburg, F.J., Gleißner, A., Reislhuber, J.: On 1-planar graphs with rotation systems. Tech. Rep. MIP 1207, University of Passau (2012)
3. Bannister, M.J., Cabello, S., Eppstein, D.: Parameterized complexity of 1-planarity. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 97–108. Springer, Heidelberg (2013)
4. Brandenburg, F.J., Eppstein, D., Gleißner, A., Goodrich, M.T., Hanauer, K., Reislhuber, J.: On the density of maximal 1-planar graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 327–338. Springer, Heidelberg (2013)
5. Cabello, S., Mohar, B.: Adding one edge to planar graphs makes crossing number and 1-planarity hard. Tech. Rep. arXiv:1203.5944 (cs.CG), Computing Research Repository (CoRR) (March 2012)
6. Dehkordi, H.R., Eades, P.: Every outer-1-plane graph has a right angle crossing drawing. *Internat. J. Comput. Geom. Appl.* 22(6), 543–558 (2012)
7. Di Battista, G., Tamassia, R.: On-line planarity testing. *SIAM J. Comput.* 25(5), 956–997 (1996)
8. Eades, P., Hong, S.H., Katoh, N., Liotta, G., Schweitzer, P., Suzuki, Y.: Testing maximal 1-planarity of graphs with a rotation system in linear time. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 339–345. Springer, Heidelberg (2013)
9. Eades, P., Liotta, G.: Right angle crossing graphs and 1-planarity. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 148–153. Springer, Heidelberg (2011)
10. Eggleton, R.B.: Rectilinear drawings of graphs. *Utilitas Math.* 29, 149–172 (1986)
11. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
12. Hliněný, P.: Crossing number is hard for cubic graphs. *J. Combin. Theory, Ser. B* 96(4), 455–471 (2006)
13. Hong, S.-H., Eades, P., Liotta, G., Poon, S.-H.: Fáry’s theorem for 1-planar graphs. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 335–346. Springer, Heidelberg (2012)
14. Hong, S.H., Eades, P., Naoki, K., Liotta, G., Schweitzer, P., Suzuki, Y.: A linear-time algorithm for testing outer-1-planarity. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 71–82. Springer, Heidelberg (2013)
15. Korzhik, V.P., Mohar, B.: Minimal obstructions for 1-immersion and hardness of 1-planarity testing. *J. Graph Theor.* 72, 30–71 (2013)
16. Mitchell, S.L.: Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Inform. Process. Lett.* 9(5), 229–232 (1979)
17. Ringel, G.: Ein Sechsfarbenproblem auf der Kugel. *Abh. aus dem Math. Seminar der Univ. Hamburg* 29, 107–117 (1965)
18. Thomassen, C.: Rectilinear drawings of graphs. *J. Graph Theor.* 12(3), 335–341 (1988)

Straight Line Triangle Representations

Nieke Aerts and Stefan Felsner

Technische Universität Berlin,
Institut für Mathematik, Berlin, Germany
{aerts,felsner}@math.tu-berlin.de

Abstract. A straight line triangle representation (SLTR) of a planar graph is a straight line drawing such that all the faces including the outer face have triangular shape. Such a drawing can be viewed as a tiling of a triangle using triangles with the input graph as skeletal structure. In this paper we present a characterization of graphs that have an SLTR that is based on flat angle assignments, i.e., selections of angles of the graph that have size π in the representation. We also provide a second characterization in terms of contact systems of pseudosegments. With the aid of discrete harmonic functions we show that contact systems of pseudosegments that respect certain conditions are stretchable. The stretching procedure is then used to get straight line triangle representations. Since the discrete harmonic function approach is quite flexible it allows further applications, we mention some of them.

The drawback of the characterization of SLTRs is that we are not able to effectively check whether a given graph admits a flat angle assignment that fulfills the conditions. Hence it is still open to decide whether the recognition of graphs that admit straight line triangle representation is polynomially tractable.

1 Introduction

In this paper we study a representation of planar graphs in the classical setting, i.e., vertices are represented by points in the Euclidean plane and edges by non-crossing continuous curves connecting the points. We aim at classifying the class of planar graphs that admit a straight line representation in which all faces are triangles. Haas et al. present a necessary and sufficient condition for a graph to be a pseudo-triangulation [8], however this condition is not sufficient for a graph to have a straight line triangle representation (e.g. see Fig. 2 and [1]). There have been investigations of the problem in the dual setting, i.e., in the setting of side contact representations of planar graphs with triangles. Gansner, Hu and Kobourov show that outerplanar graphs, grid graphs and hexagonal grid graphs are Touching Triangle Graphs (TTGs). They give a linear time algorithm to find the TTG [7]. Alam, Fowler and Kobourov [2] consider proper TTGs, i.e., the union of all triangles of the TTG is a triangle and there are no holes. They give a necessary and a stronger sufficient condition for biconnected outerplanar graphs to be TTG, a characterization, however, is missing. Kobourov, Mondal and Nishat present construction algorithms for proper TTGs of 3-connected

cubic graphs and some grid graphs. They also present a decision algorithm for testing whether a 3-connected planar graph is proper TTG [10].

Here is the formal introduction of the main character for this paper.

Definition 1. A plane drawing of a graph such that

- all the edges are straight line segments and
 - all the faces, including the outer face, bound a non-degenerate triangle
- is called a Straight Line Triangle Representation (SLTR).

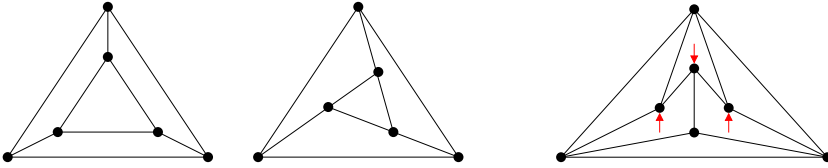


Fig. 1. A graph and one of its SLTRs **Fig. 2.** A Flat Angle Assignment (given by the arrows) that is not an SLTR

Clearly every straight line drawing of a triangulation is an SLTR. So the class of planar graphs admitting an SLTR is rich. On the other hand, graphs admitting an SLTR cannot have a cut vertex. Indeed, as shown below (Prop. 1), graphs admitting an SLTR are well connected. Being well connected, however, is not sufficient as shown e.g. by the cube graph.

To simplify the discussion we assume that the input graph is given with a plane embedding and a selection of three vertices of the outer face that are designated as corner vertices for the outer face. These three vertices are called *suspension vertices*. If needed, an algorithm may try all triples of vertices as suspensions.

If a degree two vertex has an angle of size π in one of its incident faces, then it also has an angle of size π in the face on the other side. Hence, this vertex and its two incident edges can be replaced by a single edge connecting the two neighbors of the vertex. Such an operation is called a *vertex reduction*. The only angles of an SLTR whose size exceeds π are the outer angles at the outer triangle. Therefore, we can use vertex reductions to eliminate all the degree two vertices, except for degree two vertices that are suspensions.

A plane graph G with suspensions s_1, s_2, s_3 is said to be *internally 3-connected* when the addition of a new vertex v_∞ in the outer face, that is made adjacent to the three suspension vertices, yields a 3-connected graph.

Proposition 1. *If a graph G admits an SLTR with s_1, s_2, s_3 as corners of the outer triangle and no vertex reduction is possible, then G is internally 3-connected.*

Proof. Consider an SLTR of G . Suppose there is a separating set U of size 2. It is enough to show that each component of $G \setminus U$ contains a suspension vertex, so that $G + v_\infty$ is not disconnected by U . Since G admits no vertex reduction every degree two vertex is a suspension. Hence, if C is a component and $C \cup U$ induces a path, then there is a suspension in C . Otherwise consider the convex

hull of $C \cup U$ in the SLTR. The convex corners of this hull are vertices that expose an angle of size at least π . Two of these large angles may be at vertices of U but there is at least one additional large angle. This large angle must be the outer angle at a vertex that is an outer corner of the SLTR, i.e., a suspension. \square

From Prop. 1 it follows that any graph that is not internally 3-connected but does admit an SLTR, is a subdivision of an internally 3-connected graph. Therefore we may assume that the graphs we consider are internally 3-connected.

In Section 2 we present necessary conditions for the existence of an SLTR in terms of what we call a flat angle assignment. A flat angle assignment that fulfills the conditions is shown to induce a partition of the set of edges into a set of pseudosegments. Finally, with the aid of discrete harmonic functions we show that in our case the set of pseudosegments is stretchable. Hence, the necessary conditions are also sufficient. The drawback of the characterization is that we are not aware of an effective way of checking whether a given graph admits a flat angle assignment that fulfills the conditions.

In Section 3 we consider further applications of the stretching approach. First we look at flat angle assignments that yield faces with more than three corners. Then we proceed to prove a more general result about stretchable systems of pseudosegments with our technique. The result is not new, de Fraysseix and Ossona de Mendez have investigated stretchability conditions for systems of pseudosegments in [3,4,5]. The counterpart to Theorem 2 can be found in [5, Theorem 38]. The proof there is based on a long and complicated inductive construction.

2 Necessary and Sufficient Conditions

Consider a plane, internally 3-connected graph $G = (V, E)$ with suspensions given. Suppose that G admits an SLTR. This representation induces a set of *flat angles*, i.e., incident pairs (v, f) such that vertex v has an angle of size π in the face f .

Since G is internally 3-connected every vertex has at most one flat angle. Therefore, the flat angles can be viewed as a partial mapping of vertices to faces. Since the outer angle of suspension vertices exceeds π , suspensions have no flat angle. Since each face f (including the outer face) is a triangle, each face has precisely three angles that are not flat. In other words every face f has $|f| - 3$ incident vertices that are assigned to f . This motivates the definition:

Definition 2. A flat angle assignment (FAA) is a mapping from a subset U of the non-suspension vertices to faces such that

- [C_v] Every vertex of U is assigned to at most one face,
- [C_f] For every face f , precisely $|f| - 3$ vertices are assigned to f .

Not every FAA induces an SLTR. An example is given in Fig. 2. Hence, we have to identify another condition. To state this we need a definition. Let H be a connected subgraph of the plane graph G . The *outline cycle* $\gamma(H)$ of H is

the closed walk corresponding to the outer face of H . An *outline cycle* of G is a closed walk that can be obtained as outer cycle of some connected subgraph of G . Outline cycles may have repeated edges and vertices, see Fig. 3. The interior $\text{int}(\gamma)$ of an outline cycle $\gamma = \gamma(H)$ consists of H together with all vertices, edges and faces of G that are contained in the area enclosed by γ .

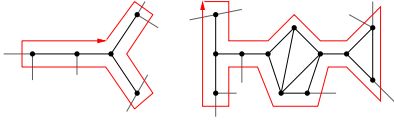


Fig. 3. Examples of outline cycles

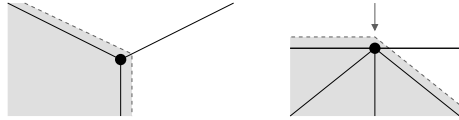


Fig. 4. Combinatorially Convex Corners

Proposition 2. *An SLTR obeys the following condition C_o :*

$[C_o]$ *Every outline cycle that is not the outline cycle of a path, has at least three geometrically convex corners.*

Proofs of Propositions 2 and 3 have been moved to the appendix.

Condition C_o has the disadvantage that it depends on a given SLTR, hence, it is useless for deciding whether a planar graph G admits an SLTR. The following definition allows to replace C_o by a combinatorial condition on an FAA.

Definition 3. *Given an FAA ψ . A vertex v of an outline cycle γ is a combinatorial convex corner for γ with respect to ψ if*

- v is a suspension vertex, or
- v is not assigned and there is an edge e incident to v with $e \notin \text{int}(\gamma)$, or
- v is assigned to a face f , $f \notin \text{int}(\gamma)$ and there exists an edge e incident to v with $e \notin \text{int}(\gamma)$.

In Fig. 4 an unassigned and an assigned combinatorially convex corner are shown. The grey area represents the interior of some outline cycle and the arrow represents the assignment of the vertex to the face in which the arrow is drawn.

Proposition 3. *Let G admit an SLTR Γ , that induces the FAA ψ and let H be a connected subgraph of G . If v is a geometrically convex corner of the outline cycle $\gamma(H)$ in Γ , then v is a combinatorially convex corner of $\gamma(H)$ with respect to ψ .*

The proposition enables us to replace the condition on geometrically convex corners w.r.t. an SLTR by a condition on combinatorially convex corners w.r.t. an FAA.

$[C_o^*]$ *Every outline cycle that is not the outline cycle of a path, has at least three combinatorially convex corners.*

From Prop. 2 and Prop. 3 it follows that this condition is necessary for an FAA that induces an SLTR. In Thm. 1 we prove that if an FAA obeys C_o^* then it induces an SLTR. The proof is constructive. In anticipation of this result we say that an FAA obeying C_o^* is a *good flat angle assignment* and abbreviate it as a *GFAA*.

Next we show that a GFAA induces a contact family of pseudosegments. This family of pseudosegments is later shown to be stretchable, i.e., it is shown to be homeomorphic to a contact system of straight line segments.

Definition 4. A contact family of pseudosegments is a family $\{c_i\}_i$ of simple curves $c_i : [0, 1] \rightarrow \mathbb{R}^2$, with different endpoints, i.e., $c_i(0) \neq c_i(1)$, such that any two curves c_j and c_k ($j \neq k$) have at most one point in common. If so, then this point is an endpoint of (at least) one of them.

A GFAA ψ on a graph G gives rise to a relation ρ on the edges: Two edges, both incident to v and f are in relation ρ if and only if v is assigned to f . The transitive closure of ρ is an equivalence relation on the edges of G .

Proposition 4. The equivalence classes of edges of G defined by ρ form a contact family of pseudosegments.

Proof. Let the equivalence classes of ρ be called arcs.

Condition C_v ensures that every vertex is interior to at most one arc. Hence, the arcs are simple curves and no two arcs cross.

Every arc has two distinct endpoints, otherwise it would be a cycle and its outline cycle has only one combinatorially convex corner. If an arc touched itself, the outline cycle of this equivalence class would have at most one combinatorially convex corner. This again contradicts C_o^* .

If two arcs share two points, the outline cycle has at most two combinatorially convex corners. This again contradicts C_o^* .

We conclude that the family of arcs satisfies the properties of a contact family of pseudosegments. □

Definition 5. Let Σ be a family of pseudosegments and let S be a subset of Σ . A point p of a pseudosegment from S is a free point for S if

1. p is an endpoint of a pseudosegment in S , and
2. p is not interior to a pseudosegment in S , and
3. p is incident to the unbounded region of S , and
4. p is incident to the unbounded region of Σ or p is incident to a pseudosegment that is not in S .

With Lem. 1 we prove that the family of pseudosegments Σ that arises from a GFAA has the following property

[C_P] Every subset S of Σ with $|S| \geq 2$ has at least three free points.

Lemma 1. Let ψ a GFAA on a plane, internally 3-connected graph G . For every subset S of the family of pseudosegments associated with ψ , it holds that, if $|S| \geq 2$ then S has at least 3 free points.

Proof. Let S be a subset of the contact family of pseudosegments defined by the GFAA (Prop. 4).

Each pseudosegment of S corresponds to a path in G . Let H be the subgraph of G obtained as union of the paths of pseudosegments in S . We assume that H

is connected and leave the discussion of the cases where it is not to the reader. If H itself is not a path, then by C_o^* the outline cycle $\gamma(H)$ must have at least three combinatorially convex corners. Every combinatorially convex corner of $\gamma(H)$ is a free point of S .

If S induces a path, then the two endpoints of this path are free points for S . Moreover, there exists at least one vertex v in this path which is an endpoint for two pseudosegments and not an interior point for any. Now there must be an edge e incident to v , such that $e \notin S$, therefore v is a free point for S . \square

Given an internally 3-connected, plane graph G with a GFAA. To find a corresponding SLTR we aim at representing each of the pseudosegments induced by the FAA as a straight line segment. If this can be done, every assigned vertex will be between its two neighbors that are part of the same pseudosegment. This property can be modeled by requiring that the coordinates $p_v = (x_v, y_v)$ of the vertices of G satisfy a harmonic equation at each assigned vertex.

Indeed if uv and vw are edges belonging to a pseudosegment s , then the coordinates satisfy

$$x_v = \lambda_v x_u + (1 - \lambda_v)x_w \quad \text{and} \quad y_v = \lambda_v y_u + (1 - \lambda_v)y_w \quad (1)$$

where the parameter λ_v can be chosen arbitrarily from $(0, 1)$. These are the harmonic equations for v .

In the SLTR every unassigned vertex v is placed in a weighted barycenter of its neighbors. In terms of coordinates this can be written as

$$x_v = \sum_{u \in N(v)} \lambda_{vu} x_u, \quad y_v = \sum_{u \in N(v)} \lambda_{vu} y_u. \quad (2)$$

These are the harmonic equations for an unassigned vertex v . The λ_{vu} can be chosen arbitrarily in the range set by the convexity conditions: $\sum_{u \in N(v)} \lambda_{vu} = 1$ and $\lambda_{vu} > 0$.

Vertices whose coordinates are not restricted by harmonic equations are called *poles*. In our case the suspension vertices are the three poles of the harmonic functions for the x and y -coordinates. The coordinates for the suspension vertices are fixed as the corners of some non-degenerate triangle, this adds six equations to the linear system.

The theory of harmonic functions and applications to (plane) graphs are nicely explained by Lovász [11]. The following proposition is taken from Chapter 3 of [11].

Proposition 5. *For every choice of the parameters λ_v and λ_{vu} complying with the conditions, the system has a unique solution.*

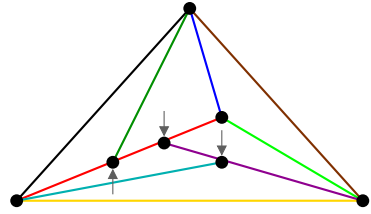


Fig. 5. A stretched representation of a contact family of pseudosegments that arises from a GFAA in the graph of Fig 2

Now we state our main result, it shows that the necessary conditions are also sufficient.

Theorem 1. *Let G be an internally 3-connected, plane graph and Σ a family of pseudosegments associated to an FAA, such that each subset $S \subseteq \Sigma$ has three free points or cardinality at most one. The unique solution of the system of equations that arises from Σ is an SLTR.*

Proof. The proof consists of 7 arguments, which together yield that the drawing induced from the GFAA is a non-degenerate, plane drawing. The proof has been inspired by the proof of Colin de Verdière [6] for convex straight line drawings of plane graphs via spring embeddings.

1. *Pseudosegments become Segments.* Let $(v_1, v_2), (v_2, v_3), \dots, (v_{k-1}, v_k)$ be the set of edges of a pseudosegment defined by ψ . The harmonic conditions for the coordinates force that v_i is placed between v_{i-1} and v_{i+1} for $i = 2, \dots, k-1$. Hence all the vertices of the pseudosegment are placed on the segment with endpoints v_1 and v_k .

2. *Convex Outer Face.* The outer face is bounded by three pseudosegments and the suspensions are the endpoints for these three pseudosegments. The coordinates of the suspensions (the poles of the harmonic functions) have been chosen as corners of a non-degenerate triangle and the pseudosegments are straight line segments, therefore the outer face is a triangle and in particular convex.

3. *No Concave Angles.* Every vertex, not a pole, is forced either to be on the line segment between two of its neighbors (if assigned) or in a weighted barycenter of all its neighbors (otherwise). Therefore every non-pole vertex is in the convex hull of its neighbors. This implies that there are no concave angles at non-poles.

4. *No Degenerate Vertex.* A vertex is degenerate if it is placed on a line, together with at least three of its neighbors. Suppose there exists a vertex v , such that v and at least three of its neighbors are placed on a line ℓ . Let S be the connected component of pseudosegments that are aligned with ℓ , such that S contains v . The set S contains at least two pseudosegments. Therefore S must have at least three free points, v_1, v_2, v_3 .

By property 4 in the definition of free points, each of the free points is incident to a segment that is not aligned with ℓ . Suppose the free points are not suspension vertices. If v_i is interior to $s_i \in S$, then s_i has an endpoint on each side of ℓ . If v_i is not assigned by the GFAA it is in the strict convex hull of its neighbors, hence, v_i is an endpoint of a segment reaching into each of the two half-planes defined by ℓ .

Now suppose v_1 and v_2 are suspension vertices¹ and consider the third free point, v_3 . If v_3 is interior to a pseudosegment not on ℓ , then one endpoint of this pseudosegment lies outside the convex hull of the three suspensions, which is a contradiction. Hence it is not interior to any pseudosegment and at least one of its neighbors does not lie on ℓ , but then v_3 should be in a weighted barycenter of its neighbors, hence again we would find a vertex outside the convex hull of

¹ Not all three suspension vertices lie on one line, hence at least one of the three free points is not a suspension.

the suspension vertices. Therefore at most one of the free points is a suspension and ℓ is incident to at most one of the suspension vertices.

In any of the above cases each of v_1, v_2, v_3 has a neighbor on either side of ℓ .

Let n^+ and $n^- = -n^+$ be two normals for line ℓ and let p^+ and p^- be the two poles, that maximize the inner product with n^+ resp. n^- . Starting from the neighbors of the v_i in the positive halfplane of ℓ we can always move to a neighbor with larger² inner product with n^+ until we reach p^+ . Hence v_1, v_2, v_3 have paths to p^+ in the upper halfplane of ℓ and paths to p^- in the lower halfplane. Since v_1, v_2, v_3 also have a path to v we can contract all vertices of the upper and lower halfplane of ℓ to p^+ resp. p^- and all inner vertices of these paths to v to produce a $K_{3,3}$ minor of G . This is in contradiction to the planarity of G . Therefore, there is no degenerate vertex.

5. *Preservation of Rotation System.* Let $\theta(v) = \sum_f \theta(v, f)$ denote the sum of the angles around an interior vertex. Here f is a face incident to v and $\theta(v, f)$ is the (smaller!) angle between the two edges incident to v and f in the drawing obtained by solving the harmonic system. If the incident faces are oriented consistently around v , then the angles sum up to 2π , otherwise $\theta(v) > 2\pi$ (see Fig. 6). We do not consider the outer face in the sums so that the b vertices incident to the outer face contribute a total angle of $(b - 2)\pi$ to the inner faces.

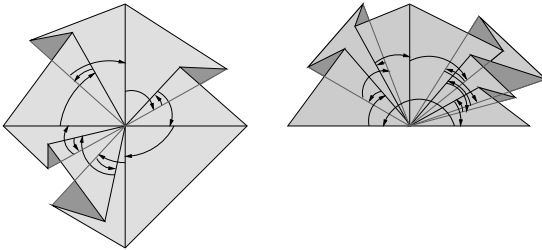


Fig. 6. Vertices with their surrounding faces not oriented consistently

Now consider the sum $\theta(f) = \sum_v \theta(v, f)$ of the angles of a face f . At each vertex incident to f the contribution $\theta(v, f)$ is at most of size π . A closed polygonal chain with k corners, selfintersecting or not, has a sum of inner angles equal to $(k - 2)\pi$. Therefore $\theta(f) \leq (|f| - 2)\pi$. The sum over all vertices $\sum_v \theta(v)$ and the sum over all faces $\sum_f \theta(f)$ must be

equal since they count the same angles in two different ways.

$$(|V| - b)2\pi + (b - 2)\pi \leq \sum_v \theta(v) = \sum_f \theta(f) \leq ((2|E| - b) - 2(|F| - 1))\pi \quad (3)$$

This yields $|V| - |E| + |F| \leq 2$. Since G is planar Euler's formula implies equality. Therefore $\theta(v) = 2\pi$ for every interior vertex v and the faces must be oriented consistently around every vertex, i.e. the rotation system is preserved. Note that the rotation system could have been flipped, between clockwise and counter-clockwise but then it is flipped at every vertex.

6. *No Crossings.* Suppose two edges cross. On either side of both of the edges there is a face, therefore there must be a point p in the plane which is covered by at least two faces. Outside of the drawing there is only the unbounded face.

² If n^+ is perpendicular to another segment this may not be possible. In this case we can use a slightly perturbed vector n^+_ϵ to break ties.

Move along a ray, that does not pass through a vertex of the graph, from p to infinity. A change of the cover number, i.e. the number of faces by which the point is covered, can only occur when crossing an edge. But if the cover number changes then the rotation system at a vertex of that edge must be wrong. This would contradict the previous item. Therefore a crossing cannot exist.

7. *No Degeneracy.* Suppose there is an edge of length zero. Since every vertex has a path to each of the three suspensions there has to be a vertex a that is incident to an edge of length zero and an edge ab of non-zero length. Following the direction of forces we can even find such a vertex-edge pair with b contributing to the harmonic equation for the coordinates of a . We now distinguish two cases.

If a is assigned, it is on the segment between b and some b' , together with the neighbor of the zero length edge this makes three neighbors of a on a line. Hence, a is a degenerate vertex. A contradiction.

If a is unassigned it is in the convex hull of its neighbors. However, starting from a and using only zero-length edges we eventually reach some vertex a' that is incident to an edge $a'b'$ of non-zero length, such that b' is contributing to the harmonic equation for the coordinates of a' . Vertex a' has the same position as a and is also in the convex hull of its neighbors. This makes a crossing of edges unavoidable. A contradiction. Hence, there are no edges of length zero.

Suppose there is an angle of size zero. Since every vertex is in the convex hull of its neighbors there are no angles of size larger than π . Moreover there are no crossings, hence the face with the angle of size zero is stretching along a line segment with two angles of size zero. Since there are no edges of length zero and all vertices are in the convex hull of their neighbors, all but two vertices of the face must be assigned to this face. Therefore, there are two pseudosegments bounding this face, which have at least two points in common, this contradicts that Σ is a family of pseudosegments. We conclude that there is no degeneracy.

From items 1-7 we conclude that the drawing is plane and thus an SLTR. □

3 Further Applications of the Proof Technique

We have shown that a graph G has an SLTR exactly if it admits an FAA satisfying C_v , C_f and C_o^* . Conditions C_v and C_o^* are necessary for the proof that the system of pseudosegments corresponding to the FAA is stretchable. Condition C_f , however, is only needed to make all the faces triangles. Modifying condition C_f allows for further applications of the stretching technique. Of course we still need at least three corners for every face. Also we have to make sure that all the non-suspension vertices of the outer face are assigned to the outer face. Together this makes the modified face condition:

$[C_f^*]$ For every face f , at most $|f| - 3$ vertices are assigned to f and all non-suspension vertices of the outer face f^o are assigned to f^o .

If we use the empty flat angle assignment, i.e., if the harmonic equations of all non-suspensions are of type (2), then we obtain a drawing such that all non-suspension vertices are in the barycenter of their neighbors. This is the Tutte

drawing [12] with asymmetric elastic forces given by the parameters λ_{uv} , see also [11]. Note that in this case the existence of at least three combinatorially convex corners at an outline cycle (condition C_o^*) follows from the internally 3-connectedness of the graph.

The construction of Section 2 also applies when

- the assignment has $|f| - i$ vertices assigned to every inner face f , for $i = 4, 5$ (drawing with only convex 4-gon or only convex 5-gon faces.)
- the assignment has some number c_f of corners at inner face f (drawing with convex faces of prescribed complexity).

The drawback is that again in these cases we do not know how to find an FAA that fulfills C_o^* .

In [9] Kenyon and Sheffield study T -graphs in the context of dimer configurations (weighted perfect matchings). In our terminology T -graphs correspond to straight line representations such that each non-suspension is assigned. In [9] the straight line representations of T -graphs are obtained by analyzing random walks. Cf. [11] for further connections between discrete harmonic functions and Markov chains.

Stretchability of Systems of Pseudosegments. A contact system of pseudosegments is *stretchable* if it is homeomorphic to a contact system of straight line segments. De Fraysseix and Ossona de Mendez characterized stretchable systems of pseudosegments [3,4,5]. They use the notion of an extremal point.

Definition 6. *Let Σ be a family of pseudosegments and let S be a subset of Σ . A point p is an extremal point for S if*

1. p is an endpoint of a pseudosegment in S , and
2. p is not interior to a pseudosegment in S , and
3. p is incident to the unbounded region of S .

Theorem 2 (De Fraysseix & Ossona de Mendez [5, Theorem 38]).

A contact family Σ of pseudosegments is stretchable if and only if each subset $S \subseteq \Sigma$ of pseudosegments with $|S| \geq 2$, has at least 3 extremal points.

Our notion of a free point (Def. 5) is more restrictive than the notion of an extremal point. In the following we show that there is no big difference. First in Prop. 6 we show that in the case of families of pseudosegments that live on a plane graph via an FAA, the two notions coincide. Then we continue by reproving Thm. 2 as a corollary of Thm. 1.

Proposition 6. *Let G be an internally 3-connected, plane graph and Σ a family of pseudosegments associated to an FAA, such that each subset $S \subseteq \Sigma$ has three extremal points or cardinality at most one. The unique solution of the system of equations corresponding to Σ , is an SLTR.*

Proof. Note that in the proof of Thm. 1 the notion of free points is only used to show that there is no degenerate vertex. We show how to modify this part of the argument for the case of extremal points:

Consider again the set S of pseudosegments aligned with ℓ . We will show that all extremal points are also free points. Let p an extremal point of S . Assuming that p is not free we can negate item 4. from Def. 5, i.e., all the pseudosegments for which p is an endpoint are in S . By 3-connectivity p is incident to at least three pseudosegments, all of which lie on the line ℓ . Since all regions are bounded by three pseudosegments and p is not interior to a segment of S , all the regions incident to p must lie on ℓ . But then p is not incident to the unbounded region of S , hence p is not an extremal point. Therefore all extremal points of S are also free points of S . Prop. 6 now follows from Thm. 1. \square

Proof (of Thm. 2). Let Σ a contact family of pseudosegments which is stretchable. Consider a set $S \subseteq \Sigma$ of cardinality at least two in the stretching, i.e., in the segment representation. Endpoints (of segments) on the boundary of the convex hull of S are extremal points. There are at least three of them unless S lies on a line ℓ . In the latter case, there is a point q on ℓ that is the endpoint of two colinear segments. This is a third extremal point.

Conversely, assume that each subset $S \subseteq \Sigma$ of pseudosegments, with $|S| \geq 2$, has at least 3 extremal points. We aim at applying Prop 6. To this end we construct an extended system Σ^+ of pseudosegments in which every region is bounded by precisely three pseudosegments.

First we take a set Δ of three pseudosegments that intersect like the three sides of a triangle so that Σ is in the interior. The corners of Δ are chosen as suspensions and the sides of Δ are deformed such that they contain all extremal points of the family Σ . Let the new family be Σ' .

Next we add new *protection points*, these points ensure that the pseudosegments of Σ' will be mapped to straight lines. For each inner region R in Σ' , for each pseudosegment s in R , we add a protection point for each visible side of s . The protection point is connected to the endpoints of s , with respect to R from the visible side of s .

Now the inner part of R is bounded by an alternating sequence of endpoints of Σ' and protection points. We connect two protection points if they share a neighbor in this sequence. Last we add a *triangulation point* in R and connect it to all protection points of R .

This construction yields a family Σ^+ of pseudosegments such that every region is bounded by precisely three pseudosegments and every subset $S \subseteq \Sigma^+$ has at least 3 extremal points, unless it has cardinality one.

Let V be the set of points of Σ^+ and E the set of edges induced by Σ^+ . It follows from the construction that $G = (V, E)$ is internally 3-connected.

By Prop. 6 the graph $G = (V, E)$ together with Σ^+ is stretchable to an SLTR. Removing the protection points, triangulation points and their incident edges yields a contact system of straight line segments homeomorphic to Σ . \square

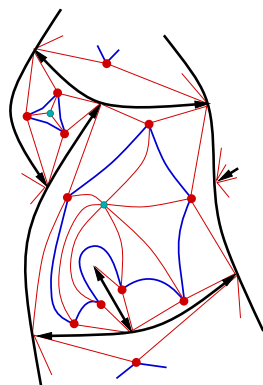


Fig. 7. Protection points in red and the triangulation point in cyan for two faces of some Σ'

4 Conclusion and Open Problems

We have given necessary and sufficient conditions for a 3-connected planar graph to have an SLT Representation. Given an FAA and a set of rational parameters $\{\lambda_i\}_i$, the solution of the harmonic system can be computed in polynomial time. Checking whether a solution is degenerate can also be done in polynomial time. Hence, we can decide in polynomial time whether a given FAA corresponds to an SLTR. In other words, checking whether a given FAA is a GFAA can be done in polynomial time. However, most graphs admit different FAAs of which only some are good. We are not aware of an effective way of finding a GFAA. Therefore we have to leave this problem open: Is the recognition of graphs that have an SLTR (GFAA) in P ?

Given a 3-connected planar graph and a GFAA, interesting optimization problems arise, e.g. find the set of parameters $\{\lambda_i\}_i$ such that the smallest angle in the graph is maximized, or the set of parameters such that the length of the shortest edge is maximized.

References

1. Aerts, N., Felsner, S.: Henneberg steps for Triangle Representations, <http://page.math.tu-berlin.de/~aerts/pubs/ptsltr.pdf>
2. Alam, M.J., Fowler, J., Kobourov, S.G.: Outerplanar graphs with proper touching triangle representations (unpublished manuscript)
3. de Fraysseix, H., de Mendez, P.O.: Stretching of jordan arc contact systems. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 71–85. Springer, Heidelberg (2004)
4. de Fraysseix, H., Ossona de Mendez, P.: Contact and intersection representations. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 217–227. Springer, Heidelberg (2005)
5. de Fraysseix, H., de Mendez, P.O.: Barycentric systems and stretchability. *Discrete Applied Mathematics* 155, 1079–1095 (2007)
6. de Verdière, Y.C.: Comment rendre géodésique une triangulation d’une surface? *L’Enseignement Mathématique* 37, 201–212 (1991)
7. Gansner, E.R., Hu, Y., Kobourov, S.G.: On Touching Triangle Graphs. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 250–261. Springer, Heidelberg (2011)
8. Haas, R., Orden, D., Rote, G., Santos, F., Servatius, B., Servatius, H., Souvaine, D.L., Streinu, I., Whiteley, W.: Planar minimally rigid graphs and pseudo-triangulations. *Comput. Geom.* 31, 31–61 (2005)
9. Kenyon, R., Sheffield, S.: Dimers, tilings and trees. *J. Comb. Theory, Ser. B* 92, 295–317 (2004)
10. Kobourov, S.G., Mondal, D., Nishat, R.I.: Touching triangle representations for 3-connected planar graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 199–210. Springer, Heidelberg (2013)
11. Lovász, L.: Geometric representations of graphs, Draft version (December 11, 2009), <http://www.cs.elte.hu/~lovasz/geomrep.pdf>
12. Tutte, W.T.: How to draw a graph. *Proc. of the London Math. Society* 13, 743–767 (1963)

Extending Partial Representations of Circle Graphs^{*}

Steven Chaplick¹, Radoslav Fulek¹, and Pavel Klavík^{2,**}

¹ Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
chaplick@kam.mff.cuni.cz, radoslav.fulek@gmail.com

² Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
klavik@iuuk.mff.cuni.cz

Abstract. *The partial representation extension problem* is a recently introduced generalization of the recognition problem. A *circle graph* is an intersection graph of chords of a circle. We study the partial representation extension problem for circle graphs, where the input consists of a graph G and a partial representation \mathcal{R}' giving some pre-drawn chords that represent an induced subgraph of G . The question is whether one can extend \mathcal{R}' to a representation \mathcal{R} of the entire G , i.e., whether one can draw the remaining chords into a partially pre-drawn representation.

Our main result is a polynomial-time algorithm for partial representation extension of circle graphs. To show this, we describe the structure of all representation a circle graph based on split decomposition. This can be of an independent interest.

1 Introduction

Graph drawings and visualizations are important topics of graph theory and computer science. A frequently studied type of representations are so-called *intersection representations*. An intersection representation of a graph represents its vertices by some objects and encodes its edges by intersections of these objects, i.e., two vertices are adjacent if and only if the corresponding objects intersect. Classes of intersection graphs are obtained by restricting these objects; e.g., *interval graphs* are intersection graphs of intervals of the real line, *string graphs* are intersection graphs of strings in plane, and so on. These representations are well-studied; see e.g. [25].

For a fixed class \mathcal{C} of intersection-defined graphs, a very natural computational problem is *recognition*. It asks whether an input graph G belongs to \mathcal{C} . In this paper, we study a recently introduced generalization of this problem called *partial representation extension* [19]. Its input gives with G a part of the representation and the problem asks whether this partial representation can be extended to a representation of the entire G ; see Fig. 1 for an illustration. We show that this problem can be solved in polynomial time for the class of *circle graphs*.

^{*} Supported by ESF Eurogiga project GraDR as GAČR GIG/11/E023.

^{**} Supported by Charles University as GAUK 196213.

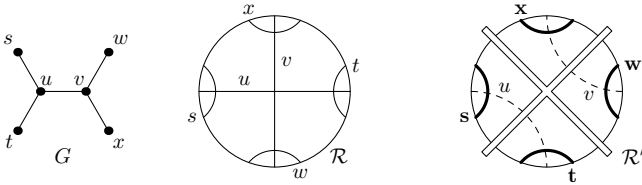


Fig. 1. On the left, a circle graph G with a representation \mathcal{R} is given. A partial representation \mathcal{R}' given on the right with the pre-drawn chords s , t , w , and x is not extendible. The chords are depicted as arcs to make the figure more readable.

Circle Graphs. Circle graphs are intersection graphs of chords of a circle. They were first considered by Even and Itai [12] in the early 1970s in study of stack sorting techniques. Other motivations are due to their relations to Gauss words [11] (see Fig. 2) and matroid representations [10,5]. Circle graphs are also important regarding rank-width [22].

Let $\chi(G)$ denote chromatic number of G , and let $\omega(G)$ denote the clique-number of G . Trivially we have $\omega(G) \leq \chi(G)$ and the graphs for which every induced subgraph satisfies equality are the well-known *perfect graphs* [6]. In general, the difference between these two numbers can be arbitrarily high, e.g, there is a triangle-free graph with arbitrary high chromatic number. Circle graphs are known to be *almost perfect* which means that $\chi(G) \leq f(\omega(G))$ for some function f . The best known result for circle graphs [20] states that $f(k)$ is $\Omega(k \log k)$ and $\mathcal{O}(2^k)$.

The complexity of recognition of circle graphs was a long standing open problem; see [25] for an overview. The first results, e.g., [12], gave existential characterizations which did not give polynomial-time algorithms. The mystery whether circle graphs can be recognized in polynomial time frustrated mathematicians for some years. It was resolved in the mid-1980s and several polynomial-time algorithms were discovered [4,13,21] (in time $\mathcal{O}(n^7)$ and similar). Later, a more efficient algorithm [24] based on *split decomposition* was given, and the current state-of-the-art recognition algorithm [14] runs in a quasi-linear time in the number of vertices and the number of edges of the graph.

The Partial Representation Extension Problem. It is quite surprising that this very natural generalization of the recognition problem was considered only recently. It is currently an active area of research which is inspiring a deeper investigation of many classical graph classes. For instance, a recent result of Angelini et al. [1] states that the problem is decidable in linear time for planar

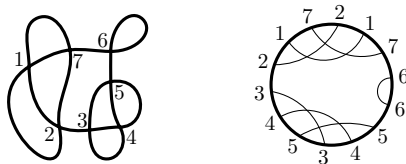


Fig. 2. A self-intersecting closed curve with n intersections numbered $1, \dots, n$ corresponds to a representation of circle graph with the vertices $1, \dots, n$ where the endpoints of the chords are placed according to the order of the intersections along the curve.

graphs. On the other hand, Fáry's Theorem claims that every planar graph has a straight-line embedding, but extension of such an embedding is NP-hard [23].

In the context of intersection-defined classes, this problem was first considered in [19] for interval graphs. Currently, the best known results are linear-time algorithms for interval graphs [3] and proper interval graphs [17], a quadratic-time algorithm for unit interval graphs [17], and polynomial-time algorithms for permutation and function graphs [16]. For chordal graphs (as subtree-in-a-tree graphs) several versions of the problems were considered [18] and all of them are NP-complete.

The Structure of Representations. To solve the recognition problem for G , one just needs to build a single representation. However, to solve the partial representation extension problem, the structure of all representations of G must be well understood. A general approach used in the above papers is the following. We first derive necessary and sufficient constraints from the partial representation \mathcal{R}' . Then we efficiently test whether some representation \mathcal{R} satisfies these constraints. If none satisfies them, then \mathcal{R}' is not extendible. And if some \mathcal{R} satisfies them, then it extends \mathcal{R}' .

It is well-known that the split decomposition [8, Theorem 3] captures the structure of all representations of circle graphs. The standard recognition algorithms produce a special type of representations using split decomposition as follows. We find a *split* in G , construct two smaller graphs, build their representation recursively, and then join these two representations to produce \mathcal{R} . In Section 3, we give a simple recursive descriptions of all possible representations based on splits. Our result can be interpreted as “describing a structure like PQ-trees for circle graphs.” It is possible that the proof techniques from other papers on circle graphs such as [7,14] would give a similar description. However, these techniques are more involved than our approach which turned out to be quite elementary and simple.

Restricted Representations. The partial representation extension problem belongs to a larger group of problems dealing with *restricted representations of graphs*. These problems ask whether there is some representation of an input graph G satisfying some additional constraints. We describe two examples of these problems.

An input of the *simultaneous representations problem*, shortly SIM, consists of graphs G_1, \dots, G_k with some vertices common for all the graphs. The problem asks whether there exist representations $\mathcal{R}_1, \dots, \mathcal{R}_k$ representing the common vertices the same. This problem is polynomially solvable for permutation and comparability graphs [15]. They additionally show that for chordal graphs it is NP-complete when k is part of the input and polynomially solvable for $k = 2$. For interval graphs, a linear-time algorithm is known for $k = 2$ [3] and the complexity is open in general. For some classes, these problems are closely related to the partial representation extension problems. For example, there is an FPT algorithm for interval graphs with the number of common vertices as the parameter [19], and partial representations of interval graphs can be extended in linear time by reducing it to a corresponding simultaneous representations problem [3].

The *bounded representation problem* [17] prescribes bounds for each vertex of the input graph and asks whether there is some representation satisfying these bounds. For circle graphs, the input specifies a pair of arcs (A_v, A'_v) of the circle for each chord v and a solution is required to have one endpoint of v in A_v and the other one in A'_v . This problem is clearly a generalization of partial representation extension since one can describe a partial representation using singleton arcs. It is known to be polynomially solvable for interval and proper interval graphs [2], and surprisingly it is NP-complete for unit interval graphs [17]. The complexity for other classes of graphs is not known.

Our Results. We study the following problem (see Section 2 for definitions):

Problem: Partial Representation Extension – REPEXT(CIRCLE)
Input: A circle graph G and a partial representation \mathcal{R}' .
Output: Is there a representation \mathcal{R} of G extending \mathcal{R}' ?

In Section 3, we describe a simple structure of all representations. This is used in Section 4 to obtain our main algorithmic result:

Theorem 1. *The problem REPEXT(CIRCLE) can be solved in polynomial time.*

To spice up our results, we show in the full version of the paper the following.

Proposition 2. *For k part of the input, the problem SIM(CIRCLE) is NP-complete.*

Corollary 3. *The problem SIM(CIRCLE) is FPT in the size of the common subgraph.*

2 Definitions and Preliminaries

Circle Representations. A *circle graph representation* \mathcal{R} is a collection of chords a circle $\{C_u \mid u \in V(G)\}$ such that C_u intersects C_v if and only if $uv \in E(G)$. A graph is a *circle graph* if it has a circle representation, and we denote the class of circle graphs by CIRCLE.

Notice that the representation of a circle graph is completely determined by the circular order of the endpoints of the chords in the representation, and two chords C_u and C_v cross if and only if their endpoints alternate in this order. For convenience we label both endpoints of the chord representing a vertex by the same label as the vertex.

A *partial representation* \mathcal{R}' is a representation of an induced subgraph G' . The vertices of G' are *pre-drawn* vertices and the chords of \mathcal{R}' are *pre-drawn chords*. A representation \mathcal{R} *extends* \mathcal{R}' if $C_u = C'_u$ for every $u \in V(G')$.

Word Representations. A sequence τ over an alphabet of symbols Σ is a *word*. A *circular word* represents a set of words which are cyclical shifts of one another. In the sequel, we represent a circular word by a word from its corresponding set of words. We denote words and circular words by small Greek letters.

For a word τ and a symbol u we write $u \in \tau$, if u appears at least once in τ . Thus, τ is also used to denote the set of symbols occurring in τ . A word τ is a *subword* of σ , if τ appears consecutively in σ . A word τ is a *subsequence* of σ ,

if the word τ can be obtained from σ by deleting some symbols. We say that u *alternates* with v in τ , if uvw or vuv is a subsequence of τ . The corresponding definitions also apply to circular words. If σ and τ are two words, we denote their concatenation by $\sigma\tau$.

From now on each representation \mathcal{R} of G corresponds to the unique circular word τ over V . The word τ is obtained by the circular order of the endpoints of the chords in \mathcal{R} as they appear along the circle when traversed clockwise. The occurrences of u and v alternate in τ if and only if $uv \in E(G)$. For example \mathcal{R} in Fig. 1 corresponds to the circular word $\tau = susxvxtutvwv$.

Let G be a circle graph, and let \mathcal{R} be its representation with the corresponding circular word τ . If G' is an induced subgraph of G , then the subsequence of τ consisting of the vertices in G' is a circular word σ . This σ corresponds to a representation \mathcal{R}' of G' which is extended by \mathcal{R} .

3 Structure of Representations of Splits

Let G be a connected graph. A *split* of G is a partition of the vertices of G into four parts $A, B, \mathfrak{s}(A)$ and $\mathfrak{s}(B)$, such that:

- For every $a \in A$ and $b \in B$, we have $ab \in E(G)$.
- There is no edge between $\mathfrak{s}(A)$ and $B \cup \mathfrak{s}(B)$, and between $\mathfrak{s}(B)$ and $A \cup \mathfrak{s}(A)$.
- Both sides of the split have at least two vertices: $|A \cup \mathfrak{s}(A)| \geq 2$ and $|B \cup \mathfrak{s}(B)| \geq 2$.

Fig. 3 shows two possible representations of a split. Notice that a split is uniquely determined just by the sets A and B , since $\mathfrak{s}(A)$ consists of connected components of $G \setminus (A \cup B)$ attached to A , and similarly for $\mathfrak{s}(B)$ and B . We refer to this split as a split *between* A and B .

In this section, we examine the recursive structure of every possible representation of G based on splits.

3.1 Split Structure of a Representation

Let \mathcal{R} be a representation of a graph G with a split between A and B . The representation \mathcal{R} corresponds to a unique circular word τ and we consider the

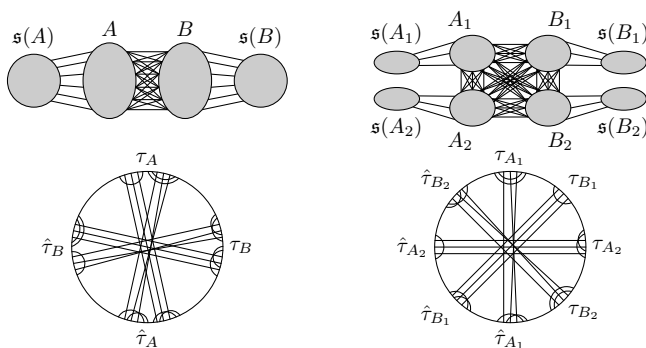


Fig. 3. Two different representations of G with the split between A and B .

circular subsequence γ induced by $A \cup B$. The maximal subwords of γ consisting of A alternate with the maximal subwords of γ consisting of B . We denote all these maximal subwords $\gamma_1, \dots, \gamma_{2k}$ according to their circular order; so $\gamma = \gamma_1 \gamma_2 \cdots \gamma_{2k}$. Without loss of generality, we assume that γ_1 consists of symbols from A . We call γ_i an A -word when i is odd, and a B -word when i is even.

We first investigate for each γ_i which symbols it contains.

Lemma 4. *For the subwords $\gamma_1, \dots, \gamma_k$ the following holds:*

- (a) *Each γ_i contains each symbol at most once.*
- (b) *The opposite words γ_i and γ_{i+k} contains the same symbols.*
- (c) *Let $i \neq j$. If $x \in \gamma_i$ and $y \in \gamma_j$, then $xy \in E(G)$.*

Proof (Sketch). These three properties are easily forced by the structure of the split; see the full version. \square

If $A, B \subset V(G)$ give rise to a split in G , we call the vertices of A and B the *long vertices* with respect to the split between A and B . Similarly the vertices $\mathfrak{s}(A)$ and $\mathfrak{s}(B)$ are called *short vertices* with respect to the split between A and B . In the sequel, if the split is clear from the context, we will just call some vertices long and some vertices short.

Consider a connected component C of $\mathfrak{s}(A)$ (for a component of $\mathfrak{s}(B)$ the same argument applies) and consider the subsequence of τ induced by $A \cup B \cup C$. By Lemma 4(a)-(b) and the fact that no vertex of $\mathfrak{s}(A)$ is adjacent to B , this subsequence almost equals γ . The only difference is that one subword γ_i is replaced by a subword which additionally contains all occurrences of the vertices of C . By accordingly adding the vertices of all components of $\mathfrak{s}(A)$ and $\mathfrak{s}(B)$ to γ , we get τ . Thus, τ consists of the circular subwords τ_1, \dots, τ_{2k} concatenated in this order, where τ_i is obtained from γ_i by adding the components of $\mathfrak{s}(A)$ or $\mathfrak{s}(B)$ attached to it. In particular, we also have the following:

Lemma 5. *If two long vertices $x, y \in A$ are connected by a path having the internal vertices in $\mathfrak{s}(A)$, then x and y belong to the same pair γ_i and γ_{i+k} in any representation.*

Proof. If x and y belong to different subwords γ_i and γ_j , where $i < j$ and $j \neq i, i+k$, of γ , by Lemma 4(a)-(b) any path connecting x and y has an internal vertex adjacent to a vertex of B . However, no vertex in $\mathfrak{s}(A)$ is adjacent to a vertex of B . \square

3.2 Conditions Forced by a Split

Now, we want to investigate the opposite relation. Namely, what can one say about a representation from the structure of a split? Suppose that x and y are two long vertices. We want to know the properties of x and y which force every representation \mathcal{R} to have a subword γ_i of γ containing both x and y .

We define a relation \sim on $A \cup B$ where $x \sim y$ means that x, y has to be placed in the same subword γ_i of γ . This relation is given by two conditions:

- (C1) Lemma 4(c) states that if $xy \notin E(G)$, then $x \sim y$, i.e., if x and y are placed in different subwords, then C_x intersects C_y .

(C2) Lemma 5 gives $x \sim y$ when x and y are connected by a non-trivial path with all the inner vertices in $\mathfrak{s}(A) \cup \mathfrak{s}(B)$.

Let us take the transitive closure of \sim , which we denote by \sim thereby slightly abusing the notation. Thus, we obtain an equivalence relation \sim on $A \cup B$. Notice that every equivalence class of \sim is either fully contained in A or in B . For the graph in Fig. 3, the relation \sim has four equivalence classes A_1, A_2, B_1 and B_2 .

Now, let Φ be an equivalence class of \sim . We denote by $\mathfrak{s}(\Phi)$ the set consisting of all the vertices in the connected components of $G \setminus (A \cup B)$ which have a vertex adjacent to a vertex of Φ . Since \sim satisfies (C2), we know that the sets $\mathfrak{s}(\Phi)$ of the equivalent classes of \sim define a partition of $\mathfrak{s}(A) \cup \mathfrak{s}(B)$.

Recognition Algorithms Based on Splits. The splits are used in the current state-of-the-art algorithms for recognizing circle graphs. If a circle graph contains no split, it is called a *prime graph*. The representation of a prime graph is uniquely determined (up to the orientation of the circle) and can be constructed efficiently. There is an algorithm which finds a split between two sets A and B in linear time [9]. In fact, the entire *split decomposition tree* (i.e., the recursive decomposition tree obtained via splits) can be found in linear time. Usually the representation \mathcal{R} is constructed as follows.

We define two graphs G_A and G_B where G_A is a subgraph of G induced by the vertices corresponding to $A \cup \mathfrak{s}(A) \cup \{v_A\}$ where the vertex v_A is adjacent to all the vertices in A and non-adjacent to all the vertices in $\mathfrak{s}(A)$, and G_B is defined similarly for $B, \mathfrak{s}(B)$, and v_B . Then we apply the algorithm recursively on G_A and G_B and construct their representations \mathcal{R}_A and \mathcal{R}_B ; see Fig. 4. It remains to join the representations \mathcal{R}_A and \mathcal{R}_B in order to construct \mathcal{R} .

To this end we take \mathcal{R}_A and replace C_{v_A} by the representation of $B \cup \mathfrak{s}(B)$ in \mathcal{R}_B . More precisely, let the circular ordering of the endpoints of chords defined by \mathcal{R}_A be $v_A \tau_A v_A \hat{\tau}_A$ and let the circular ordering defined by \mathcal{R}_B be $v_B \tau_B v_B \hat{\tau}_B$. The constructed \mathcal{R} has the corresponding circular ordering $\tau_A \tau_B \hat{\tau}_A \hat{\tau}_B$. It is easy to see that \mathcal{R} is a correct circle representation of G .

Structure of All Representations. The above algorithm constructs a very specific representation \mathcal{R} of G , and a representation like the one in Fig. 3 on the right cannot be constructed by the algorithm. In what follows we describe a structure of all the representations of G , based on different circular orderings of the classes of \sim . First, we show that every representation obtained in this way is a correct representation of G . Second, we prove that every representation \mathcal{R} of G can be constructed like this.

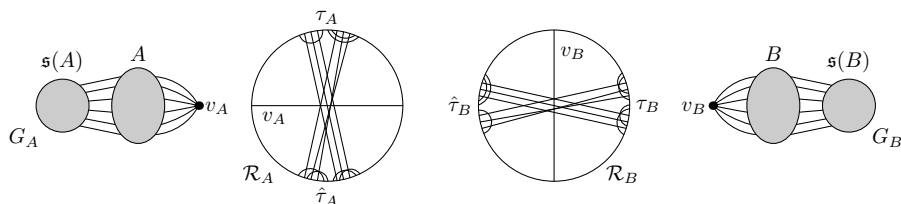


Fig. 4. The graphs G_A and G_B together with some constructed representations \mathcal{R}_A and \mathcal{R}_B . By joining these representations, we get the left representation of Fig. 3.

We choose an arbitrary circular ordering Φ_1, \dots, Φ_ℓ of the classes of \sim . Let G_i be a graph constructed from G by contracting the vertices of $V(G) \setminus (\Phi_i \cup \mathfrak{s}(\Phi_i))$ into one vertex v_i ; i.e., G_i is defined similarly to G_A and G_B above. Let $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ be arbitrary representations of G_1, \dots, G_ℓ . We join these representations as follows. Let $v_i \tau_i v_i \hat{\tau}_i$ be the circular ordering of \mathcal{R}_i . We construct \mathcal{R} as the circular ordering

$$\tau_1 \tau_2 \dots \tau_{k-1} \tau_k \hat{\tau}_1 \hat{\tau}_2 \dots \hat{\tau}_{k-1} \hat{\tau}_k. \tag{1}$$

In Fig. 3, we obtain the representation on the left by the circular ordering $A_1 A_2 B_1 B_2$ of the classes of \sim and the representation on the right by $A_1 B_1 A_2 B_2$.

Lemma 6. *Every circular ordering (1) constructed as above defines a circle representation of G .*

Proof. Every long vertex $u \in \Phi_i$ alternates with v_i in \mathcal{R}_i and every short vertex $v \in \mathfrak{s}(\Phi_i)$ has both occurrences either in τ_i , or in $\hat{\tau}_i$, since it is not adjacent to v_i . Thus, we get a correct representation \mathcal{R} of G . \square

We now show that this approach can construct every representation of G .

Lemma 7. *Let τ be the circular word corresponding to a representation \mathcal{R} of G . Then the symbols of $\Phi_i \cup \mathfrak{s}(\Phi_i)$ form exactly two subwords of τ .*

Proof (Sketch). The conditions (C1) and (C2) imply consecutivity for some pairs x and y . The statement is then proved by applying induction on the classes of \sim ; see the full version for details. \square

Now, we are ready to prove the main structural proposition, which is inspired by Section IV.4 of the thesis of Naji [21].

Proposition 8. *Let \sim be the equivalence relation defined by (C1) and (C2) on $A \cup B$. Then every representation \mathcal{R} corresponds to some circular ordering Φ_1, \dots, Φ_ℓ and to some representations $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ of G_1, \dots, G_ℓ . More precisely, \mathcal{R} can be constructed by arranging $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ as in (1): $\tau_1 \dots \tau_k \hat{\tau}_1 \dots \hat{\tau}_k$.*

Proof. Let \mathcal{R} be any representation with the corresponding circular word τ . According to Lemma 7, we know $\Phi_i \cup \mathfrak{s}(\Phi_i)$ forms two subwords τ_i and $\hat{\tau}_i$ of τ . For $i \neq j$, the edges between Φ_i and Φ_j form a complete bipartite graph. The subwords $\tau_i, \hat{\tau}_i, \tau_j$ and $\hat{\tau}_j$ alternate, i.e., appear as $\tau_i \tau_j \hat{\tau}_i \hat{\tau}_j$ or $\tau_j \tau_i \hat{\tau}_j \hat{\tau}_i$ in τ . Thus, if we start from some point along the circle, the order of τ_i 's gives a circular ordering Φ_1, \dots, Φ_ℓ of the classes. The representation \mathcal{R}_i is given by the circular ordering $v_i \tau_i v_i \hat{\tau}_i$. \square

4 Algorithm

In this section, we give a polynomial-time algorithm for the partial representation extension problem of circle graphs. Our algorithm is based on the structure of all representations described in Section 3. We assume that the graph is connected and we deal with disconnected graphs in the full version of this paper.

Overview. Let τ' be the circular word corresponding to the given partial rep. \mathcal{R}' . We want to extend τ' to a circular word τ corresponding to a rep. \mathcal{R} of G .

Our algorithm proceeds recursively via split decomposition.

1. If G is prime, we have two possible representations (one is reversal of the other) and we test whether one of them is compatible with \mathcal{R}' .
2. Otherwise, we find a split and compute the \sim relation.
3. We test whether some ordering Φ_1, \dots, Φ_ℓ of these classes along the circle is compatible with the partial representation \mathcal{R}' . This order is partially prescribed by short chords and long chords of \mathcal{R}' .
4. If no ordering is compatible, we stop and output “no”. If there is an ordering which is compatible with \mathcal{R}' , we recurse on the graphs G_1, \dots, G_ℓ constructed according to the equivalence classes of \sim .

Now we describe everything in detail.

Splits. Now we assume that the graph is not prime, otherwise the problem is easy to solve (details will be given in the full version). A split between A and B is called *trivial* if for one side, let us say A , we have $|A| = 1$ and $|\mathfrak{s}(A)| = 1$. If G contains only trivial splits, then we call it *trivial*. For technical purposes, we assume that the split is non-trivial, again the full version of this paper contains the details.

So we have a non-trivial split between A and B which can be constructed in polynomial time [9]. We compute the equivalence relation \sim and we want to find an ordering of its equivalence classes. For a class Φ of \sim , we define the *extended class* Ψ of \sim as $\Phi \cup \mathfrak{s}(\Phi)$. We can assume that each extended class has a vertex pre-drawn in the partial representation, otherwise any representation of it is good. So \sim has ℓ equivalence classes, and all of them appear in τ' .

Now, τ' is composed of k *maximal subwords*, each containing only symbols of one extended class Ψ . We denote these maximal subwords as τ'_1, \dots, τ'_k according to their circular order in τ' , so $\tau' = \tau'_1 \cdot \dots \cdot \tau'_k$. According to Proposition 8, each extended class Ψ corresponds to at most two different maximal subwords. Also, if two extended classes Ψ and $\hat{\Psi}$ correspond to two different maximal subwords, then occurrences of these subwords in τ' alternate. Otherwise we reject the input.

Case 1: An extended class corresponds to two maximal subwords.

We denote this class by Ψ_1 and put this class as first in the ordering. By renumbering, we may assume that Ψ_1 corresponds to τ'_1 and τ'_t . Then one circular order of the classes can be determined as follows. We have $\Psi_1 < \Psi$ for any other class Ψ . Let Ψ_i and Ψ_j be two distinct classes. If Ψ_i corresponds to τ'_a and Ψ_j corresponds to τ'_b such that either $a < b < t$ or $t < a < b$, we put $\Psi_i < \Psi_j$. We obtain the ordering of the classes as any linear extension of $<$. One can observe that $<$ is acyclic, otherwise the maximal subwords would not alternate correctly.

Now, we have ordered the extended classes Ψ_1, \dots, Ψ_ℓ and the corresponding classes Φ_1, \dots, Φ_ℓ . We construct each G_i with the vertices $\Psi_i \cup \{v_i\}$ as in Section 3.2, so v_i is adjacent to Φ_i and non-adjacent to $\mathfrak{s}(\Phi_i)$. As the partial representation \mathcal{R}'_i of G_i , we put the word $v_i \tau'_i v_i \tau'_j$ where Ψ_i corresponds to τ_i and τ_j (possibly one of them is empty). We test recursively, whether each representation \mathcal{R}'_i of G_i is extendible to a representation of \mathcal{R}_i . If yes, we join $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ as in Proposition 8. Otherwise, the algorithm outputs “no”.

Lemma 9. *For Case 1, the representation \mathcal{R}' is extendible if and only if the representations $\mathcal{R}'_1, \dots, \mathcal{R}'_\ell$ of the graphs G_1, \dots, G_ℓ are extendible.*

Proof. Suppose that \mathcal{R} extends \mathcal{R}' . According to Proposition 8, the representations of Ψ_1, \dots, Ψ_ℓ are somehow ordered along the circle, and so we obtain representations $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ extending $\mathcal{R}'_1, \dots, \mathcal{R}'_\ell$.

For the other implications, we just take $\mathcal{R}_1, \dots, \mathcal{R}_\ell$ and put them in \mathcal{R} together as in (1). The ordering $<$ was constructed exactly in such a way that \mathcal{R} extends \mathcal{R}' . \square

Case 2: No extended class corresponds to two maximal subwords

In this case, we have the ordering of the classes according to their appearance in τ' , so Ψ_i corresponds to the subword τ'_i . According to Proposition 8, we know that in any representation \mathcal{R} of G the class Ψ_i corresponds to two subwords τ_i and $\hat{\tau}_i$. The difficulty here arises from the potential for τ'_i to be a subsequence of only one of τ_i and $\hat{\tau}_i$.

We solve this as follows. Instead of constructing just one graph G_i with one partial representation \mathcal{R}'_i , we construct an additional graph \tilde{G}_i with a partial representation $\tilde{\mathcal{R}}'_i$ as follows. The graph \tilde{G}_i is G_i with an additional leaf w_i attached to v_i . The partial representation $\tilde{\mathcal{R}}'_i$ corresponds to the word $\tau'_i v_i v_i$ and the partial representation $\tilde{\mathcal{R}}'_i$ corresponds to $\tau'_i w_i w_i$. The difference is that $\tilde{\mathcal{R}}'_i$ is less restrictive and only one endpoint of v_i is prescribed (i.e., the location of the “other” end of v_i is not restricted). We can easily observe that if \mathcal{R}'_i is extendible, then $\tilde{\mathcal{R}}'_i$ is also extendible.

The following lemma is fundamental for the algorithm, and it states that at most one class can be forced to use \tilde{G}_i with $\tilde{\mathcal{R}}'_i$, if τ' is extendible:

Lemma 10. *The representation \mathcal{R}' is extendible if and only if $\tilde{\mathcal{R}}'_i$ is extendible for some i and \mathcal{R}'_j is extendible for all $j \neq i$.*

Proof. Suppose that \mathcal{R}_j corresponding to a word $v_j \tau_j v_j \hat{\tau}_j$ is an extension of \mathcal{R}'_j for $j \neq i$. And let \mathcal{R}_i corresponding to a word $w_i v_i w_i \tau_i v_i \hat{\tau}_i$ be an extension of $\tilde{\mathcal{R}}'_i$. Then the representation \mathcal{R} (after removing w_i) constructed as in (1) extends \mathcal{R}' .

For the other implication, suppose that \mathcal{R} extends \mathcal{R}' . For contradiction, suppose that two distinct partial representations \mathcal{R}'_i and \mathcal{R}'_j are not extendible. According to Proposition 8, the representation \mathcal{R} gives a representation \mathcal{R}_i corresponding to $v_i \tau_i v_i \hat{\tau}_i$ of G_i and \mathcal{R}_j corresponding to $v_j \tau_j v_j \hat{\tau}_j$ of G_j . But since both Ψ_i and Ψ_j correspond to single maximal words of τ' , we have that τ'_i is a subsequence of τ_i or $\hat{\tau}_i$, or τ'_j is a subsequence of τ_j or $\hat{\tau}_j$, and so \mathcal{R}'_i or \mathcal{R}'_j is extendible. Contradiction. \square

For the algorithm, we can efficiently test which of \mathcal{R}'_i and $\tilde{\mathcal{R}}'_i$ are extendible with the pseudocode of Algorithm 1.

Analysis of the Algorithm. By using the established results, we show that the partial representation extension problem of circle graphs can be solved in polynomial time.

Lemma 11. *The described algorithm correctly decides whether the partial representation \mathcal{R}' of G' is extendible.*

Algorithm 1. Subroutine for Case 2.

1. Let Ψ_1 be the largest class (i.e., $|\Psi_i| \leq n/2$ for $i > 1$).
 2. **If** $\mathcal{R}'_2, \dots, \mathcal{R}'_\ell$ are extendible **then**
 3. **If** $\tilde{\mathcal{R}}'_1$ is extendible **then** ACCEPT **else** REJECT.
 4. **Else if** only \mathcal{R}'_i is not extendible **then**
 5. **If** $\tilde{\mathcal{R}}'_i$ and \mathcal{R}'_1 are extendible **then** ACCEPT **else** REJECT.
 6. **Else** REJECT.
-

Proof (Sketch). We just put together the lemmas which are already proved; see the full version for details. \square

The next lemma states that the algorithm runs in polynomial time. A precise time analysis depends on algorithm used for split decomposition, and on the order in which we choose splits for recursion. We avoid this technical analysis and just note that the degree of the polynomial is reasonable small. Certainly, it would be easy to show the complexity of order $\mathcal{O}(nm)$.

Lemma 12. *The running time of the algorithm is polynomial.* \square

Proof (Theorem 1). The result is implied by Lemma 11 and Lemma 12. \square

5 Conclusions

The structural results described in Section 3, namely Proposition 8, are the main new tools developed in this paper. Using it, one can easily work with the structure of all representations which is a key component of the algorithm of Section 4 that solves the partial representation extension problem for circle graphs. The algorithm works with the recursive structure of all representations and matches the partial representation on it. Proposition 8 also seems to be useful in attacking the following open problems:

Question 13. What is the complexity of $\text{SIM}(\text{CIRCLE})$ for a fixed number k of graphs? In particular, what is it for $k = 2$?

Recall that in the bounded representation problem, we give for some chords two circular arcs and we want to construct a representation which places endpoints into these circular arcs.

Question 14. What is the complexity of the bounded representation problem for circle graphs?

References

1. Angelini, P., Battista, G.D., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. In: SODA 2010, pp. 202–221 (2010)
2. Balko, M., Klavík, P., Otachi, Y.: Bounded representations of interval and proper interval graphs. In: ISAAC (to appear, 2013)

3. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. In: SODA 2013, pp. 1030–1043 (2013)
4. Bouchet, A.: Reducing prime graphs and recognizing circle graphs. *Combinatorica* 7(3), 243–254 (1987)
5. Bouchet, A.: Unimodularity and circle graphs. *Discrete Mathematics* 66(1-2), 203–208 (1987)
6. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. *Annals of Mathematics* 164, 51–229 (2006)
7. Courcelle, B.: Circle graphs and monadic second-order logic. *J. Applied Logic* 6(3), 416–442 (2008)
8. Cunningham, W.: Decomposition of directed graphs. *SIAM J. Alg. and Disc. Methods* 3, 214–228 (1982)
9. Dahlhaus, E.: Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *Journal of Algorithms* 36(2), 205–240 (1998)
10. de Fraysseix, H.: Local complementation and interlacement graphs. *Discrete Mathematics* 33(1), 29–35 (1981)
11. de Fraysseix, H., de Mendez, P.O.: On a characterization of gauss codes. *Discrete & Computational Geometry* 22(2), 287–295 (1999)
12. Even, S., Itai, A.: Queues, stacks, and graphs. In: Kohavi, Z., Paz, A. (eds.) *Theory of Machines and Computation*, pp. 71–76 (1971)
13. Gabor, C.P., Supowit, K.J., Hsu, W.: Recognizing circle graphs in polynomial time. *J. ACM* 36(3), 435–473 (1989)
14. Gioan, E., Paul, C., Tedder, M., Corneil, D.: Practical and efficient circle graph recognition. *Algorithmica*, 1–30 (2013)
15. Jampani, K.R., Lubiw, A.: The simultaneous representation problem for chordal, comparability and permutation graphs. *Journal of Graph Algorithms and Applications* 16(2), 283–315 (2012)
16. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. In: Epstein, L., Ferragina, P. (eds.) *ESA 2012. LNCS*, vol. 7501, pp. 671–682. Springer, Heidelberg (2012)
17. Klavík, P., Kratochvíl, J., Otachi, Y., Rutter, I., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs (in preparation, 2013)
18. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T.: Extending partial representations of subclasses of chordal graphs. In: Chao, K.-M., Hsu, T.-S., Lee, D.-T. (eds.) *ISAAC 2012. LNCS*, vol. 7676, pp. 444–454. Springer, Heidelberg (2012)
19. Klavík, P., Kratochvíl, J., Vyskočil, T.: Extending partial representations of interval graphs. In: Ogihara, M., Tarui, J. (eds.) *TAMC 2011. LNCS*, vol. 6648, pp. 276–285. Springer, Heidelberg (2011)
20. Kostochka, A., Kratochvíl, J.: Covering and coloring polygon-circle graphs. *Discrete Mathematics* 163(1-3), 299–305 (1997)
21. Naji, W.: *Graphes de Cordes: Une Caractérisation et ses Applications*. PhD thesis, l’Université Scientifique et Médicale de Grenoble (1985)
22. Oum, S.: Rank-width and vertex-minors. *J. Comb. Theory, Ser. B* 95(1), 79–100 (2005)
23. Patrignani, M.: On extending a partial straight-line drawing. In: Healy, P., Nikolov, N.S. (eds.) *GD 2005. LNCS*, vol. 3843, pp. 380–385. Springer, Heidelberg (2006)
24. Spinrad, J.P.: Recognition of circle graphs. *J. of Algorithms* 16(2), 264–282 (1994)
25. Spinrad, J.P.: *Efficient Graph Representations*. Field Institute Monographs (2003)

On Balanced $\+$ -Contact Representations

Stephane Durocher* and Debajyoti Mondal**

Department of Computer Science, University of Manitoba, Canada
{durocher, jyoti}@cs.umanitoba.ca

Abstract. In a $\+$ -contact representation of a planar graph G , each vertex is represented as an axis-aligned plus shape consisting of two intersecting line segments (or equivalently, four axis-aligned line segments that share a common endpoint), and two plus shapes touch if and only if their corresponding vertices are adjacent in G . Let the four line segments of a plus shape be its arms. In a c -balanced representation, $c \leq 1$, every arm can touch at most $\lceil c\Delta \rceil$ other arms, where Δ is the maximum degree of G . The widely studied T - and L -contact representations are c -balanced representations, where c could be as large as 1. In contrast, the goal in a c -balanced representation is to minimize c . Let c_k , where $k \in \{2, 3\}$, be the smallest c such that every planar k -tree has a c -balanced representation. In this paper we show that $1/4 \leq c_2 \leq 1/3 (= b_2)$ and $1/3 < c_3 \leq 1/2 (= b_3)$. Our result has several consequences. Firstly, planar k -trees admit 1-bend box-orthogonal drawings with boxes of size $\lceil b_k \Delta \rceil \times \lceil b_k \Delta \rceil$, which generalizes a result of Tayu, Nomura, and Ueno. Secondly, they admit 1-bend polyline drawings with $2\lceil b_k \Delta \rceil$ slopes, which is significantly smaller than the 2Δ upper bound established by Keszegh, Pach, and Pálvölgyi for arbitrary planar graphs.

1 Introduction

In a contact representation of a planar graph G , the vertices of G are represented using different non-overlapping geometric shapes (e.g., lines, triangles, or circles) and the adjacencies are represented by the contacts of the corresponding objects. Contact representations arise in many applied fields, such as cartography, VLSI floor-planning, and data visualization, which has motivated extensive research over the past several decades. In this paper we examine $\+$ -contact representations of planar graphs, i.e., each vertex in such a representation Γ corresponds to an axis-aligned plus shape, two plus shapes never cross, but touch if and only if their corresponding vertices are adjacent in the input planar graph. Let the four orthogonal parts associated with a plus symbol be its *left*, *right*, *up* and *down* arms. We call Γ a c -balanced representation, where $c \leq 1$, if every arm in Γ touches at most $\lceil c\Delta \rceil$ other arms, where Δ is the maximum degree of the

* Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

** Work of the author is supported in part by a University of Manitoba Graduate Fellowship.

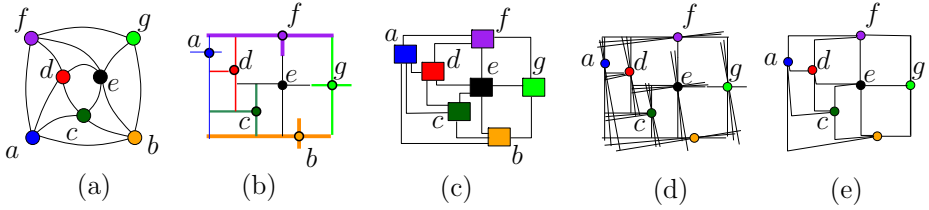


Fig. 1. (a) A graph G with $\Delta = 5$. (b) A $(1/2)$ -balanced \blackplus -contact representation of G . (c) A box-orthogonal drawing of G . (d)–(e) A transformation into a polyline drawing.

underlying graph. The horizontal (or vertical) segments of two touching plus shapes in Γ may be collinear, e.g., the shapes representing the vertices e and g in Figures 1(a)–(b).

In 1994, de Fraysseix et al. [1] gave an algorithm to construct contact representations of planar graphs with axis-aligned T shapes. Many studies followed to characterize classes of planar graphs that admit contact representations with shapes simpler than T , such as axis-aligned segments [2] and L shapes [3]. L - and T -contact representations can be viewed as c -balanced \blackplus -contact representations, however, c may be required to be as large as 1. On the other hand, in a c -balanced representation, our goal is to minimize c .

Box-Orthogonal Drawings with Small Boxes of Constant Aspect Ratio.

Balanced \blackplus -contact representations are useful in the study of box-orthogonal drawings in \mathbb{R}^2 . A k -bend box-orthogonal drawing of a planar graph G is a planar drawing of G , where each vertex is represented as an axis-aligned box and each edge is drawn as an orthogonal polygonal chain with at most k bends. Every \blackplus -contact representation can be transformed into a box-orthogonal drawing [4], as shown in Figure 1(c). Some important aesthetics of a box-orthogonal drawing are the number of bends per edge, and the aspect ratio and size of the boxes. Biedl and Kaufmann [4] showed that every planar graph admits a 1-bend box-orthogonal drawing on an integer grid, but the width or height of a box in such a drawing could be as large as Δ . A c -balanced \blackplus -contact representation implies a 1-bend box-orthogonal drawing with boxes of size $\lceil c\Delta \rceil \times \lceil c\Delta \rceil$.

Orthogonal drawings are box-orthogonal drawings with boxes of degenerate shapes, i.e., points. The graphs that admit orthogonal drawings are of maximum degree four. Hence a 0- and 1-bend orthogonal drawing gives a $(1/4)$ -balanced \blackplus -contact representation. There have been several attempts in the literature to characterize the graphs that admit 0- and 1-bend orthogonal drawings [5,6]. Recently, Tayu, Nomura, and Ueno [7] showed that every 2-tree with maximum degree four admits a 1-bend orthogonal drawing. In this paper we show that 2-trees and planar 3-trees admit $(1/3)$ - and $(1/2)$ -balanced \blackplus -contact representations, respectively, and thus admit 1-bend box-orthogonal drawings with boxes of size $\lceil \Delta/3 \rceil \times \lceil \Delta/3 \rceil$ and $\lceil \Delta/2 \rceil \times \lceil \Delta/2 \rceil$, respectively.

Planar Slope Number with One Bend Per Edge.

A k -bend *polyline drawing* of a planar graph G is a planar drawing Γ of G , where each vertex is represented as a point and each edge is drawn as a polygonal chain with at most k bends. Γ is

a t -slope drawing of G if the number of distinct slopes used by the line segments in Γ is at most t . The *planar slope number* of G is the smallest number t such that G admits a t -slope 0-bend drawing. A rich body of literature examines planar slope number of different subclasses of planar graphs [8,9,10]. Keszegh et al. [11] proved a q^Δ upper bound on the planar slope number, where q is a constant. They also showed that every planar graph G admits a 1-bend polyline drawing with at most 2Δ slopes, by a transformation from T -contact representations into 1-bend polyline drawings, as follows. Replace each vertical (respectively, horizontal) arm with Δ closely spaced nearly vertical (respectively, horizontal) slopes, e.g., see Figure 1(d). Finally, choose the bend points from the intersection points of these slopes such that the resulting drawing remains planar, e.g., see Figure 1(e). In this paper we show that 2-trees and planar 3-trees admit $(1/3)$ - and $(1/2)$ -balanced \dagger -contact representations, respectively, and thus admit 1-bend polyline drawings with at most $2\lceil\Delta/3\rceil$ slopes, and $2\lceil\Delta/2\rceil$ slopes, respectively.

2 Definitions and Preliminary Approach

In this section we introduce some definitions and construct $(1/2)$ -balanced \dagger -contact representations for 2-trees.

A *2-tree*, or *series-parallel graph* (SP graph) G is a two-terminal directed simple graph with $n \geq 2$ vertices, which is defined recursively as follows.

- (a) If $n = 2$, then G has a single edge (u, v) , where either u or v is the source and the other vertex is the sink.
- (b) If $n > 2$, then G can be constructed from two SP graphs G_1 and G_2 from one of the following two operations, e.g., see Figure 2(a).
 - *Series Composition*: Identify the sink of G_1 with the source of G_2 .
 - *Parallel Composition*: Identify the source and sink of G_1 with the source and sink of G_2 , respectively. Finally, identify any parallel edges.

A c -balanced representation of a given SP graph G can be constructed as follows. Construct a rectangle R and place the source and sink of G at the top-left and bottom-right corners, respectively. Initially each edge of R can have $\lceil c\Delta \rceil$ contact points. If G is formed by a series composition of two SP graphs G_1 and G_2 , then we split R into four rectangles, e.g., see Figure 2(b), and draw G_1 and G_2 into the top-left and bottom-right rectangles, respectively. If G is formed by a parallel composition of G_1 and G_2 , then we take two copies $R_i, i \in \{1, 2\}$, of R and draw G_i inside R_i (later on we merge these two drawings inside R). In both series and parallel cases, we distribute the available contact points among the subproblems, i.e., we compute the recursive drawings with bounded number of contact points on the edges of their bounding rectangles. In our algorithms, we specify the distribution of contact points so that we can merge the recursively computed drawings maintaining planarity.

Let h be an arm of some vertex while constructing a \dagger -contact representation. By the *number of free points* of h we refer to the number of other arms that can touch h , which we denote by $f(h)$. If $f(h) = 0$, then we say h is *saturated*,

otherwise h is *unsaturated*. The *center* of a vertex is the point, at which all four of its arms meet. For a center m , we denote by m_l, m_r, m_u, m_d the left, right, up and down arms of m , respectively. *Distributing* an integer z among the arms of m in some order $\sigma = (m_d, m_r, m_u, m_l)$ is an operation that finds the first arm h such that $z \leq \sum_{h' \leq_\sigma h} f(h')$, then sets $f(h) = z - \sum_{h' <_\sigma h} f(h')$, and finally, for all arms h'' subsequent to h , sets $f(h'') = 0$. Such an operation is defined only when $z \leq \sum_h f(h)$. By $d_i(v, G)$ and $d_o(v, G)$ we denote the in-degree and out-degree of vertex v in G . We omit the term G if it is clear from the context. We now present a construction of $(1/2)$ -balanced representations of SP graphs.

Lemma 1. *Let G be a SP graph with source s and sink t , and let G' be the graph obtained from G by deleting the edge (s, t) , if such an edge exists. Let $R = abcd$ be an axis-parallel rectangle such that s and t lie on the opposite corners a and c , respectively. Assume that $f(a_d), f(a_r), f(c_l), f(c_u)$ are prespecified. If $d_o(s, G') \leq f(a_d) + f(a_r)$ and $d_i(t, G') \leq f(c_l) + f(c_u)$, then G' admits a $(1/2)$ -balanced \blackstar -contact representation Γ in R satisfying the following property.*

- (\star) *The number of contact points at each arm incident to s and t in Γ is at most the number of free points specified for that arm as input.*

Proof. We employ an induction on the number of vertices n of G . If $n = 2$, then G' consists of two vertices of degree zero, i.e., s and t , that lie on the two opposite corners a and c of R , respectively. It is now straightforward to verify Property (\star). Hence assume that $n > 2$, and the lemma holds for every G that has fewer than n vertices. We now consider the case when G has n vertices.

Since G is a SP graph and $n > 2$, G' must be a SP graph, i.e., G' is obtained either by a series combination or a parallel composition of some SP graphs G_1 and G_2 . Let s_i and t_i be the source and sink of G_i , respectively, where $i \in \{1, 2\}$. We now consider two cases depending on the composition of G_1 and G_2 in G' .

Case 1 (Series Composition): In this case $s = s_1, t_1 = s_2$ and $t_2 = t$. We first define two rectangles R_1 and R_2 inside R , where G_1 and G_2 will be drawn, respectively. To construct R_1 and R_2 we first add a vertex r inside R , which corresponds to the center of vertex $t_1 (= s_2)$. We then draw four orthogonal line segments re, rm, rg, rh such that $e \in ab, m \in bc, g \in cd, h \in ad$. Then $R_1 = aerh$ and $R_2 = rmcg$, as in Figure 2(b). We set $f(r_l) = f(r_r) = f(r_u) = f(r_d) = \lfloor \Delta/2 \rfloor$, and then assign the free points of s and t to s_1 and t_2 , respectively.

If the edge (s_1, t_1) exists, then we draw (s_1, t_1) either along the polygonal chain ahr or aer , depending on whether $f(a_d) = 0$ or not. If the edge (s_2, t_2) exists, then we draw (s_2, t_2) either along the polygonal chain rgc or rmc , depending on whether $f(c_l) = 0$ or not. Here we consider the case when both (s_1, t_1) and (s_2, t_2) exist (the other cases can be treated similarly). Figure 2(c) shows such an example, where $f(a_d) \neq 0$ and $f(c_l) = 0$. Observe that while drawing (s_1, t_1) and (s_2, t_2) , we use some free points of s_1 and t_2 . Therefore, we decrease the free points by one for each arm that helps routing (s_1, t_1) and (s_2, t_2) , e.g., see Figures 2(d)–(e). Since $d_o(s, G') \leq f(a_d) + f(a_r)$ in R , we have $d_o(s_1, G_1 \setminus (s_1, t_1)) \leq f(a'_d) + f(a'_r)$, where a' represents a in R_1 . Since $d_i(t_1, G_1 \setminus (s_1, t_1)) \leq \Delta - 1$, we have $d_i(t_1, G_1 \setminus (s_1, t_1)) \leq f(r_u) + f(r_l)$ in R_1 . Therefore, we can

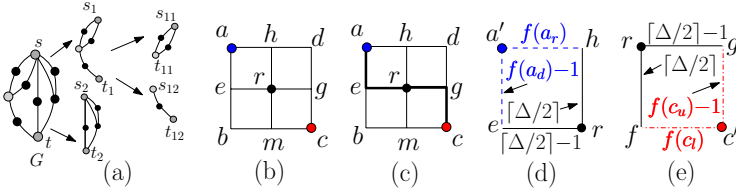


Fig. 2. (a) A few steps of series-parallel decomposition for some graph G , according to the definition. (b) Computation of R_1 and R_2 . (c) Drawing (s_1, t_1) and (s_2, t_2) . (d)–(e) Illustration for free points.

inductively draw G_i inside R_i , $i \in \{1, 2\}$. It is straightforward to merge these drawings by appropriate scaling. Since the drawings inside R_i maintain Property (\star) , the merged drawing also satisfies that property.

Case 2 (Parallel Composition): In this case $s = s_1 = s_2$ and $t = t_1 = t_2$. We first create two copies R_1 and R_2 of R , i.e., $R_1 = a'b'c'd'$ and $R_2 = a''b''c''d''$, where G_1 and G_2 will be drawn, respectively. Figure 3 illustrates an example.

We now define the free points of the arms of s_1, t_1 and s_2, t_2 that are inside R_1 and R_2 , respectively. We distribute $d_o(s_1)$ among a'_d and a'_r in this order, i.e., we set $f(a'_d) = \min\{f(a_d), d_o(s_1)\}$, and $f(a'_r) = \max\{0, d_o(s_1) - f(a'_d)\}$. Similarly, distribute $d_i(t_1)$ among c'_l and c'_u in this order, i.e., set $f(c'_l) = \min\{f(c_l), d_i(t_1)\}$, and $f(c'_u) = \max\{0, d_i(t_1) - f(c'_l)\}$, e.g., Figure 3(b). The number of free points of s_2 and t_2 is the number of free points of s and t that remains after assigning free points to s_1 and t_1 , as shown in Figure 3(c).

Since $d_o(s) \leq f(a_d) + f(a_r)$ and $d_o(s) = d_o(s_1) + d_o(s_2)$, according to our assignment of free points, $d_o(s_1) \leq f(a'_d) + f(a'_r)$. Similarly, since $d_i(t) \leq f(c_l) + f(c_u)$ and $d_i(t) = d_i(t_1) + d_i(t_2)$, we obtain $d_i(t_1) \leq f(c'_l) + f(c'_u)$. It is now straightforward to observe that $d_o(s_2) \leq f(a''_d) + f(a''_r)$ and $d_i(t_2) \leq f(c''_l) + f(c''_u)$. Therefore, by induction, can draw G_1 and G_2 inside R_1 and R_2 , respectively.

The drawing of G_1 takes consecutive free points from the arms of s (respectively, t) in anticlockwise (respectively, clockwise) order. The drawing of G_2 takes the remaining consecutive free points in the same order. Therefore, one can merge the two drawings inside R_1 and R_2 avoiding edge crossings inside R . The details are omitted due to space constraints. \square

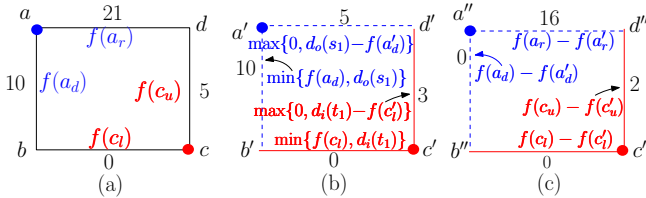


Fig. 3. (a) R . (b)–(c) Computation of R_1 and R_2 . The numbers exterior to the three rectangles illustrate a concrete example, where $d_o(s_1) = 15$ and $d_i(t_1) = 3$.

Theorem 1. *Every SP graph G has a $(1/2)$ -balanced \dagger -contact representation.*

Proof. If the source s and sink t of G are not adjacent, then by Lemma 1, G admits the required representation. Otherwise, let G' be the graph $G \setminus (s, t)$. By Lemma 1, G' admits a $(1/2)$ -balanced \dagger -contact representation inside a rectangle $R = abcd$, where s and t lie on the opposite corners a and c , respectively, and the free points $f(a_d), f(a_r), f(c_l), f(c_u)$ are prespecified such that $d_o(s, G') \leq f(a_d) + f(a_r)$ and $d_i(t, G') \leq f(c_l) + f(c_u)$.

We define $f(a_d) = \lceil \Delta/2 \rceil - 1, f(a_r) = \lceil \Delta/2 \rceil, f(c_l) = \lceil \Delta/2 \rceil - 1$ and $f(c_u) = \lceil \Delta/2 \rceil$. Since $d_o(s, G') \leq \Delta - 1$ and $d_i(t, G') \leq \Delta - 1$, the conditions $d_o(s, G') \leq f(a_d) + f(a_r)$ and $d_i(t, G') \leq f(c_l) + f(c_u)$ hold. Hence by Lemma 1, we can compute a $(1/2)$ -balanced \dagger -contact representation Γ' of G' inside R . Finally, we draw the edge (s, t) along the polygonal chain abc . \square

3 Balanced Representations for 2-Trees ($c = 1/3$)

The idea of the algorithm for computing $(1/3)$ -balanced \dagger -contact representations is similar to that of Section 2, however, here the construction is more involved. Let \overline{uv} denote the line segment from u to v . We first prove the following lemma, which is similar to Lemma 1.

Lemma 2. *Let G be a SP graph with source s and sink t , and let G' be the graph obtained from G by deleting the edge (s, t) , if such an edge exists. Let $R = \overline{k_1 k_2 k_3 k_4}$ be an axis-parallel rectangle such that s and t are centered at $a \in \overline{k_1 k_2}$ and $c \in \overline{k_2 k_3}$, respectively, but not at k_2 . Assume that the free points of the arms of s and t that lie on R are prespecified. Let x (respectively, y) be the total number of free points of all arms of s (respectively, t) that lie inside R . If $d_o(s, G') \leq x$ and $d_i(t, G') \leq y$, then G' admits a $(1/3)$ -balanced \dagger -contact representation Γ in R satisfying the following property.*

(\star) *The number of contact points at each arm incident to s and t in Γ is at most the number of free points specified for that arm as input.*

Proof. We employ an induction on the number of vertices n of G . The case when $n = 2$ is straightforward, hence we now assume that $n > 2$, and the lemma holds for every G that has fewer than n vertices. We now consider the case when G has n vertices. Since G is a SP graph and $n > 2$, G' must be a SP graph, i.e., G' is obtained either by a series or a parallel composition of some SP graphs G_1 and G_2 . Let s_j and t_j be the source and sink of G_j , respectively, where $j \in \{1, 2\}$. We consider two cases depending on the composition of G_1 and G_2 in G' .

Case 1 (Series Composition): We first construct two rectangular regions R_1 and R_2 inside R , where G_1 and G_2 will be drawn, respectively, and then define the free points. In the following we construct R_1 and R_2 assuming that

$d_i(t_1) \geq 2\lceil \Delta/3 \rceil$. Therefore, we ensure that three of the arms of t_1 lie in R_1 and one of the arms of s_2 lies in R_2 . The case when $d_i(t_1) < \lceil \Delta/3 \rceil$ (i.e., $d_o(s_2) \geq 2\lceil \Delta/3 \rceil$) is symmetric. By slightly modifying the construction¹ we can deal with the case when $\lceil \Delta/3 \rceil \leq d_i(t_1) < 2\lceil \Delta/3 \rceil$. We omit the details due to space constraints.

- A. Determine the leftmost arm h in the sequence a_d, a_r, a_u that is not saturated.
- B. Determine the leftmost arm h' in the sequence c_l, c_u, c_r that is not saturated.
- C. If h and h' lie on the boundary of R , then we compute R_1 and R_2 according to the cases (C₁)–(C₃). Figure 4 shows that the case analysis is exhaustive by examining all possible positions of a and c in R . In Figure 4, the point r corresponds to the center of $t_1 (= s_2)$.

(C₁) If h is parallel to h' and $h = a_r$ (i.e., Column 3 of Row 1 in Figure 4), then we draw a straight line pq such that p, q are two points on h, h' , respectively. Let r and r' be two distinct points on pq such that $\text{dist}(p, r) < \text{dist}(p, r')$. We then draw a line segment $r'z \perp pq$, such that $z \in c_u$. R_1 and R_2 are the rectangles that contain the unsaturated arms, i.e., in this case R_1 (respectively, R_2) is the rectangle with diagonal $r'k_4$ (respectively, $r'c$). Sometimes $R_i, i \in \{1, 2\}$, may not contain the center of the corresponding source and sink. In such a case, we add a dummy copy of the source or sink, e.g., see the gray diamond shapes in Figure 4. Note that while computing the drawing of G_1 inside R_1 inductively, we rotate R_1 by 90° anticlockwise such that the preconditions of the induction hold. Furthermore, we define $f(s_1z) = 0$ such that no unnecessary adjacencies are created in the inductive drawing. Since the addition of dummy copy of a source or sink is straightforward, we do not explicitly describe them in the subsequent cases.

(C₂) If h is parallel to h' and $h \in \{a_d, a_u\}$ (i.e., Column 2 of Row 1 and Columns 1–2 of Row 2 in Figure 4), then we draw a straight line pq such that p, q are two points on h, h' , respectively. Let r be a point on pq . We then draw a line segment $rz \perp pq$, such that $z \in c_l$.

(C₃) Otherwise, $h \perp h'$ (i.e., Columns 1 and 4 of Row 1, Columns 3–4 of Row 2, and Row 3 in Figure 4). Here we draw a polygonal chain p, r, q such that p, q are two points on h, h' , respectively, $pr \perp rq$. We then draw a line segment $rz \perp rq$, such that either $z \in k_2k_3$ (when rq is horizontal), or $z \in k_3k_4$ (when rq is vertical).

An interesting case is shown in Column 2 of Row 3 in Figure 4, where the dummy vertex is placed in the proper interior of the segment qk_3 instead of placing it on q . The reason is to respect the precondition of the induction that s_2 and t_2 should not lie on q . Here we set the free points of the left and up arms of t_2 to 0 to avoid any unnecessary adjacencies in the recursive construction.

¹ Here we ensure that at least two arms of t_1 (respectively, s_2) lie in R_1 (respectively, R_2). At most one arm of t_1 on the boundary of R_1 may coincide with an arm of s_2 on the boundary of R_2 , where we assign the free points depending on $d_i(t_1)$.

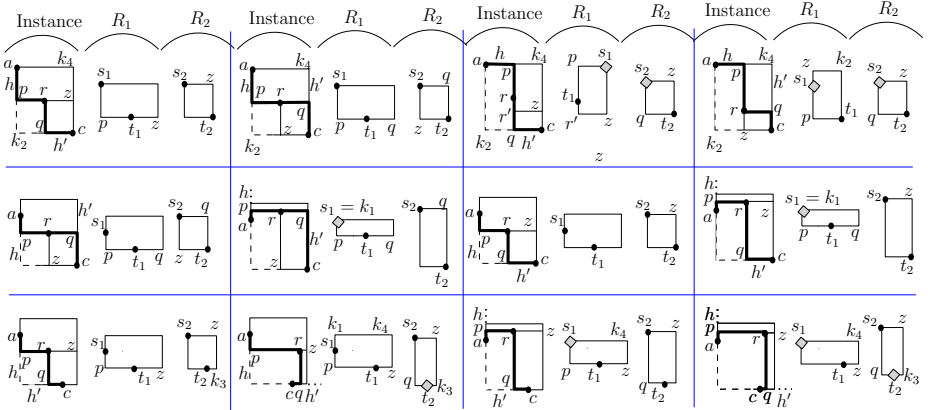


Fig. 4. Computation of R_1 and R_2 when h and h' lie on the boundary of R , and $d_i(t_1) \geq \lceil \Delta/3 \rceil$. (**Case** $a = k_1, c = k_3$): Row 1. (**Case** $a \neq k_1, c = k_3$): Row 2. (**Case** $a = k_1, c \neq k_3$): Symmetric to Row 2. (**Case** $a \neq k_1, c \neq k_3$): Row 3.

- D. Otherwise, at least one of h, h' is in the proper interior of R . In this scenario we consider the following cases depending on the positions of a and c in R .
- (D₁) If $a \neq k_1$ and $c = k_3$ (i.e., Row 1 of Figure 5), then we follow (C₂) or (C₃), depending on whether $h \parallel h'$ or $h \perp h'$, setting $p = r$.
 - (D₂) If $a = k_1$ and $c \neq k_3$, then the computation is symmetric to (D₁).
 - (D₃) Otherwise, both h and h' may lie in the proper interior of R . In this case, if h' lies on the boundary of R , then the computation of R_1 and R_2 is shown in Row 2 of Figure 5. Otherwise, h' lies in the proper interior of R , and the computation of R_1 and R_2 depends on whether $f(c_r) \neq 0$ (i.e., see Row 3 of Figure 5) and $f(c_r) = 0$ (i.e., Row 4 in Figure 5). The details are omitted due to space constraints.

Computation of free points: If R_2 contains an arm of r that does not lie on the boundary of R_1 , then we set $f(r_l) = f(r_r) = f(r_u) = f(r_d) = \lceil \Delta/3 \rceil$. Otherwise, R_2 contains only one arm of r and it is shared with R_1 , i.e., Columns 3–4 of Row 1 in Figure 4, and Row 4 of Figure 5. In such a case, we assign $\lceil \Delta/3 \rceil$ free points to the arms of r that are not shared, and for the shared arm, we assign $d_o(s_2, G_2)$ free points in R_2 and $\lceil \Delta/3 \rceil - d_o(s_2, G_2)$ free points in R_1 .

We now assign the free points of s and t to s_1 and t_2 , respectively, and place $t_1 (= s_2)$ on r . If the edge (s_i, t_i) exists, $i \in \{1, 2\}$, then we draw (s_i, t_i) along h and h' , as shown in bold in Figures 4 and 5. Observe that while drawing (s_i, t_i) , we use some free points of s_1 and t_2 . Therefore, we decrease the free points by one for each arm that helps routing (s_i, t_i) . Since R_1 includes all unsaturated arms of s that lie in R , the number of free points of a in R_1 is at least $d_o(s_1, G_1 \setminus (s_1, t_1))$. According to our assignment of free points, the number of free points of r in R_1 is at least $d_i(t_1, G_1 \setminus (s_1, t_1))$. Therefore, we can inductively draw G_1 inside R_1 , and similarly G_2 inside R_2 .

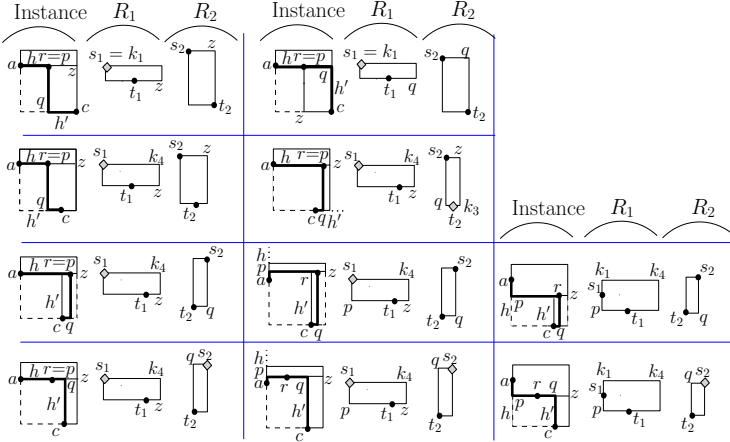


Fig. 5. Computation of R_1 and R_2 when at least one of h, h' lie in the proper interior of R , and $d_i(t_1) \geq \lceil \Delta/3 \rceil$. (**Case $a \neq k_1$ and $c = k_3$**): Row 1. (**Case $a = k_1$ and $c \neq k_3$**): Symmetric to Row 1. (**Case $a \neq k_1$ and $c \neq k_3$**): Rows 2-4.

While computing the drawings of G_1 and G_2 inductively, sometimes we rotated R_1 and R_2 anticlockwise. Therefore, before merging such a drawing, we rotate it clockwise by the same amount. Furthermore, while computing the drawings of G_1 and G_2 , sometimes we added some dummy source and sink. For any arm h of the dummy vertex, that is not a part of the arm of its real copy (e.g., see the illustration in Case (C_1)), we set $f(h) = 0$. Therefore, the merged drawing correctly realizes all adjacencies. Since the drawings inside R_1 and R_2 maintains Property (\star) , the merged drawing also satisfies that property.

Case 2 (Parallel Composition): In this case $s = s_1 = s_2$ and $t = t_1 = t_2$. We first create two copies R_1 and R_2 of R , i.e., $R_1 = a'b'c'd'$ and $R_2 = a''b''c''d''$, where G_1 and G_2 will be drawn, respectively. We now define the free points. Recall that in Case 2 of Lemma 1, we distributed $d_o(s_1)$ among $f(a'_d), f(a'_r)$, and $d_i(t_1)$ among $f(c'_l), f(c'_u)$. Since here we may have at most three arms of s and t inside R , we distribute $d_o(s_1)$ among a'_d, a'_r and a'_u in this order, and similarly, distribute $d_i(t_1)$ among c'_l, c'_u and c'_r in this order. The number of free points in the arms of s_2 and t_2 is determined by the free points of s and t that remains after assigning free points to s_1 and t_1 .

According to our assignment of free points, $d_o(s_1) = f(a'_d) + f(a'_r) + f(a'_u)$. Since $d_o(s) \leq f(a_d) + f(a_r) + f(a_u)$ and $d_o(s) = d_o(s_1) + d_o(s_2)$, the inequality $d_o(s_2) \leq f(a''_d) + f(a''_r) + f(a''_u)$ holds. Similarly, $d_i(t_1) = f(c'_l) + f(c'_u) + f(c'_r)$ and $d_i(t_2) \leq f(c''_l) + f(c''_u) + f(c''_r)$. Therefore, by induction, we can draw G_1 and G_2 inside R_1 and R_2 , respectively.

The idea of merging the drawings of G_1 and G_2 into R is similar to the Case 2 of Lemma 1. Observe that the drawing of G_1 takes consecutive free points from the arms of s (respectively, t) in anticlockwise (respectively, clockwise) order. On the other hand, the drawing of G_2 takes the remaining consecutive free points from the arms of s (respectively, t) in anticlockwise (respectively,

clockwise) order. Therefore, one can merge the two drawings inside R_1 and R_2 avoiding edge crossings inside R . Since the drawings inside R_1 and R_2 maintains Property (\star) , the combined drawing also satisfy that property. \square

Theorem 2. *Every SP graph G has a $(1/3)$ -balanced \dagger -contact representation, but not necessarily a $(1/4 - \epsilon)$ -balanced representation, for any $\epsilon > 0$.*

Proof. The proof for the upper bound is analogous to the proof of Theorem 1. The only difference is that here we use Lemma 2 instead of Lemma 1. The proof for the lower bound is implied by SP graphs with $\Delta \geq 4$ and $\Delta \bmod 4 = 0$. \square

4 Balanced Representations of Planar 3-Trees ($c = 1/2$)

In this section we show that planar 3-trees admit $(1/2)$ -balanced representations. A planar 3-tree G with $n \geq 3$ vertices is a triangulated planar graph such that if $n > 3$, then G contains a vertex whose deletion yields a planar 3-tree with $n - 1$ vertices. Let x, y, z be a cycle in G . By G_{xyz} we denote the subgraph induced by x, y, z and the vertices that lie interior to the cycle. Every planar 3-tree G with $n > 3$ vertices contains a vertex that is the common neighbor of all three outer vertices of G . We call this vertex the *representative vertex* of G . Let p be the representative vertex of G and let a, b, c be the three outer vertices of G , as in Figure 6(a). The subgraphs G_{abp}, G_{bcp} and G_{cap} are planar 3-trees. Let G'_{abp}, G'_{bcp} and G'_{cap} be the subgraphs obtained by deleting the outer edges of G_{abp}, G_{bcp} and G_{cap} , respectively. These subgraphs the three *nested components* of G . By $d(u, G)$, we denote the degree of vertex u in G . Given a planar 3-tree G and a rectangle R , we recursively divide R into three sub-rectangles where the nested components of G will be drawn. We first prove the following lemma.

Lemma 3. *Let G be a planar 3-tree with outer vertices a, b, c and representative vertex p , and let G' be the graph obtained from G by deleting the outer edges of G . Let $R = k_1k_2k_3k_4$ be an axis-parallel rectangle such that a, b, c lie on k_1k_2, k_2k_3 and k_4 , respectively. Assume that the number of free points of each arm of a, b, c is prespecified. If the inequalities $d(a, G') \leq f(a_d) + f(a_u), d(b, G') \leq f(b_l) + f(b_r)$, and $d(c, G') \leq f(c_l) + f(c_d)$ hold, then G' admits a $(1/2)$ -balanced \dagger -contact representation Γ in R satisfying the following property.*

(\star) *The number of contact points at each arm incident to a, b and c in Γ is at most the number of free points specified for that arm as input.*

Proof. We employ an induction on the number of vertices of G . If $n = 3$, then G' consists of only three isolated vertices a, b and c that lie on k_1k_2, k_2k_3 and k_4 , respectively. It is now straightforward to verify Property (\star) . Hence we assume that $n \geq 4$ and the lemma holds for all G with smaller than n vertices. We now consider the case when G has n vertices.

We first compute three sub-rectangles R_1, R_2 and R_3 , where G'_{abp}, G'_{bcp} and G'_{cap} will be drawn, respectively. Define h to be either a_u or a_d depending on whether $d(a, G'_{abp}) \geq f(a_d)$ or not. Similarly, define h' to be either b_r or b_l

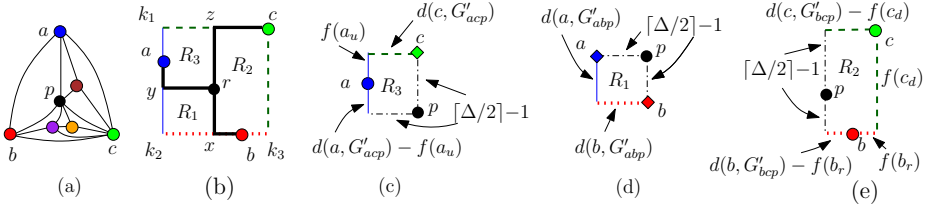


Fig. 6. (a) A plane 3-tree. (b) Computation of R_1, R_2 and R_3 , where $d(a, G'_{abp}) < f(a_d)$, $d(b, G'_{abp}) < f(b_l)$, and $d(c, G'_{bcp}) \geq f(c_d)$. (c)–(e) Assignment of free points.

depending on whether $d(b, G'_{abp}) \geq f(b_l)$ or not. Since the inequalities $f(a_d) + f(a_u) \geq d(a, G') \geq 1$ and $f(b_l) + f(b_r) \geq d(b, G') \geq 1$ hold, h and h' must be unsaturated. Let x and y be two points on h' and h , respectively, as shown in Figure 6(b). Draw two line segments $xr \perp h'$ and $yr \perp h$ such that they meet at point r . Define h'' to be the arm c_l or c_d depending on whether $d(c, G'_{bcp}) \geq f(c_d)$ or not. Since $f(c_l) + f(c_d) \geq d(c, G') \geq 1$, h'' must be unsaturated. We draw an orthogonal line segment rz such that $z \in h''$. Observe that rx, ry and rz divides R into three sub-rectangles R_1, R_2 and R_3 , i.e., the sub-rectangles that contain corners k_2, k_3 and k_1 , respectively.

We place the vertex p on r , draw the edges (a, p) , (b, p) and (c, p) along ry, rx and rz , respectively, and then assign $\lceil \Delta/2 \rceil - 1$ free points at each arm of r . To define the free points of the other arms of R_i , we distribute the free points of a, b and c as follows. We distribute $d(a, G'_{abp})$ among a_d and a_u (in R_1), and $d(a, G'_{acp})$ among a_u and a_d (in R_3). We then distribute $d(b, G'_{abp})$ among b_l and b_r (in R_1), and $d(b, G'_{bcp})$ among b_r and b_l (in R_2). Finally, we distribute $d(c, G'_{bcp})$ among c_d and c_l (in R_2), and $d(c, G'_{acp})$ among c_l and c_d (in R_3).

Let G'_i be the nested component of G that corresponds to R_i , $i \in \{1, 2, 3\}$. Observe that some outer vertices of G'_i may not lie on R_i . Hence we cannot directly apply the induction hypothesis. Hence for each vertex a, b or c that does not lie on the boundary of R_i but belongs to G'_i , we add a dummy copy of that vertex at x, y or z , respectively. Furthermore, for each arm h of the dummy copy that is not a part of any arm of its real copy, we set $f(h) = 0$, e.g., $f(c_d) = 0$ in Figure 6(c). Consequently, the recursively computed drawings do not create any unnecessary adjacencies. Observe that each R_i now meets the preconditions of the induction, as shown in Figures 6(c)–(e), and hence we inductively draw G'_i inside R_i . To apply the induction, we need to be careful of the vertex that play the role of k_4 , i.e., the corner having exactly two arms inside the rectangle that are perpendicular to each other, e.g., the position of p in Figures 6(c)–(d), and the position of c in Figure 6(e). Each R_i contains exactly one of r and c at one of its four corners, which plays the role of k_4 in R_i . Since the smaller drawings satisfy Property $(*)$, the final drawing satisfies Property $(*)$. \square

Theorem 3. *Every planar 3-tree G admits a $(1/2)$ -balanced \dagger -contact representation, but not necessarily a $(1/3)$ -balanced representation.*

Proof. Let a, b, c be the outer vertices of G , and let $R = k_1k_2k_3k_4$ be an axis-parallel rectangle. Place a, b, c on k_1k_2, k_2k_3 and k_4 , respectively, draw the outer edges of G along the boundary of R , and finally, assign $\lceil \Delta/2 \rceil - 1$ free points at each arm of a, b and c . Let G' be the graph obtained by removing the outer edges of G . By Lemma 3, G' has a $(1/2)$ -balanced \dagger -contact representation in R . The lower bound that $c > 1/3$ is implied by the graph K_4 . \square

5 Conclusion

We have proved that 2-trees (respectively, planar 3-trees) admit c -balanced \dagger -contact representations, where $1/4 \leq c \leq 1/3$ (respectively, $1/3 < c \leq 1/2$). A natural open question is to find tight bounds on c . Although our representations for planar 3-trees preserve input embedding, the representations for 2-trees do not have this property. Thus it would be interesting to examine whether there exist algorithms for $(1/3)$ -balanced representations of 2-trees that preserve input embedding. Another intriguing open question is to characterize planar graphs that admit c -balanced \dagger -contact representations, for small fixed values c .

References

1. de Fraysseix, H., de Mendez, P.O., Rosenstiehl, P.: On triangle contact graphs. *Combinatorics, Probability and Computing* 3(2), 233–246 (1994)
2. Czyzowicz, J., Kranakis, E., Urrutia, J.: A simple proof of the representation of bipartite planar graphs as the contact graphs of orthogonal straight line segments. *Information Processing Letters* 66(3), 125–126 (1998)
3. Kobourov, S.G., Ueckerdt, T., Verbeek, K.: Combinatorial and geometric properties of planar Laman graphs. In: Khanna, S. (ed.) *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1668–1678. SIAM (2013)
4. Biedl, T.C., Kaufmann, M.: Area-efficient static and incremental graph drawings. In: Burkard, R.E., Woeginger, G.J. (eds.) *ESA 1997. LNCS*, vol. 1284, pp. 37–52. Springer, Heidelberg (1997)
5. Kant, G.: Drawing planar graphs using the canonical ordering. *Algorithmica* 16(1), 4–32 (1996)
6. Zhou, X., Nishizeki, T.: Orthogonal drawings of series-parallel graphs with minimum bends. *SIAM Journal on Discrete Mathematics* 22(4), 1570–1604 (2008)
7. Tayu, S., Nomura, K., Ueno, S.: On the two-dimensional orthogonal drawing of series-parallel graphs. *Discrete Applied Mathematics* 157(8), 1885–1895 (2009)
8. Dujmović, V., Suderman, M., Wood, D.R.: Graph drawings with few slopes. *Computational Geometry* 38(3), 181–193 (2007)
9. Lenhart, W., Liotta, G., Mondal, D., Nishat, R.I.: Planar and plane slope number of partial 2-trees. In: Wismath, S., Wolff, A. (eds.) *GD 2013. LNCS*, vol. 8242, pp. 412–423. Springer, Heidelberg (2013)
10. Jelínek, V., Jelínková, E., Kratochvíl, J., Lidický, B., Tesar, M., Vyskocil, T.: The planar slope number of planar partial 3-trees of bounded degree. *Graphs and Combinatorics* 29(4), 981–1005 (2013)
11. Keszegh, B., Pach, J., Pálvölgyi, D.: Drawing planar graphs of bounded degree with few slopes. *SIAM Journal on Discrete Mathematics* 27(2), 1171–1183 (2013)

Strongly-Connected Outerplanar Graphs with Proper Touching Triangle Representations

J. Joseph Fowler

Department of Computer Science, University of Arizona, Tucson, AZ, USA
fowler@email.arizona.edu

Abstract. A *proper touching triangle representation* \mathcal{R} of an n -vertex planar graph consists of a triangle divided into n non-overlapping triangles. A pair of triangles are considered to be adjacent if they share a partial side of positive length. Each triangle in \mathcal{R} represents a vertex, while each pair of adjacent triangles represents an edge in the planar graph. We consider the problem of determining when a proper touching triangle representation exists for a *strongly-connected outerplanar graph*, which is biconnected and after the removal of all degree-2 vertices and outeredges, the resulting *connected* subgraph only has chord edges (w.r.t. the original graph). We show that such a graph has a proper representation if and only if the graph has at most two *internal faces* (i.e., faces with no outeredges).

1 Introduction

Although the node-link model has been the traditional form of drawing a planar graph $G(V, E)$, many application areas demand alternate models of representing graphs, such as polygon *edge-contact representations*. Here vertices are represented by simple polygons and edges are represented by adjacent polygons that have at least a partial side in common. As pointed out by de Fraysseix *et al.* [2], one can easily find such a representation of any planar graph with non-convex polygons with complexity as high as $|V| - 1$, where much area is unused leading to many gaps and holes within the representation. Recently, convex hexagons have been shown to always be sufficient in producing hole-free representations [1], although, 6-sided polygons are sometimes necessary. The problem thus arises in determining which classes of planar graphs can be represented by polygons with fewer than six sides.

In this context, we focus on the case of minimal polygonal complexity, where all the representing polygons are triangles. Specifically, an n -vertex *touching triangle graph* (TTG) has a representation \mathcal{R} where each vertex is represented by one of n non-overlapping triangles and each edge is represented by a pair of adjacent triangles in \mathcal{R} . Again, triangles are only considered to be adjacent if they share at least a partial side of positive length. Thus, pairs of triangles having only point contacts are not considered to be adjacent, and hence, do not represent edges.

The most natural looking edge-contact representations have triangular boundaries where their interiors contains no gaps or holes. If this is the case, then \mathcal{R} is a *proper* TTG representation, and the graph is a proper TTG. Visually, \mathcal{R} can be thought of as a triangle that has been subdivided into n non-overlapping triangles, where adjacent

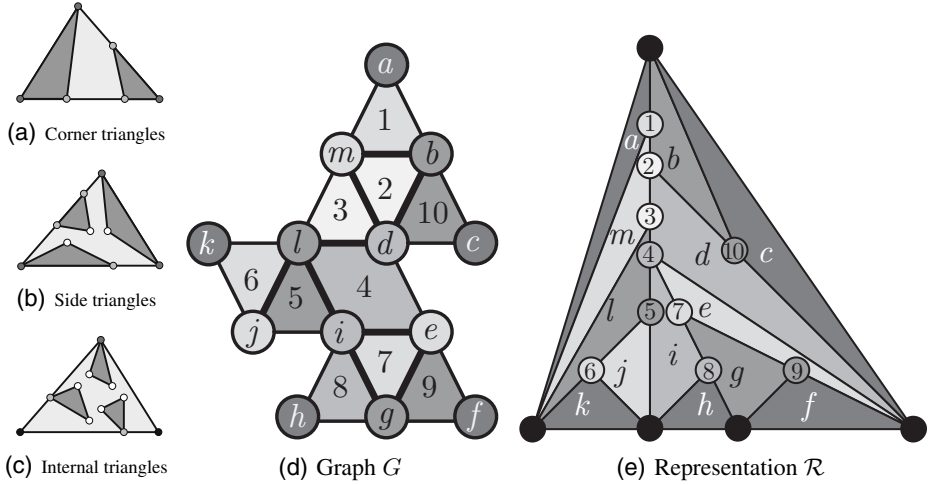


Fig. 1. (a–c) Three types of representing triangles; (d) a strongly-connected outerplanar graph G with two internal faces: 2 and 7; and (e) a proper TTG representation \mathcal{R} of G

triangles do not necessarily share entire sides as they do in a triangulation. Not all planar graphs are TTGs [3], let alone proper. While it was also shown in [3] that all biconnected outerplanar graphs have hole-free TTG representations, their boundaries are not necessarily triangular, and hence, are not necessarily proper. This raises the question as to which outerplanar graphs have proper TTG representations.

A proper outerplanar TTG has several restrictions. Degree-1 vertices can only be represented by *corner triangles* of \mathcal{R} with two edges along the boundary T of \mathcal{R} , while degree-2 vertices can also be represented by *side triangles* of \mathcal{R} with one side along T . All other vertices are represented by *internal triangles* of \mathcal{R} ; cf. Figs. 1(a)–1(c).

A biconnected outerplanar graph G is *strongly-connected* if after the removal of all degree-2 vertices and outeredges, the resulting *connected* subgraph only has chord edges (w.r.t. G) as in Fig. 1(d). Such graphs are not necessarily maximal. We characterize this graph class in terms of *internal faces* (i.e. faces with no outeredges) as follows:

- (1) First, we construct proper TTG representations for strongly-connected outerplanar graphs, as in Fig. 1(e), using a *chord-to-endpoint assignment* that pairs each chord (except for one) with a distinct vertex that is also an endpoint of the chord.
- (2) Second, we show that having at most two internal faces is sufficient when the graph is strongly-connected, since a chord-to-endpoint assignment exists in this case.
- (3) Third, we finish our characterization by proving that having at most two internal faces is also necessary in order for G to have a proper TTG representation.

To the best of our knowledge, the only other results specifically for proper TTGs are in [4], where a fixed-parameter tractable decision algorithm for 3-connected planar max-degree- Δ graphs is described, and where it is shown that planar 3-connected cubic graphs are proper TTGs.

2 Proper Strongly-Connected Outerplanar TTGs

Let G be a strongly-connected outerplanar graph. A *strong reversed peeling order* σ is an ordering of the inner faces F_1, \dots, F_q of G such that the subgraph $G_i = F_1 \cup \dots \cup F_i$ is also strongly-connected for each $i \in [1 .. q]$. We index the chords $\mathcal{C} = \{c_2, \dots, c_q\}$ of G by σ such that c_i is the chord of face F_i that becomes an outeredge of G_{i-1} when F_i is “peeled” from G_i . Thus, c_i is common to $F_{i'}$ and F_i for some $i' < i$. Finally, a *chord-to-endpoint assignment* $\tau : \mathcal{C}' \rightarrow V'$ of a strong reversed peeling order σ assigns the chords $\mathcal{C}' = \{c_3, \dots, c_q\} = \mathcal{C} \setminus \{c_2\}$ to a subset of their endpoints $V' = \{v_3, \dots, v_q\}$ such that $\tau(c_i) = v_i$ is an endpoint of c_i for each $i \in [3 .. q]$ where $v_i \neq v_j$ if $i \neq j$.

Claim 1. *If G is a strongly-connected outerplanar graph, then G has a strong reversed peeling order σ .*

Proof. Faces F_1 and F_2 of σ can be any pair of adjacent faces. For $i \in [2 .. q-1]$, assume that subgraph $G_i = F_1 \cup \dots \cup F_i$, has the connected *chord subgraph* $H_i = c_2 \cup \dots \cup c_i$, where c_i is the chord of face F_i that was added to G_{i-1} . At least one outeredge of G_i is a chord in G . Otherwise, the outerface of G_i , which is a separating cycle C in G , would have a cut-vertex in G —all the cut edges in G are chords, none of which can be in C —violating the biconnectivity of G . Since G is strongly-connected, every chord c in the outerface of G_i must be incident to some chord of H_i . Thus, the next face F_{i+1} in σ can be any remaining face whose chord c_{i+1} is an outeredge of G_i . \square

Lemma 2. *If G is a strongly-connected outerplanar graph for which there exists a chord-to-endpoint assignment τ , then G has a proper touching triangle representation.*

Proof. Let σ be the strong reversed peeling order of the faces F_1, \dots, F_q of G given by the order of chords as assigned by τ . We apply induction on k and assume that we have a proper TTG representation \mathcal{R}_k for G_k , and show how to modify \mathcal{R}_k to obtain \mathcal{R}_{k+1} for $G_{k+1} = G_k \cup F_{k+1}$. When G_1 has a single face F_1 with the edge (u, v) connected by the chain of vertices x_1, \dots, x_i , Fig. 2(a) gives a proper TTG representation \mathcal{R}_1 for G_1 . Next, when G_2 has two faces with the common chord (u, v) (where y_1, \dots, y_j forms a chain in F_2), Fig. 2(b) gives a proper TTG representation \mathcal{R}_2 for G_2 where the triangle Δu representing u in \mathcal{R}_1 was subdivided into a total of $j + 1$ triangles.

For $k \in [2 .. q-1]$, we also assume by induction on k that triangle Δv_k representing $\tau(c_k) = v_k$ is a side triangle in \mathcal{R}_k . This holds in the base case of $k = 2$, since \mathcal{R}_2 in

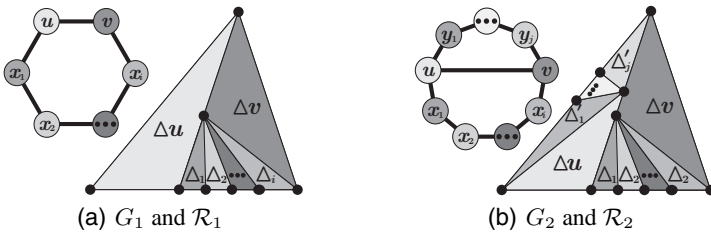


Fig. 2. Proper TTG representations of strongly-connected outerplanar graphs with 1 or 2 faces

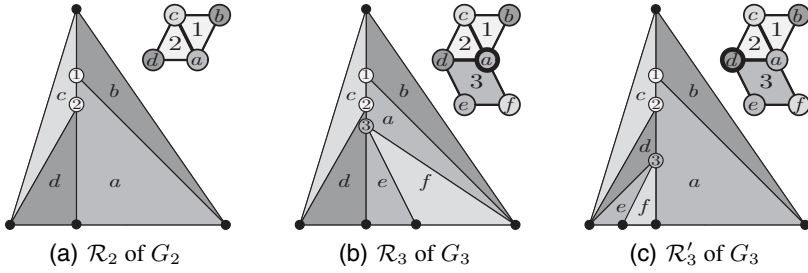


Fig. 3. For chord $c_3 = (a, d)$, either triangle Δa or Δd in \mathcal{R}_2 in (a) is divided into triangles $\Delta\tau(c_3)$, Δe , Δf to form \mathcal{R}_3 if $\tau(c_3) = a$ as in (b) or \mathcal{R}'_3 if $\tau(c_3) = d$ as in (c)

Fig. 2(b) only has side triangles. This allows us to subdivide Δv_k in \mathcal{R}_k a total of j_k times to obtain \mathcal{R}_{k+1} in which an internal triangle now represents v_k and j_k new side triangles represent the j_k degree-2 vertices in F_k that were added to G_k to form G_{k+1} . Figure 3 illustrates this process. The chord $c_3 = (a, d)$ in G_3 either has $\tau(c_3) = a$ or $\tau(c_3) = d$. This results in \mathcal{R}_3 in Fig. 3(b) (or in \mathcal{R}'_3 in Fig. 3(c)) after dividing the side triangle Δa (or Δd) in \mathcal{R}_2 of G_2 in Fig. 3(a) into the internal triangle Δa (or Δd) and the two side triangles Δe and Δf . As a consequence, each chord endpoint v_k has its representing triangle Δv_k subdivided once, at which point Δv_k is an internal triangle in $\mathcal{R}_{k'}$ for $k' > k$. However, this is not a problem in maintaining our inductive hypothesis for $k + 1$ since τ assigns each chord to a distinct vertex in G . \square

Lemma 3. *If G is a strongly-connected outerplanar graph with at most two internal faces, then G is a proper touching triangle graph.*

Proof. We construct a chord-to-endpoint assignment τ , where Lemma 2 then implies that G is a proper TTG. If G has no internal faces, its connected chord subgraph H is acyclic, and hence, a tree. Let σ be a strong reversed peeling order of faces F_1, \dots, F_q for G given by Claim 1. Each face F_i was picked so that its chord c_i is a leaf edge in the subtree of chords $H_i = c_2 \cup \dots \cup c_i$ of G_i . Thus, we can assign $\tau(c_i) = u_i$, where u_i is the endpoint of c_i that is a leaf node in H_i , for $i \in [3 .. q]$. Both endpoints of the first chord c_2 are left unassigned by τ .

If G has one internal face F , we apply Claim 1 and assign chords as before with the following exceptions: We set $F_1 = F$ and faces F_2, \dots, F_j as the adjacent faces of F_1 , where face F_{i+1} is incident to face F_i for $i \in \{2 .. j - 1\}$. Since chord c_j forms the cycle C (edges of F_1) when added to H_j , we can set $\tau(c_j)$ to be the common endpoint of the chords c_j and c_2 , which leaves one endpoint of c_2 unassigned.

Lastly, when G has two internal faces, we apply Claim 1 as follows: We set F_1 and F_k in σ to be the two internal faces in G such that k is minimal. The chords of G_k contain a path p connecting F_1 to F_k . For τ , we assign each chord of p to the endpoint first encountered along p so that c_2 and c_k each have exactly one assigned endpoint. To σ we add each remaining face F_i adjacent first to F_1 (and then to F_k) starting from the endpoints of p in cyclic order along each respective face. For τ , we assign chord c_i of each cycle to its newly added endpoint in H_i —except for the last chord, call it c' , in F_1

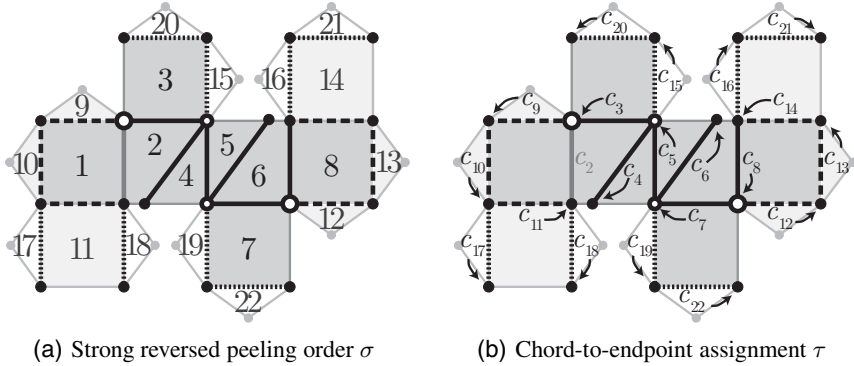


Fig. 4. Example of determining σ and τ for strongly-connected outerplanar graph G with two internal faces F_1 and F_8 . Subgraph G_8 (dark gray faces) is the minimal strongly-connected subgraph containing both F_1 and F_8 , whose chord subgraph H_8 (solid chords) is a caterpillar. Subgraph G_{14} (light/dark gray faces) has faces F_9, \dots, F_{11} and F_{12}, \dots, F_{14} added in cyclic order along F_1 and F_8 , resp., each starting from an endpoint of path p (white vertices). The chords of path p are assigned the endpoints first encountered along p from F_1 to F_8 . The dashed chords of F_1 and F_8 are assigned endpoints next along each cycle (starting from endpoints of p). Chord c_i for $i \in \{4, 6, 15, \dots, 22\}$ is assigned its endpoint that was not in the chord subgraph H_{i-1} .

(or in F_k). Given the greedy assignment of chords along p , c' has an available endpoint in common with c_2 in F_1 (or c_k in F_k). Each remaining face can be added to σ and have its chord assigned by τ as before. Figure 4 illustrates this procedure. \square

For the next lemma, we consider the *representing dual graph* $\mathcal{G}_{\mathcal{R}}$ (the graph formed by the representation \mathcal{R}) of a proper TTG G . With respect to the triangular boundary T of a proper TTG representation \mathcal{R} , each vertex or edge of $\mathcal{G}_{\mathcal{R}}$ (common to one or more representing triangles) is either *external* if along the boundary T or *internal* if inside T . Likewise, we term the faces of $\mathcal{G}_{\mathcal{R}}$ as being either *external faces* if they correspond to corner or side triangles or *internal faces*, otherwise. We have the following relationships between G and $\mathcal{G}_{\mathcal{R}}$: (1) each vertex v in G corresponds to a bounded triangular face F_v in $\mathcal{G}_{\mathcal{R}}$ and (2) each bounded face f in G corresponds to an internal vertex v_f in $\mathcal{G}_{\mathcal{R}}$.

Clearly, the angle of a vertex v_f in $\mathcal{G}_{\mathcal{R}}$ along the face F_v in $\mathcal{G}_{\mathcal{R}}$ can be at most 180° . If the angle is less than 180° , then v_f is a corner of the triangular face F_v . Each internal vertex v_f in $\mathcal{G}_{\mathcal{R}}$ has at most one 180° angle since $deg(v_f) > 2$ in $\mathcal{G}_{\mathcal{R}}$. For example, the angle of vertex v_1 in \mathcal{R} in Fig. 1(e) is 180° for face F_b , but not for faces F_a and F_m . Face F_v in $\mathcal{G}_{\mathcal{R}}$ representing v has exactly three vertices (each with an angle less than 180°) if either (i) F_v is an external face where $deg(v) = 2$ or (ii) F_v is an internal face where $deg(v) = 3$. Otherwise, F_v has at least $ch(v) = \max\{deg(v) - 3, 0\}$ vertices in $\mathcal{G}_{\mathcal{R}}$ whose internal angles are 180° along the boundary of F_v . These correspond to $ch(v)$ faces in G that are incident to v .

We denote $ch(v)$ as the *charge* of v since each incident face F in G can *dissipate* at most one charge from v . This is done by having the internal angle for v_f be 180° along the face F_v representing v in $\mathcal{G}_{\mathcal{R}}$. However, face F can dissipate at most one charge from an incident vertex, since v_f has at most one 180° angle. For example, vertex d in

Fig. 1(d) has charge $ch(d) = 2$ dissipated by faces F_3 and F_{10} , where the corresponding internal vertices v_3 and v_{10} have 180° angles for face F_d in Fig. 1(e).

Thus, in order for G to be a TTG, there must exist a *discharge function*, $\pi : \mathcal{F}' \rightarrow V$ that assigns a subset faces $\mathcal{F}' \subseteq \mathcal{F}$ (the faces of G) to incident vertices to fully dissipate the *total charge* $ch(G) = \sum_{v \in V(G)} ch(v)$ of G . Hence, G cannot be a proper TTG graph if $av(G) = q - ch(G) < 0$, where $q = |\mathcal{F}|$ and $av(G)$ is the *availability* of G .

Lemma 4. *If G is a strongly-connected outerplanar graph with more than two internal faces, then G cannot be a proper touching triangle graph.*

Proof. We apply Claim 1 to get a strong reversed peeling order σ of the faces F_1, \dots, F_q of G , where the subgraph $G_i = F_1 \cup \dots \cup F_i$ is strongly-connected. If G_i does not contain any internal faces, then H_i is a tree. When peeling face F_i from G_i to obtain G_{i-1} , chord c_i cannot have both of its endpoints u_i and v_i in H_{i-1} . Otherwise, c_i would form a cycle in H_i . Thus, one endpoint $v_i \notin H_{i-1}$ of c_i has $deg(v_i) = 2$ in G_{i-1} . While both degrees of u_i and v_i increased by 1 in going from G_{i-1} to G_i , only $deg(u_i) > 3$ in G_i , while $deg(v_i) = 3$ in G_i . Thus, the total charge $ch(G_i) = ch(G_{i-1}) + 1$ increases by one, so that the availability $av(G_i) = av(G_{i-1})$ remains constant.

If G_i contains a new internal face C that G_{i-1} does not, then both endpoints u_i and v_i of c_i must be in H_{i-1} in order for c_i to form the new cycle C in H_i . Hence, both $deg(u_i) > 3$ and $deg(v_i) > 3$ in G_i , so that $ch(G_i) = ch(G_{i-1}) + 2$, and as a result, $av(G_i) = av(G_{i-1}) - 1$ where the availability decreases by one.

Initially the availability is at most 2, where G_2 has maximum degree 3 with the two faces F_1 and F_2 so that $av(G_2) = 2$. Consequently, G can have at most two internal faces, before the availability drops below 0, preventing it from being a TTG. \square

We conclude by combining Lemmas 3 and 4 to give the main theorem of the paper.

Theorem 5. *A strongly-connected outerplanar graph G has a proper touching triangle representation if and only if G has at most two internal faces.*

Acknowledgments. We thank our colleagues for helpful insights: Jawaherul Alam, Michael Kaufman, Stephen Kobourov, Martin Nöllenburg, Ignaz Rutter, Alexander Wolff, and many others.

References

1. Duncan, C.A., Gansner, E.R., Hu, Y.F., Kaufmann, M., Kobourov, S.G.: Optimal polygonal representation of planar graphs. *Algorithmica* 63(3), 672–691 (2012)
2. de Fraysseix, H., de Mendez, P.O., Rosenstiehl, P.: On triangle contact graphs. *Combinatorics, Probability and Computing* 3, 233–246 (1994)
3. Gansner, E.R., Hu, Y., Kobourov, S.G.: On touching triangle graphs. In: Brandes, U., Cornelsen, S. (eds.) *GD 2010. LNCS*, vol. 6502, pp. 250–261. Springer, Heidelberg (2011)
4. Kobourov, S.G., Mondal, D., Nishat, R.I.: Touching triangle representations for 3-connected planar graphs. In: Didimo, W., Patrignani, M. (eds.) *GD 2012. LNCS*, vol. 7704, pp. 199–210. Springer, Heidelberg (2013)

Achieving Good Angular Resolution in 3D Arc Diagrams

Michael T. Goodrich and Paweł Pszozna

Dept. of Computer Science
University of California, Irvine

Abstract. We study a three-dimensional analogue to the well-known graph visualization approach known as *arc diagrams*. We provide several algorithms that achieve good angular resolution for 3D arc diagrams, even for cases when the arcs must project to a given 2D straight-line drawing of the input graph. Our methods make use of various graph coloring algorithms, including an algorithm for a new coloring problem, which we call *localized edge coloring*.

1 Introduction

An *arc diagram* is a two-dimensional graph drawing where the vertices of a graph, G , are placed on a one-dimensional curve (typically a straight line) and the edges of G are drawn as circular arcs that may go outside that curve (e.g., see [1,2,6,8,19,20,23]). By way of analogy, we define a *three-dimensional arc diagram* to be a drawing where the vertices of a graph, G , are placed on a two-dimensional surface (such as a sphere or plane) and the edges of G are drawn as circular arcs that may go outside that surface. (See Fig. 1.) This 3D drawing paradigm is used, for example, to draw geographic networks or flight networks (e.g., see [3]).

In this paper, we are interested in the angular resolution of 3D arc diagrams, that is, the smallest angle determined by the tangents at a vertex, v , to two

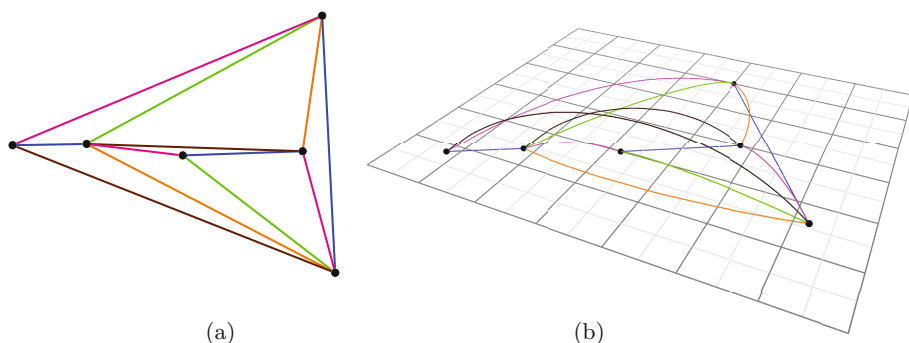


Fig. 1. A graph rendered (a) as a straight-line drawing and (b) as a 3D arc diagram

arcs incident to v in such a drawing. Specifically, we provide algorithms for achieving good angular resolution in 3D arc diagrams where the (*base*) surface that contains the vertices for the graph, G , is a sphere or a plane. Moreover, for the 3D arc diagrams that we consider in this paper, we assume that all the edges of G are drawn to protrude out of only one side of the base surface.

1.1 Previous Related Results

The term “*arc diagram*” was defined in 2002 by Wattenberg [23], but the drawing paradigm actually can be traced back to the 1960’s, including work by Saaty [20] and Nicholson [19]. Also, earlier work by Brandes [2] explores symmetry in arc diagrams, earlier work by Cimikowski and Shope [6] explores heuristics for minimizing the number of arc crossings, and earlier work by Djidjev and Vrt’o [8] explores lower bounds for the crossing numbers of such drawings. Most recently, Angelini *et al.* [1] show that there is a universal set of $O(n)$ points on a parabola that allows any planar graph to be drawn as a planar arc diagram.

In terms of previous work on arc diagrams for optimizing the angular resolution of such drawings, Duncan *et al.* [11] give a complete characterization of which regular graphs can be drawn as arc diagrams with vertices placed on a circle and perfect angular resolution, using a drawing style inspired by the artist, Mark Lombardi, where edges are drawn using circular arcs so as to achieve good angular resolution. With respect to a lower bound for this drawing style, Cheng *et al.* [5] give a planar graph with bounded degree, d , that requires exponential area if it is drawn as a plane graph with circular-arc edges and angular resolution $\Omega(1/d)$. Even so, it is possible to draw any planar graph as a plane graph with poly-line or poly-circular edges to achieve polynomial area and $\Omega(1/d)$ angular resolution, based on results by a number of authors (e.g., see Brandes *et al.* [4], Cheng *et al.* [5], Duncan *et al.* [9,11], Garg and Tamassia [15], Goodrich and Wagner [17], and Gutwenger and Mutzel [18]).

In addition, several researchers have investigated how to achieve good angular resolution for various straight-line drawings of graphs. Duncan *et al.* [10] show that one can draw an ordered tree of degree d as a straight-line planar drawing with angular resolution $\Omega(1/d)$. Formann *et al.* [14] show that any graph of degree d has a straight-line drawing with polynomial area and angular resolution $\Omega(1/d^2)$, and this can be improved to be $\Omega(1/d)$ for planar graphs, albeit with a drawing that may not be planar.

We are not familiar with any previous work on achieving good angular resolution for 3D arc diagrams, but there is previous related work on other types of 3D drawings [7]. For instance, Brandes *et al.* [3] show that one can achieve $\Omega(1/d)$ angular resolution for 3D geometric network drawings, but their edges are curvilinear splines, rather than simple circular arcs. Garg *et al.* [16] study 3D straight-line drawings so as to satisfy various resolution criteria, but they do not constrain vertices to belong to a 2D surface. In addition, Eppstein *et al.* [12] provide an algorithm for achieving optimal angular resolution in 3D drawings of low-degree graphs using poly-line edges.

1.2 Our Results

In this paper, we give several algorithms for achieving good angular resolution for 3D arc diagrams. In particular, we show the following for a graph, G , with maximum degree, d :

- We can draw G as a 3D arc diagram with an angular resolution of $\Omega(1/d)$ ($\Omega(1/d^{1/2})$ if G is planar) using straight-line segments and vertices placed on a sphere.
- We can draw G as a 3D arc diagram with an angular resolution of $\Omega(1/d)$ using circular arcs that project perpendicularly to a given straight-line drawing for G in a base plane, no matter how poor the angular resolution of that projected drawing.
- If a straight-line 2D drawing of G already has an angular resolution of $\Omega(1/d)$ in a base plane, \mathcal{P} , then we can draw G as a 3D arc diagram with an angular resolution of $\Omega(1/d^{1/2})$ using circular arcs that project perpendicularly to the given drawing of G in \mathcal{P} .
- Given any 2D straight-line drawing of G in a base plane, \mathcal{P} , we can draw G as a 3D arc diagram with an angular resolution of $\Omega(1/d^{1/2})$ using circular arcs that project to the edges of the drawing of G in \mathcal{P} , with each arc possibly using a different projection direction.

Our algorithms make use of various graph coloring methods, including an algorithm for a new coloring problem, which we call *localized edge coloring*.

Note that $O(1/d^{1/2})$ is an upper bound on the resolution of a 3D arc drawing of G , as maximizing the smallest angle between two edges around a vertex, v , is equivalent to maximizing smallest distance between intersections of a unit sphere centered at v , and lines tangent to edges incident to v , which is known as the *Tammes problem* [21]. The $O(1/d^{1/2})$ upper bound is due to Fejes Tóth [13].

2 Preliminaries

In this section, we provide formal definitions of two notions of 3D arc diagrams.

We extend the notion of arc diagrams and define *3D arc diagram drawings* of a graph, G , to be 3D drawings that meet the following criteria:

- (1) nodes (vertices) are placed on a single (*base*) sphere or plane
- (2) each edge, e , is drawn as a *circular arc*, i.e., a contiguous subset of a circle
- (3) all edges lie entirely on one side of the base sphere or plane.

In addition, if the base surface is a plane, \mathcal{P}_1 , then each circular edge, e , which belongs to a plane, \mathcal{P}_2 , forms the same angle, $\alpha_e \leq \pi/2$, in \mathcal{P}_2 , at its two endpoints. Moreover, in this case, each edge projects (perpendicularly) to a straight line segment in \mathcal{P}_1 . An example of such an arc is shown in Fig. 2a.

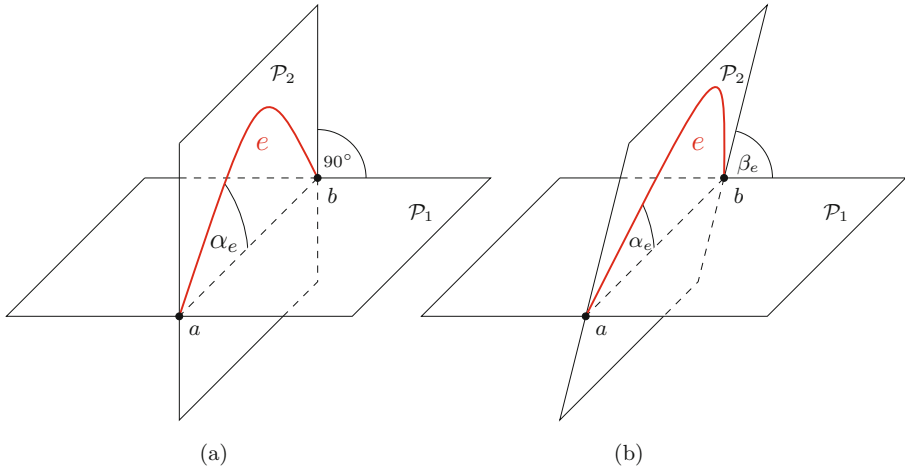


Fig. 2. Edge $e = (a, b)$ drawn as (a) circular arc with angle α_e ; (b) slanted circular arc with angles (α_e, β_e) .

For 3D arc diagrams restricted to use a base plane, \mathcal{P}_1 (rather than a sphere), by modifying the second condition, we obtain a definition of *slanted 3D arc diagram drawings*.

- (2') each edge e is a *circular arc* that lies on a plane, \mathcal{P}_2 , that contains both endpoints of e and forms an angle, $\beta_e < \pi/2$, with the base plane, \mathcal{P}_1 ; the edge, e , forms the same angle, $\alpha_e \leq \pi/2$, in \mathcal{P}_2 , at its two endpoints.

Note that in this case each circular edge, e , joining vertices a and b , in a slanted 3D arc diagram, projects to a straight line segment, $L = ab$, in the base plane, \mathcal{P}_1 , using a direction perpendicular to L in \mathcal{P}_2 . Still, a perpendicular projection of the drawing onto the base plane, \mathcal{P}_1 , is not necessarily a straight-line drawing of G and may not even be planar. For an example, see Fig. 2b.

3 Localized Edge Coloring

Recall that a *vertex coloring* of a graph is an assignment of colors to vertices so that every vertex is given a color different from those of its adjacent vertices, and an *edge coloring* is an assignment of colors to a graph's edges so that every edge is given a color different from its incident edges. A well-known greedy algorithm can color any graph with maximum degree, d , using $d + 1$ colors, and Vizing's theorem [22] states that edges of an undirected graph G can similarly be colored with $d + 1$ colors, as well.

Assuming we are given an undirected graph G together with its combinatorial embedding on a plane (i.e., the order of edges around each vertex, which is also known as a *rotation system*), we introduce a localized notion of an edge coloring, which will be useful for some of our results regarding 3D arc diagrams. Given an

even integer parameter, L , we define an L -localized edge coloring to be an edge coloring that satisfies the following condition:

Suppose an edge $e = (u, v)$ has color c , and let $(l_1, l_2, \dots, l_i = e, \dots, l_k)$ be a clockwise ordering of edges incident to u . Then none of the edges $l_{i-L/2}, l_{i-L/2+1}, \dots, l_{i-1}, l_{i+1}, \dots, l_{i+L/2}$, that is, the $L/2$ edges before e and $L/2$ edges after e in the ordering, has color c . (Note that, by symmetry, the same goes for edges around v .)

Thus, a valid d -localized edge coloring is also a valid classical edge coloring. We call the set, $\{l_{i-L/2}, l_{i-L/2+1}, \dots, l_{i-1}, l_{i+1}, \dots, l_{i+L/2}\}$, the L -neighborhood of e around u .

As with the greedy approach to vertex coloring, an L -localized edge coloring can be found by a simple greedy algorithm that incrementally assigns colors to edges, one at a time. Each edge $e = (u, v)$ is colored with color c that does not appear in both L -neighborhoods of e (around u and around v). Using reasonable data structures, this greedy algorithm can be implemented to run in $O(mL)$ time, for a graph with m edges, and combining it with Vizing’s theorem [22], allows us to find an edge coloring that uses at most $\min\{d, 2L\} + 1$ colors.

4 Improving Resolution via Edge Coloring

As mentioned above, we define the angle between two incident arcs in the 3D arc diagram to be the angle between lines tangent to the arcs at their common endpoint. In order to reason about angles in 3D, the following lemma will prove useful.

Lemma 1. *Consider two segments l_1, l_2 that share a common endpoint that lies on a plane \mathcal{P} (see Fig. 3). If both l_1 and l_2 form angle $\beta \leq \pi/4$ with their projections onto \mathcal{P} , and projections of l_1 and l_2 onto \mathcal{P} form angle α , then δ , the angle between l_1 and l_2 , is at least $\alpha/2$.*

Proof. Assume w.l.o.g. that $|l_1| = |l_2| = 1$. The distance d between endpoints of l_1 and l_2 is the same as the distance between endpoints of projections of l_1 and l_2 onto \mathcal{P} (because both l_1 and l_2 form angle β with \mathcal{P}). Lengths of the projections are $\cos \beta$, and by the law of cosines,

$$d^2 = \cos^2 \beta + \cos^2 \beta - 2 \cos \beta \cos \beta \cos \alpha = 2 \cos^2 \beta (1 - \cos \alpha).$$

On the other hand, again by the law of cosines,

$$d^2 = |l_1|^2 + |l_2|^2 - 2|l_1||l_2| \cos \delta = 2(1 - \cos \delta).$$

Comparing the two yields

$$2 \cos^2 \beta (1 - \cos \alpha) = 2(1 - \cos \delta),$$

which leads to

$$\cos \delta = 1 - \cos^2 \beta (1 - \cos \alpha).$$

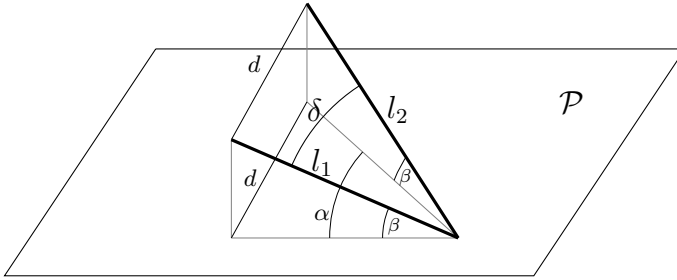


Fig. 3. Illustration of Lemma 1

For $\beta \leq \pi/4$,

$$\cos \delta \leq \cos \frac{\alpha}{2},$$

which means that

$$\delta \geq \frac{\alpha}{2}.$$

□

In addition, the following lemma will also be useful in our results.

Lemma 2. *Consider two segments, l_1 and l_2 , that share a common endpoint, with l_1 lying on a plane \mathcal{P} (see Fig. 4). If l_2 forms angle $\beta < \pi/4$ with its projection onto \mathcal{P} , then δ , the angle between l_1 and l_2 , is at least β .*

Proof. Assume w.l.o.g. that $|l_1| = |l_2| = 1$. Length of a , the projection of l_2 onto \mathcal{P} , is $\cos \beta$, and h , the distance of l_2 's endpoint from \mathcal{P} is $\sin \beta$. Let α be the angle between l_1 and a , and let b be the segment connecting their endpoints. By the law of cosines,

$$|b|^2 = |a|^2 + |l_1|^2 - 2|a||l_1| \cos \alpha = \cos^2 \beta + 1 - 2 \cos \beta \cos \alpha.$$

Then,

$$|d|^2 = |h|^2 + |b|^2 = \sin^2 \beta + \cos^2 \beta + 1 - 2 \cos \alpha \cos \beta = 2(1 - \cos \alpha \cos \beta).$$

Again, by the law of cosines,

$$|d|^2 = |l_1|^2 + |l_2|^2 - 2|l_1||l_2| \cos \delta = 2(1 - \cos \delta).$$

Comparing the two yields

$$\cos \delta = \cos \alpha \cos \beta.$$

Since $\cos \alpha \leq 1$, we get

$$\cos \delta \leq \cos \beta,$$

and it follows that $\delta \geq \beta$.

□

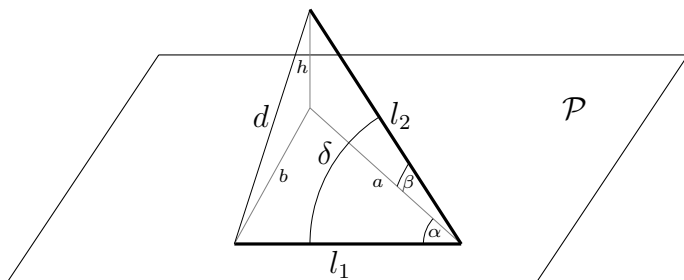


Fig. 4. Illustration of Lemma 2

4.1 Vertices on a Sphere

In this subsection, we consider the angular resolution obtained in a 3D arc diagram using straight-line edges drawn between vertices placed on a sphere. The two algorithms we present here are inspired by a two-dimensional drawing algorithm by Formann *et al.* [14]. Our main result is the following.

Theorem 1. *Let $G = (V, E)$ be a graph of degree d . There is a 3D straight-line drawing of G with an angular resolution of $\Omega(1/d)$, with the vertices of G placed on the surface on a sphere.*

Proof. Let $G^2 = (V, E^2)$ be the square of G , that is the graph with the same set of vertices as G , and an edge between vertices (u, v) if there is a path of length ≤ 2 between u and v in G . Since G has degree d , G^2 has degree $\leq d(d - 1) < d^2$. Therefore, we can color the vertices of G^2 with at most d^2 colors, with the requirement that adjacent vertices have different colors.

We place the vertices on a unit sphere \mathcal{S} . We define d^2 cluster positions as follows. First, we cut the circle with $d + 1$ uniformly spaced parallel planes (see Fig. 5), such that the maximum distance between the center of \mathcal{S} and a plane is h (thus, the distance between two neighboring planes is $2h/d$). Then, we uniformly place d points on each resulting circle. These are the cluster positions.

Since a coloring \mathcal{C} of G^2 uses $\leq d^2$ colors, we can assign distinct cluster positions to colors in \mathcal{C} . To obtain a drawing of G , we place all vertices of the same color in \mathcal{C} on the sphere, \mathcal{S} , within a small distance, ϵ , around this color’s cluster position, and draw edges in E as straight lines. We can remove any intersections by perturbing the vertices slightly.

The claim is that the resulting drawing has resolution $\Omega(1/d)$. Indeed, by setting $h = \pi/(\sqrt{1 + \pi^2})$, we get $\Omega(1/d)$ minimal distance between any two planes, and $\Omega(1/d)$ minimal distance between any two cluster positions on the same plane. So, the distance between any two cluster positions is at least $\Omega(1/d)$.

Now let us consider any angle $\sphericalangle abc$ formed by edges (a, b) and (b, c) . The edges forming $\sphericalangle abc$ define a plane, \mathcal{P} , whose intersection with \mathcal{S} is a circle, C . Angle $\sphericalangle abc$ is inscribed in C , and based on the arc \widehat{ac} . Therefore, any other angle inscribed in C and based on \widehat{ac} has the same size, in particular the one formed

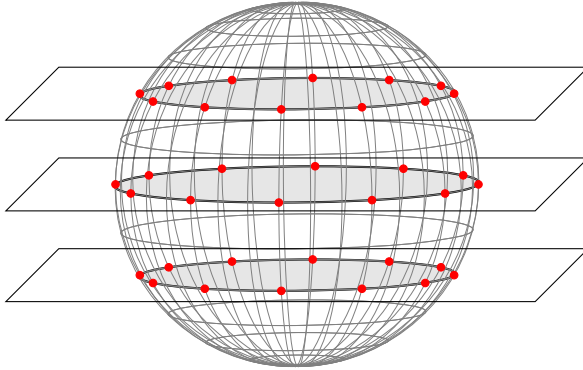


Fig. 5. Sphere cut with equidistant planes. Red points are the *cluster positions*.

by an isosceles triangle $\triangle adc$. Since $|ad| = |cd| \leq 2$ (\mathcal{S} has radius 1), and $|ac|$ is at least $\Omega(1/d)$, then $|\sphericalangle abc| = |\sphericalangle adc|$ and is at least $\Omega(1/d)$. \square

In addition, we also have the following.

Corollary 1. *Let $G = (V, E)$ be a planar graph of degree d . There is a 3D straight-line drawing of G with an angular resolution of $\Omega(1/d^{1/2})$, with the vertices of G placed on the surface of a sphere.*

Proof. The proof is a direct consequence of applying the algorithm from the proof of Theorem 1 and the fact that the degree of G^2 , the square of a planar graph, G , has degree $O(d)$ [14]. \square

Thus, we can produce 3D arc diagram drawings of planar graphs that achieve an angular resolution that is within a constant factor of optimal. Admittedly, this type of drawing is probably not going to be very pretty when rendered, say, as a video fly-through on a 2D screen, as this type of drawing is unlikely to project to a planar drawing in any direction.

4.2 Stationary Vertices

In this subsection, we show how to overcome the drawback of the above method, in that we show how to start with any existing 2D straight-line drawing and dramatically improve the angular resolution for that drawing using a 3D arc diagram rendering that projects perpendicularly to the 2D drawing.

Theorem 2. *Let $D(G)$ be a straight-line drawing of a graph, G , with arbitrary, but distinct, placements for its vertices in the base plane. There is a 3D arc diagram drawing of G with the same vertex placements as $D(G)$ and with an angular resolution at least $\Omega(1/d)$, where d is the degree of G , regardless of the angular resolution of $D(G)$.*

Proof. Since we are not allowed to move vertices, and edges have to lie on planes perpendicular to the *base plane*, we are restricted to selecting angles α_e for edges e of G . We do it by utilizing classical edge coloring, observing that the “entry” and “exit” angles for each vertex need to match.

First, we compute an edge coloring \mathcal{C} of G with c colors ($c \leq d + 1$). Then, for each edge e , if its color in \mathcal{C} is i ($i = 0, 1, \dots, c - 1$), we set its angle to be $\alpha_e = i \cdot \pi/4(c - 1)$. For any two edges e_1, e_2 , the difference between their angles α_{e_1} and α_{e_2} is at least $\pi/4(c - 1)$ (let $\alpha_{e_1} < \alpha_{e_2}$; consider the plane, \mathcal{P} , determined by both tangent lines having angle α_{e_1} ; the angle between e_2 and the plane \mathcal{P} , on which tangent of e_1 lies, is $\alpha_{e_2} - \alpha_{e_1}$). Therefore, by Lemma 2, the angle between e_1 and e_2 in the arc diagram is also at least $\pi/4(c - 1) = \Omega(1/d)$.

It is unlikely that any pairs of the arcs touch each other in 3D, but if any pair of them do touch, we can perturb one of them slightly to eliminate the crossing, while still keeping the angular separation for every pair of incident edges to be $\Omega(1/d)$. □

In addition, through the use of a slanted 3D arc diagram rendering, we can produce a drawing with angular resolution that is within a constant factor of optimal, with each arc projecting to its corresponding straight-line edge in some direction.

Theorem 3. *Let $D(G)$ be a straight-line drawing of a graph, G , with arbitrary, but distinct, placements for its vertices in the base plane. There is a slanted 3D arc diagram drawing of G with the same vertex placements as $D(G)$ and with an angular resolution at least $\Omega(1/d^{1/2})$, where d is the degree of G , regardless of the angular resolution of $D(G)$.*

Proof. Let C be a set of $\lceil d^{1/2} \rceil + 1$ uniformly distributed angles from 0 to $\pi/4$. Define a set of $d + 1$ “colors” as distinct pairs, (α, β) , where α and β are each in C . Compute an edge coloring of G using these colors. Now let e be an edge in G , which is colored with (α, β) . Draw the edge, e , using a circular arc that lies in a plane, \mathcal{P} , that makes an angle of α with the *base plane* and which has a tangent in \mathcal{P} that forms an angle of β at each endpoint of e . (For instance, in Fig. 1b, we give a slanted 3D arc diagram based on the edge coloring of the graph in Fig. 1a, corresponding to the following (α_e, β_e) “colors:” $(0^\circ, 0^\circ)$, $(22.5^\circ, 0^\circ)$, $(45^\circ, 0^\circ)$, $(22.5^\circ, 22.5^\circ)$, $(45^\circ, 45^\circ)$.)

The claim is that every pair of incident edges is separated by an angle of size at least $\Omega(1/d^{1/2})$. So suppose e and f are two edges incident on the same vertex, v . Let (α_e, β_e) be the color of e and let (α_f, β_f) be the color of f . Since e and f are incident and we computed a valid coloring for G , $\alpha_e \neq \alpha_f$ or $\beta_e \neq \beta_f$. In either case, this implies that e and f are separated by an angle of size at least $\Omega(1/d^{1/2})$ (by Lemma 1 if $\beta_e = \beta_f$, by Lemma 2 otherwise), which establishes the claim. As previously, we can perturb the arcs to eliminate crossings in 3D. □

Thus, we can achieve optimal angular resolution in a 3D arc diagram for any graph, G , to within a constant factor, for any arbitrary placement of vertices of G

in the plane. Note, however, that even if $D(G)$ is planar, the 3D arc diagram this algorithm produces, when projected to the *base plane*, may create edge crossings in the projected drawing. It would be nice, therefore, to have 3D arc diagrams that could have good angular resolution and also have planar perpendicular projections in the *base plane*.

4.3 Free Vertices

In this section, we show how to take any 2D straight-line drawing with good angular resolution and convert it to a 3D arc diagram with angular resolution that is within a constant factor of optimal. Moreover, this is the result that makes use of a localized edge coloring.

Theorem 4. *Let $D(G)$ be a straight-line drawing of a graph, G , with arbitrary, but distinct, placement for its vertices in the base plane, and $\Omega(1/d)$ angular resolution. There is a 3D arc diagram drawing of G with the same vertex placements as $D(G)$ and with angular resolution at least $\Omega(1/d^{1/2})$, where d is the degree of G , such that all arcs project perpendicularly as straight lines onto the base plane.*

Proof. The algorithm is similar to the one from the proof of Theorem 2. This time, however, we first compute an L -localized edge coloring, \mathcal{C} , of G utilizing c colors ($c \leq 2L + 1$). Then, as previously, we assign angle $\alpha_e = i \cdot \pi/4(c - 1)$ to an edge e of color i in \mathcal{C} ($i = 0, 1, \dots, c - 1$).

Let us consider two arcs, e and f , incident on a vertex, v . If $\alpha_e \neq \alpha_f$, then the angle between e and f is at least $\pi/(4c) = \Omega(1/L)$, by Lemma 2. Otherwise, $\alpha_e = \alpha_f$, and e and f have the same color in \mathcal{C} . By the definition of L -localized edge coloring, e and f are separated by at least $L/2$ edges around v . Because $D(G)$ has resolution $\Omega(1/d)$, the angle between e and f in $D(G)$ is $\Omega(L/d)$. Thus, by Lemma 1, the angle between e and f is also $\Omega(L/d)$. Therefore, the angle between e and f is $\Omega(\min\{1/L, L/d\})$. We achieve the advertised angular resolution by setting $L = d^{1/2}$. \square

Theorem 4 shows that we can achieve $\Omega(1/d^{1/2})$ angular resolution in a 3D arc diagram drawing of a graph, G , with arcs projecting perpendicularly onto the base plane as straight-line segments, if there is a straight-line drawing of G on a plane with an angular resolution of $\Omega(1/d)$. The following is an immediate consequence.

Corollary 2. *There is a 3D arc diagram drawing of any planar graph, G , with straight-line projection onto the base plane, and an angular resolution of $\Omega(1/d^{1/2})$.*

Proof. By [14], we can draw G in a straight-line manner on a plane with an angular resolution of $\Omega(1/d)$. \square

Admittedly, the 2D projection of this graph is not necessarily planar. We can nevertheless also achieve the following.

Corollary 3. *There is a 3D arc diagram drawing of any ordered tree, T , with straight-line projection onto the base plane, and an angular resolution of $\Omega(1/d^{1/2})$.*

Proof. By Duncan *et al.* [10], we can draw T in a straight-line manner on a plane with an angular resolution of $\Omega(1/d)$. \square

In addition, the area of the projection of the drawings produced by the previous two corollaries is polynomial.

5 Conclusion

We have given efficient algorithms for drawing 3D arc diagrams that achieve polynomial area in the base plane or sphere that contains all the vertices while also achieving good angular resolution. Since our algorithms deal with arc intersections via arc perturbation, the results may not be satisfactory, as the perturbed edges will still be very close. Therefore, one direction for future work is a related resolution question of what volumes are achievable if, in addition to angular resolution, we also insist that every circular arc always be at least unit distance from every other non-incident arc edge.

Acknowledgements. We thank Joe Simons, Michael Bannister, Lowell Trott, Will Devanny, and Roberto Tamassia for helpful discussions regarding angular resolution in 3D drawings.

References

1. Angelini, P., Eppstein, D., Frati, F., Kaufmann, M., Lazard, S., Mchedlidze, T., Teillaud, M., Wolff, A.: Universal point sets for planar graph drawings with circular arcs (2013) (manuscript)
2. Brandes, U.: Hunting down Graph B. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 410–415. Springer, Heidelberg (1999)
3. Brandes, U., Shubina, G., Tamassia, R.: Improving angular resolution in visualizations of geographic networks. In: Leeuw, W., Liere, R. (eds.) Data Visualization, Eurographics, pp. 23–32. Springer (2000)
4. Brandes, U., Shubina, G., Tamassia, R.: Improving angular resolution in visualizations of geographic networks. In: Proc. Joint Eurographics — IEEE TCVG Symposium on Visualization (VisSym 2000) (2000)
5. Cheng, C., Duncan, C., Goodrich, M., Kobourov, S.: Drawing planar graphs with circular arcs. *Discrete & Computational Geometry* 25(3), 405–418 (2001)
6. Cimikowski, A., Shope, P.: A neural-network algorithm for a graph layout problem. *IEEE Trans. on Neural Networks* 7(2), 341–345 (1996)
7. Cohen, R., Eades, P., Lin, T., Ruskey, F.: Three-dimensional graph drawing. *Algorithmica* 17(2), 199–208 (1997)
8. Djidjev, H.N., Vrt'o, I.: An improved lower bound for crossing numbers. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 96–101. Springer, Heidelberg (2002)

9. Duncan, C.A., Eppstein, D., Goodrich, M.T., Kobourov, S.G., Löffler, M.: Planar and poly-arc Lombardi drawings. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 308–319. Springer, Heidelberg (2011)
10. Duncan, C.A., Eppstein, D., Goodrich, M.T., Kobourov, S.G., Nöllenburg, M.: Drawing trees with perfect angular resolution and polynomial area. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 183–194. Springer, Heidelberg (2011)
11. Duncan, C.A., Eppstein, D., Goodrich, M.T., Kobourov, S.G., Nöllenburg, M.: Lombardi drawings of graphs. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 195–207. Springer, Heidelberg (2011)
12. Eppstein, D., Löffler, M., Mumford, E., Nöllenburg, M.: Optimal 3d angular resolution for low-degree graphs. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 208–219. Springer, Heidelberg (2011)
13. Fejes Tóth, L.: Über die Abschätzung des kürzesten Abstandes zweier Punkte eines auf einer Kugelfläche liegenden Punktsystems. *Jb. Deutsch. Math. Verein* 53, 66–68 (1943)
14. Formann, M., Hagerup, T., Haralambides, J., Kaufmann, M., Leighton, F.T., Symvonis, A., Welzl, E., Woeginger, G.J.: Drawing graphs in the plane with high resolution. In: FOCS, pp. 86–95. IEEE Computer Society (1990)
15. Garg, A., Tamassia, R.: Planar drawings and angular resolution: Algorithms and bounds. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 12–23. Springer, Heidelberg (1994)
16. Garg, A., Tamassia, R., Vocco, P.: Drawing with colors. In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 12–26. Springer, Heidelberg (1996)
17. Goodrich, M.T., Wagner, C.G.: A framework for drawing planar graphs with curves and polylines. *Journal of Algorithms* 37(2), 399–421 (2000)
18. Gutwenger, C., Mutzel, P.: Planar polyline drawings with good angular resolution. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 167–182. Springer, Heidelberg (1999)
19. Nicholson, T.: Permutation procedure for minimising the number of crossings in a network. *Proc. of the Inst. of Electrical Engineers* 115(1), 21–26 (1968)
20. Saaty, T.L.: The minimum number of intersections in complete graphs. *Proceedings of the National Academy of Sciences of the United States of America* 52(3), 688–690 (1964)
21. Tammes, P.: On the origin of number and arrangements of the places of exit on the surface of pollen grains. *Rec. Trav. Bot. Neerl.* 27, 1–81 (1930)
22. Vizing, V.G.: On an estimate of the chromatic class of a p -graph. *Diskret. Analiz.* 3, 25–30 (1964)
23. Wattenberg, M.: Arc diagrams: Visualizing structure in strings. In: IEEE Symp. on Information Visualization (InfoVis), pp. 110–116 (2002)

A Duality Transform for Constructing Small Grid Embeddings of 3D Polytopes^{*}

Alexander Igamberdiev and André Schulz

Institut für Mathematische Logik und Grundlagenforschung,
Universität Münster, Germany
{alex.igamberdiev, andre.schulz}@uni-muenster.de

Abstract. We study the problem of how to obtain an integer realization of a 3d polytope when an integer realization of its dual polytope is given. We focus on grid embeddings with small coordinates and develop novel techniques based on Colin de Verdière matrices and the Maxwell–Cremona lifting method.

As our main result we show that every truncated 3d polytope with n vertices can be realized on a grid of size polynomial in n . Moreover, for a class \mathcal{C} of simplicial 3d polytopes with bounded vertex degree, at least one vertex of degree 3, and polynomial size grid embedding, the dual polytopes of \mathcal{C} can be realized on a polynomial size grid as well.

1 Introduction

By Steinitz’s theorem the graphs of convex 3d polytopes¹ are exactly the planar 3-connected graphs [16]. Several methods are known for realizing a planar 3-connected graph G as a polytope with graph G on the grid [4,7,11,12,13,15]. It is challenging to find algorithms that produce polytopes with small integer coordinates. Having a realization with small grid size is a desirable feature, since then the polytope can be stored and processed efficiently. Moreover, grid embeddings imply good vertex and edge resolution. Hence, they produce “readable” drawings.

In 2d, it is well known that planar 3-connected graphs with n vertices can be drawn on a $O(n) \times O(n)$ grid without crossings [5], and a drawing with convex faces can be realized on a $O(n^{3/2} \times n^{3/2})$ grid [2]. For the realization as a polytope the best algorithm guarantees an integer embedding with coordinates at most $O(147.7^n)$ [3,11]. The current best lower bound is $\Omega(n^{3/2})$ [1]. Closing this large gap is probably one of the most interesting open problems in lower dimensional polytope theory. Recently, progress has been made for a special class of 3d polytopes, the so-called *stacked polytopes*. A *stacking operation* replaces a triangular face of a polytope with a tetrahedron, while maintaining the convexity of the embedding. A polytope that can be constructed from a tetrahedron and a

^{*} This work was funded by the German Research Foundation (DFG) under grant SCHU 2458/2-1.

¹ In our terminology polytopes are always considered *convex*.

sequence of stacking operation is called stacked polytope. The graphs of stacked polytopes are planar 3-trees. Stacked polytopes can be embedded on a grid that is polynomial in n [6]. This is, however, the only nontrivial polytope class for which such an algorithm is known.

In this paper we introduce a duality transform that maintains a polynomial grid size. In other words, we provide a technique that takes a grid embedding of a polytope with graph G and generates a grid embedding of a polytope whose skeleton is G^* , the dual graph of G . We call a 3d polytope with graph G^* a dual polytope. If the original polytope has integer coordinates bounded by a polynomial in n , then the dual polytope obtained with our techniques has also integer coordinates bounded by a (different) polynomial in n . Our methods can only be applied to special polytopes. Namely, we require that the graph of the polytope is a triangulation (the polytope is simplicial), that it contains a K_4 , and that the maximum vertex degree is bounded.

For the class of stacked polytopes (although their maximum vertex degree is not bounded) we can also apply our approach to show that all graphs dual to planar 3-trees can be embedded as polytopes on a polynomial size grid. These polytopes are known as truncated polytopes. Truncated polytopes are simple polytope, that can be generated from a tetrahedron and a sequence of *truncations*. A truncation is the dual operation to stacking. This means that a degree-3 vertex of the polytope is cut off by adding a new bounding hyperplane that separates this vertex from the remaining vertices of the polytope. We show that all truncated polytopes can be realized with integer coordinates in $O(n^{44})$. The approach for this class is more direct, since stronger results for realizations of stacked polytopes on the grid are known [6].

Duality. There exist several natural approaches how to construct for a given polytope a dual. The most prominent construction is polarity. Let P be some polytope that contains the origin. Then $P^* = \{y \in \mathbb{R}^d: x^T y \leq 1 \text{ for all } x \in P\}$ is a realization of a polytope dual to P , called its *polar*. The vertices of P^* are intersection points of planes with integral normal vectors, and hence not necessarily integer points. In order to scale to integrality one has to multiply P^* with the product of all denominators of its vertex coordinates, which may cause an exponential increase of the grid size.

A second approach uses the classic Maxwell–Cremona correspondence technique (also known as lifting approach) [10], which is applied in many embedding algorithms for 3d polytope realization. The idea here is to first draw the graph of the polytope as a convex 2d embedding with an additional equilibrium condition. The equilibrium condition guarantees that the 2d drawing is a projection of a convex 3d polytope, furthermore the polytope can be reconstructed from its projection in a canonical way (called lifting) in linear time. There is a classical transformation that constructs for a 2d drawing in equilibrium a 2d drawing of its dual graph, also in equilibrium. This drawing is called the *reciprocal diagram*. The induced lifting realizes the dual polytope, but it does not provide small integer coordinates for two reasons. First, the weights that define the equilibrium of the reciprocal diagram are the reciprocals of the weights in the original graph.

Second, the lifting realizes the dual polytope in projective space with one point “over the horizon”. The second property can be “fixed” with a projective transformation. This, however, makes a large scaling factor for an integer embedding unavoidable. Also the reciprocal weights are difficult to handle without scaling by a large factor.

Structure and notation. As a novelty we work with Colin de Verdière matrices to construct *small* grid embeddings. In order to make these techniques (as introduced by Lovász) applicable we extend this framework slightly; see Sect. 2. In Sect. 3 we then present the main idea, combining the classical lifting approach with the methods of Sect. 2, which finds applications in the following sections, where the results on truncated polytopes and triangulations are presented.

Throughout the paper we denote by G the graph of the original polytope, and by G^* its dual graph. For any graph H we write $V(H)$ for its vertex set, $E(H)$ for its edge set and $N(v, H)$ for the set of neighbors of a vertex v in H . Since we consider 3-connected planar graphs, the facial structure of the graph is predetermined up to a global reflection [17]. The set of faces is therefore predetermined, and we name it $F(H)$. For convenience we denote an edge (v_i, v_j) as (i, j) . A face spanned by vertices $v_i, v_j,$ and v_k is denoted as $(v_i v_j v_k)$. A graph obtained from H by stacking a vertex v_1 on a face $(v_2 v_3 v_4)$, is denoted as $\text{Stack}(H; v_1; v_2 v_3 v_4)$. For convenience we use $|p|$ for the Euclidean norm of the vector p . We denote the maximum vertex degree of a graph G as Δ_G . Finally, we write $G[X]$ for the induced subgraph of a vertex set $X \subseteq V(G)$.

2 3d Representations with CDV Matrices

In this section we review some of the methods Lovász introduced in his paper on Steinitz representations [9]. In our constructions throughout the paper every face of any graph is realized such that all its vertices lie on a common plane. From this perspective drawings of graphs in \mathbb{R}^3 and the realizations of their corresponding polyhedra are the same objects.

Definition 1. We call a straight-line embedding $(u_1, \dots, u_n) \in (\mathbb{R}^3)^n$ of a planar 3-connected graph G in \mathbb{R}^3 a cone-convex embedding iff the cones over its faces, $C_f = \{\lambda x \mid x \in f, \lambda > 0\}$, $f \in F(G)$ are convex and have disjoint interiors.

In other words, an embedding is a cone-convex embedding if its projection to the sphere $S = \{|x| = 1\}$ is a convex drawing of G with edges drawn as geodesic arcs. We remark that the vertices of a cone-convex embedding are not supposed to form a convex polytope.

Definition 2. Let (u_1, \dots, u_n) be an embedding of a graph G into \mathbb{R}^d . We call a symmetric matrix $M = [M_{ij}]_{1 \leq i, j \leq n}$ a CDV matrix of the embedding if

1. $M_{ij} = 0$ for $i \neq j, (i, j) \notin E(G)$, and
2. $\sum_{1 \leq j \leq n} M_{ij} u_j = 0$ for $1 \leq i \leq n$.

We call a CDV Matrix positive if $M_{ij} > 0$ for all $(i, j) \in E(G)$.

We call the second condition in the above definition the *CDV equilibrium condition*.

The CDV equilibrium condition can also be expressed in a slightly different, more geometric form as

$$\sum_{j \in N(i, G)} M_{ij} u_j = -M_{ii} u_i \quad \text{for } 1 \leq i \leq n. \tag{1}$$

Hence, a positive CDV Matrix witnesses that every vertex of the embedding can be written as a convex combination of its neighbors using symmetric weights. The following lemma appears in [9], we include the proof since it illustrates how to construct a realization out of a CDV matrix.

Lemma 1 (Lemma 4, [9]). *Let (u_1, \dots, u_n) be a cone-convex embedding of a graph G with a positive CDV matrix $[M_{ij}]$. Then every face f in G can be assigned a vector ϕ_f s.t. for each adjacent face g and separating edge (i, j)*

$$\phi_f - \phi_g = M_{ij}(u_i \times u_j), \tag{2}$$

where f lies to the left and g lies to the right from $\overrightarrow{u_i u_j}$. The set of vectors $\{\phi_f\}$ is uniquely defined up to translations.

Proof. To construct the family of vectors $\{\phi_f\}$, we start by assigning an arbitrary value to ϕ_{f_0} (for an arbitrary face f_0); then we proceed iteratively. To prove the consistency of the construction, we show that the vectors $(\phi_f - \phi_g)$ sum to zero over every cycle in G^* . Since G as well as G^* is planar and 3-connected, it suffices to check this condition for all elementary cycles of G^* , which are the faces of G^* . Let $\tau(i)$ denote the set of counterclockwise oriented edges of the face in G^* dual to $v_i \in V(G)$. Then, combining 1 and 2 yields

$$\sum_{(f, g) \in \tau(i)} (\phi_f - \phi_g) = \sum_{j \in N(i, G)} M_{ij}(u_i \times u_j) = u_i \times \left(\sum_{j \in N(i, G)} M_{ij} u_j \right) = u_i \times (-M_{ii} u_i) = 0.$$

The vectors $\{\phi_f\}$ are unique up to the initial choice of ϕ_{f_0} . □

Note that there is a canonical way to derive a CDV matrix from a 3d polytope [9]. Every 3d embedding of a graph G as a polytope (u_i) possesses a positive CDV matrix defined by the vertices (ϕ_i) of its polar and equation (2). We refer to this matrix as the *canonical CDV matrix*.

The following theorem, which is a variation of Lemma 5 in [9], is the main tool in our construction.

Theorem 1 (based on Lovász [9]). *Let (u_1, \dots, u_n) be a cone-convex embedding of a graph G and M a positive CDV matrix for this embedding. Then for any set of vectors $\{\phi_f\}_{f \in F(G)}$ fulfilling (2), the convex hull $\text{Conv}(\{\phi_f\}_{f \in F(G)})$ is a convex polytope with graph G^* ; and the isomorphism between G^* and the skeleton of $\text{Conv}(\{\phi_f\}_{f \in F(G)})$ is given by $f \rightarrow \phi_f$.*

The proof of the theorem is included in the full version of the paper. It relies on a projection of the cone-convex embedding onto the sphere and an appropriate “scaling” of the CDV matrix.

3 Construction of Cone-Convex Embeddings

In this section we describe how to go from a convex 2d embedding with a positive equilibrium stress to a cone-convex 3d embedding with a positive CDV matrix.

Definition 3. We call a set of reals $\{\omega_{ij}\}_{(i,j) \in E(G)}$ an equilibrium stress for an embedding (u_1, \dots, u_n) of a graph G into \mathbb{R}^d if for each $i \in V(G)$

$$\sum_{j \in N(i,G)} \omega_{ij}(u_j - u_i) = 0.$$

We call an equilibrium stress of a 2d embedding with a distinguished boundary face f_0 positive if it is positive on every edge that does not belong to f_0 .

The concept of equilibrium stresses plays a central role in the classical Maxwell–Cremona lifting approach and it is also a crucial concept in our embedding algorithm. The equilibrium stress on a realization of a complete graph arises as a “building block” in later constructions. The complete graph K_n , embedded in \mathbb{R}^{n-2} , has a unique equilibrium stress up to multiplication with a scalar. This stress has an easy expression in terms of volumes related to the embedding. We use the square bracket notation²

$$[q_i q_j q_k q_l] := \det \begin{pmatrix} x_i & x_j & x_k & x_l \\ y_i & y_j & y_k & y_l \\ z_i & z_j & z_k & z_l \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad \text{where } q = \begin{pmatrix} x \\ y \\ z \end{pmatrix},$$

to obtain a formulation for the equilibrium stress on the K_5 embedding.

Lemma 2 (Rote, Santos, and Streinu [14]). Let (u_0, u_1, \dots, u_4) be an integer embedding of the complete graph K_5 onto \mathbb{R}^3 . Then the set of real numbers:

$$\omega_{ij} := [u_{i-2} u_{i-1} u_{i+1} u_{i+2}] [u_{j-2} u_{j-1} u_{j+1} u_{j+2}]$$

(indices in cyclic notation) defines an integer equilibrium stress on this embedding.

Theorem 2. Let (p_2, \dots, p_n) be a convex 2d drawing of a planar 3-connected graph G_\uparrow with positive equilibrium stress $\{\omega_{ij}\}$ and designated triangular face $f_0 = (p_2 p_3 p_4)$ embedded as the boundary face. Then we can define a cone convex embedding (q_i) of the graph $G = \text{Stack}(G_\uparrow; v_1; v_2 v_3 v_4)$ into \mathbb{R}^3 equipped with a positive CDV matrix $[M_{ij}]$, such that

$$M_{ij} = \omega_{ij} \quad \text{for each internal edge } (i, j) \text{ of the 2d drawing of } G_\uparrow$$

and each entry of M is bounded by $O(n \max_{ij} |\omega_{ij}| \cdot \max_i |p_i|^6)$.

² For 2d vectors $[p_i p_j p_k]$ is defined similarly.

Proof. We can assume that $(0, 0)^T$ lies inside the embedding of f_0 . Let (q_1, \dots, q_n) be the embedding of the graph G , defined as follows: The embedding of G_\uparrow is realized in the plane $\{z = 1\}$ and the stacked vertex is placed at $(0, 0, -1)^T$. The embedding is cone-convex since it describes a tetrahedron containing the origin with one face that is refined with a plane convex subdivision.

Following the structure of $G = \text{Stack}(G_\uparrow; v_1; v_2v_3v_4)$, we decompose G into two subgraphs: $G_\uparrow = G[\{v_2, \dots, v_n\}]$ and $G_\downarrow := G[\{v_1, v_2, v_3, v_4\}]$.

We first compute a CDV matrix $[M'_{ij}]_{2 \leq i, j \leq n}$ for the embedding (q_2, \dots, q_n) of G_\uparrow . The plane embedding (p_i) of G_\uparrow has the equilibrium stress $\{\omega_{ij}\}_{2 \leq i, j \leq n}$. Since $\{q_2, \dots, q_n\}$ is just a translation of $\{p_2, \dots, p_n\}$, clearly, $\{\omega_{ij}\}_{2 \leq i, j \leq n}$ is as well an equilibrium stress for the embedding (q_2, \dots, q_n) and we can assign:

$$M'_{ij} := \begin{cases} -\sum_{k \in N(i, G_\uparrow)} \omega_{ik} & i = j, \\ \omega_{ij} & (i, j) \in E(G_\uparrow), \\ 0 & \text{else.} \end{cases}$$

Now we check the CDV equilibrium condition: for every $2 \leq i \leq n$

$$\begin{aligned} \sum_{2 \leq j \leq n} M'_{ij} q_j &= \sum_{j \in N(i, G_\uparrow)} M'_{ij} q_j + M'_{ii} q_i = \sum_{j \in N(i, G_\uparrow)} M'_{ij} (q_j - q_i) + (M'_{ii} + \sum_{j \in N(i, G_\uparrow)} M'_{ij}) q_i \\ &= \sum_{j \in N(i, G_\uparrow)} \omega_{ij} (q_j - q_i) + (M'_{ii} + \sum_{j \in N(i, G_\uparrow)} M'_{ij}) q_i = 0. \end{aligned}$$

The last transition holds since both summands equal 0. Hence, $[M'_{ij}]$ is a valid CDV matrix for the embedding $(q_i)_{2 \leq i \leq n}$ of G_\uparrow .

As a second step we compute a CDV matrix $[M''_{ij}]_{1 \leq i, j \leq 4}$ for the embedding of the tetrahedron G_\downarrow . We apply Lemma 2 for the embedding of the K_5 formed by $\{q_0 = (0, 0, 0)^T, q_1, q_2, q_3, q_4\}$ and receive an equilibrium stress $\{\omega''_{ij}\}_{0 \leq i, j \leq 4}$. We can now derive a CDV matrix $[M''_{ij}]_{1 \leq i, j \leq 4}$ for the tetrahedron $\{q_1, q_2, q_3, q_4\}$ based on the equilibrium stresses $\{\omega''_{ij}\}_{0 \leq i, j \leq 4}$ as follows: We set

$$M''_{ij} := \begin{cases} -\sum_{0 \leq j \leq 4, j \neq i} \omega''_{ij}, & i = j, \\ \omega''_{ij}, & \text{otherwise,} \end{cases}$$

and see that the CDV equilibrium condition holds, by noting

$$\begin{aligned} \forall i \quad \sum_{1 \leq j \leq 4} M''_{ij} q_j &= \sum_{1 \leq j \leq 4, j \neq i} M''_{ij} q_j + M''_{ii} q_i = \sum_{1 \leq j \leq 4, j \neq i} \omega''_{ij} q_j + M''_{ii} q_i \\ &= \sum_{0 \leq j \leq 4, j \neq i} \omega''_{ij} (q_j - q_i) - \omega''_{i0} q_0 + \left(\sum_{0 \leq j \leq 4, j \neq i} \omega''_{ij} + M''_{ii} \right) q_i = 0. \end{aligned}$$

The last transition holds since $\sum_{0 \leq j \leq 4, j \neq i} \omega''_{ij} (q_j - q_i) = 0$ by the definition of $\{\omega_{ij}\}$, $q_0 = 0$, and $\sum_{0 \leq j \leq 4, j \neq i} \omega''_{ij} + M''_{ii} = 0$ due to the choice of M''_{ii} . One can easily check that as soon as the origin lies inside the tetrahedron $\{q_1, q_2, q_3, q_4\}$ all entries M''_{ij} have the same sign. We can assume that $[M''_{ij}]$ is positive, otherwise we reorder the vertices $\{v_2, v_3, v_4\}$.

In the final step we extend the two CDV matrices M' and M'' to G and combine them. Clearly, a CDV matrix padded with zeros remains a CDV matrix. Furthermore, any linear combination of CDV matrices is again a CDV matrix. Thus, we form a CDV matrix for the whole embedding (q_1, \dots, q_n) of G by setting:

$$M := M' + \lambda M'',$$

where λ is a positive integer chosen such that M is a positive CDV matrix. This can be done as follows.

Recall that $\{\omega_{ij}\}$ is a positive stress and $[M''_{ij}]$ is a positive CDV matrix. Hence, the only six entries in $[M_{ij}]$ that may be negative are: M_{23} , M_{34} and M_{42} (and their symmetric entries), for which $M_{ij} := M'_{ij} + \lambda M''_{ij}$ with $M'_{ij} < 0$ and $M''_{ij} > 0$. Thus, we choose λ such that M is positive at these entries. To satisfy this condition we pick

$$\lambda = \left\lceil \max_{(i,j) \in \{(2,3), (3,4), (4,2)\}} (|M'_{ij}| / |M''_{ij}|) \right\rceil + 1.$$

To bound M_{ij} we notice that the entries of M''_{ij} are strictly positive integers, so $\lambda = O(\max |M'_{ij}|)$, while $|M'_{ij}| = O(n \cdot \max |\omega_{ij}|)$ and $|M''_{ij}| = O(\max |\omega''_{ij}|) = O(\max |p_i|^6)$. The bound $|M_{ij}| = O(n \cdot \max_{ij} |\omega_{ij}| \cdot \max_i |p_i|^6)$ follows. \square

4 Realizations of Truncated Polytopes

In this section we sum up previous results in Theorem 3 and present an embedding algorithm for truncated 3d polytopes in Theorem 4. We will apply Theorem 3 also in the more general setup of Sect. 5.

Theorem 3. *Let $G = \text{Stack}(G_\uparrow; v_1; v_2 v_3 v_4)$ and (p_2, \dots, p_n) be an integer planar embedding of G_\uparrow with boundary face $(v_2 v_3 v_4)$ and with positive integer equilibrium stress $\{\omega_{ij}\}$. Then one can construct a grid embedding (ϕ_f) of a convex polytope with graph G^* such that*

$$|\phi_f| = O(n^2 \cdot \max |\omega_{ij}| \cdot \max |p_i|^8).$$

Proof. We first apply Theorem 2 to obtain a cone-convex embedding (q_1, \dots, q_n) of G with a positive CDV matrix $[M_{ij}]_{1 \leq i, j \leq n}$. We then apply Lemma 1 and obtain a family of vectors $\{\phi_f\}_{f \in F(G^*)}$ fulfilling

$$\phi_f - \phi_g = M_{ij}(q_i \times q_j), \quad \forall (f, g) \text{ dual to } (i, j) - \text{edges of } G^* \text{ and } G.$$

Due to Theorem 1 the vectors $\{\phi_f\}$ form a realization of G^* as a polytope.

To finish the proof we estimate how large the coordinates of the embedding (ϕ_f) are. To do so, let us again follow the construction of (ϕ_f) as outlined in the proof of Lemma 1. We pick one face as $f_0 \in F(G)$, and assign $\phi_{f_0} = (0, 0, 0)^T$. Let us now evaluate ϕ_{f_k} for some face $f_k \in F(G)$. The following algebraic expression holds for all values $\{\phi_{f_i}\}$:

$$\phi_{f_k} = \phi_{f_0} + (\phi_{f_1} - \phi_{f_0}) + \dots + (\phi_{f_{k-1}} - \phi_{f_{k-2}}) + (\phi_{f_k} - \phi_{f_{k-1}}).$$

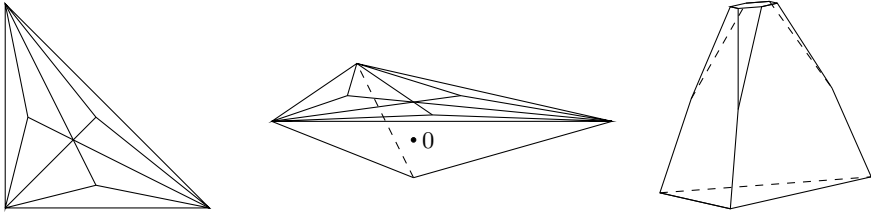


Fig. 1. A 2d embedding of G_\uparrow (left), the cone-convex embedding of G (center), and the resulting embedding of the dual (right).

Let us now consider the shortest path f_0, f_1, \dots, f_k in G^* connecting the faces f_0 and f_k . Clearly, k is less than $2n - 3$, and hence

$$\begin{aligned}
 |\phi_{f_k}| &\leq 2n \cdot \max_{(f_a, f_b) \in E(G^*)} |\phi_{f_a} - \phi_{f_b}| = 2n \cdot \max_{v_i, v_j \in V(G)} |M_{ij}(q_i \times q_j)| \\
 &= O(n \cdot (n \cdot \max |\omega_{ij}| \cdot \max |p_i|^6) \cdot \max |q_i|^2) = O(n^2 \cdot \max |\omega_{ij}| \cdot \max |p_i|^8).
 \end{aligned}$$

The bound for the entries of M is due to Theorem 2. □

Next we apply Theorem 3 to construct an integer polynomial size grid embedding for truncated polytopes. To construct small integer 2d embeddings with small integer equilibrium stresses we use a Lemma by Demaine and Schulz [6], which states that the graph of a stacked polytope with n vertices and any distinguished face f_0 can be embedded on a $10n^4 \times 10n^4$ grid with boundary face f_0 and with integral positive equilibrium stress $\{\omega_{ij}\}$ such that, for every edge (i, j) , we have $|\omega_{ij}| = O(n^{10})$.

Theorem 4. *Any truncated 3d polytope with n vertices can be realized with integer coordinates of size $O(n^{44})$.*

Proof. Let G^* be the graph of the truncated polytope and $G := (G^*)^*$ its dual. Clearly, G is the graph of a stacked polytope with $(n + 4)/2$ vertices. We denote the last stacking operation (for some sequence of stacking operations producing G) as the stacking of the vertex v_1 onto the face $(v_2v_3v_4)$ of the graph $G_\uparrow := G[V \setminus \{v_1\}]$. The graph G_\uparrow is again a stacked graph, and hence, by the Lemma of Demaine and Schulz, there exists an embedding $(p_i)_{2 \leq i \leq n}$ of G_\uparrow into \mathbb{Z}^2 with an equilibrium stress $\{\omega_{ij}\}$ satisfying the properties of Theorem 3. We apply the theorem and obtain a polytope embedding (ϕ_f) of G^* with bound $|\phi_f| = O(n^2 \cdot \max |\omega_{ij}| \cdot \max |p_i|^8) = O(n^{44})$. □

Figure 1 shows an example of our method. The computations for this example are included in the full paper.

5 A Dual Transform for Simplicial Polytopes

As we have seen a small grid embedding of a 3d polytope can be computed when a small integer (though, not necessarily convex) embedding of its dual polytope

with a small integral positive CDV matrix is known. However, if one wants to build a dual for an already embedded polytope, one usually does not possess such a matrix. The canonical CDV matrix associated with any embedding of a 3d polytope is not helpful, since its entries, when scaled to integers, might become exponentially large. We show in this section how one can tackle this problem for a special class of polytopes. In particular, we require that the original polytope is simplicial, it contains a vertex of degree 3, and its maximum vertex degree is bounded.

Before proceeding, let us review how the canonical stress associated with an orthogonal projection of a 3d polytope in the $\{z = 0\}$ plane can be described. The assignment of heights to the interior vertices of a 2d embedding resulting in a polyhedral surface is called a (polyhedral) *lifting*. By the Maxwell-Cremona correspondence the equilibrium stresses of a 2d embedding of a planar 3-connected graph and its liftings are in 1-1 correspondence. Moreover, the bijection between liftings and stresses can be defined as follows. Let (p_i) be a 2d drawing of a triangulation and let (q_i) be the 3d embedding induced by some lifting. We map this lifting to the equilibrium stress $\{\omega_{ij}\}$ by assigning to every edge (i, j) separating the faces $(v_i v_j v_k)$ (on the left) and $(v_i v_j v_l)$ (on the right)

$$\omega_{ij} := \frac{[q_i q_j q_k q_l]}{[p_i p_j p_k][p_l p_j p_i]}. \tag{3}$$

This mapping is a bijection between the space of liftings and the space of equilibrium stresses. The expression (3) is a slight reformulation of the form presented in Hopcroft and Kahn [8, Equation 11].

We continue by studying the spaces of equilibrium stresses for triangulations. A graph formed by a cycle v_1, \dots, v_n with an additional vertex v_0 , called *center*, that is adjacent to every other vertex, is called a *wheel*; we denote it as $W(v_0; v_1 \dots v_n)$. A wheel that is a subgraph of a triangulation G with $v_i \in V(G)$ as center is denoted by W_i . Every triangulation can be “covered” with a set of wheels $\{W_i\}_{v_i \in V(G)}$, such that every edge is covered four times.

Lemma 3. *Let (p_0, \dots, p_n) be an embedding of a wheel $W(v_0; v_1 \dots v_n)$ in \mathbb{R}^2 . Then the following expression defines an equilibrium stress:*

$$\omega_{ij} = \begin{cases} -1/[p_i p_{i+1} p_0] & j = i + 1, 1 \leq i \leq n, \\ [p_{i-1} p_i p_{i+1}] / ([p_{i-1} p_i p_0][p_i p_{i+1} p_0]) & j = 0, 1 \leq i \leq n. \end{cases}$$

The equilibrium stress for the embedding (p_i) is unique up to a renormalization.

Proof. This stress coincides with (3) from the lifting of W with $z_0 = 1$ and $z_i = 0$ for $1 \leq i \leq n$ and so is an equilibrium stress. The space of the stresses is 1-dimensional, since the space of the polyhedral liftings is 1-dimensional. \square

Definition 4. *1. For a wheel W embedded in the plane we refer to the equilibrium stress defined in Lemma 3 as its small atomic stress and denote it as $\{\omega_{ij}^a(W)\}$.*

2. We call the stress $\{\Omega_{ij}^a(W)\}$ that is obtained by the renormalization of $\{\omega_{ij}^a(W)\}$ by the factor $\prod_{1 \leq j \leq n} [p_j p_{j+1} p_0]$, the large atomic stress of W .

We point out that the large atomic stresses are products of $\deg(v_0) - 1$ triangle areas multiplied by 2, and so, $\{\Omega_{ij}^a(W)\}$ is a set of integers if W is realized with integer coordinates.

Theorem 5 (Wheel-decomposition). *Let G be a triangulation. Every equilibrium stress $\{\omega_{ij}\}$ of an embedding (p_1, \dots, p_n) of G can be expressed as a linear combination of the small atomic stresses on the wheels $\{W_i\}$:*

$$\omega = \sum_{i \leq n} \alpha_i \omega^a(W_i),$$

where the coefficients α_i are the heights (i.e., z -coordinates) of the corresponding vertices v_i in the Maxwell–Cremona lifting of (p_1, \dots, p_n) induced by $\{\omega_{ij}\}$.

Proof. Let (q_1, \dots, q_n) be the Maxwell–Cremona lifting of (p_i) by means of the stress $\{\omega_{ij}\}$. We rewrite this stress (given by Equation 3) using

$$[q_1 q_2 q_3 q_4] = \sum_{1 \leq i \leq 4} (-1)^{i+1} z_i [p_{i+1} p_{i+2} p_{i+3}],$$

(with cyclic notation for indices) and obtain

$$\omega_{ij} = z_i \frac{[p_j p_k p_l]}{[p_i p_j p_k][p_l p_j p_i]} + z_j \frac{[p_l p_k p_i]}{[p_i p_j p_k][p_l p_j p_i]} - z_k \frac{1}{[p_i p_j p_k]} - z_l \frac{1}{[p_l p_j p_i]},$$

which is exactly the decomposition of ω_{ij} into small atomic wheel stresses. \square

Theorem 6. *Let (q_1, \dots, q_n) be an embedding of a triangulation G into \mathbb{Z}^3 , whose projection (p_1, \dots, p_n) to the plane $\{z = 0\}$ is a noncrossing embedding of G with boundary face $(v_1 v_2 v_3)$. Then one can construct a positive integer equilibrium stress $\{\omega_{ij}\}$ for the embedding (p_1, \dots, p_n) such that*

$$|\omega_{ij}| < (\max_{i \leq n} |q_i|)^{2 \Delta_G + 5}.$$

Proof. We start with the equilibrium stress $\{\tilde{\omega}_{ij}\}$ as specified by (3) for the embedding (p_i) . Since all the coordinates are integers, all stresses are bounded by

$$\frac{1}{L^4} \leq \frac{1}{|[p_i p_j p_k][p_l p_j p_i]|} \leq |\tilde{\omega}_{ij}| \leq |[q_i q_j q_k q_l]| \leq L^3,$$

for $L = 2 \max_{i \leq n} |q_i|$.

We are left with making these stresses integral while preserving a polynomial bound. The stress $\{\tilde{\omega}_{ij}\}$ can be written as a linear combination of large atomic stresses of the wheels W_i by means of the Wheel-decomposition Theorem,

$$\tilde{\omega}_{ij} = \alpha_i \Omega_{ij}^a(W_i) + \alpha_j \Omega_{ij}^a(W_j) + \alpha_k \Omega_{ij}^a(W_k) + \alpha_l \Omega_{ij}^a(W_l).$$

Since all points p_i have integer coordinates, the large atomic stresses are integers as well. Moreover, each of them, as a product of $\deg(v_k) - 1$ triangle areas, is bounded by $|\Omega_{ij}^a(W_k)| \leq L^{2(\Delta_G - 1)}$.

To make the $\tilde{\omega}_{ij}$ s integral we round the coefficients α_i down. To guarantee that the rounding does not alter the signs of the stress, we scale the atomic stresses (before rounding) with the factor

$$C = 4 \max_{i,j,k} |\Omega_{ij}^a(W_k)| / \min_{i,j} |\tilde{\omega}_{ij}|$$

and define as the new stress:

$$\omega_{ij} := \lfloor C\alpha_i \rfloor \Omega_{ij}^a(W_i) + \lfloor C\alpha_j \rfloor \Omega_{ij}^a(W_j) + \lfloor C\alpha_k \rfloor \Omega_{ij}^a(W_k) + \lfloor C\alpha_l \rfloor \Omega_{ij}^a(W_l).$$

Clearly,

$$\begin{aligned} |\omega_{ij} - C\tilde{\omega}_{ij}| &= \left| \sum_{\tau=i,j,k,l} (\lfloor C\alpha_\tau \rfloor - C\alpha_\tau) \Omega_{ij}^a(W_\tau) \right| \\ &\leq \sum_{\tau=i,j,k,l} |\Omega_{ij}^a(W_\tau)| \leq 4 \max_{i,j,k} |\Omega_{ij}^a(W_k)| = C \min_{i,j} |\tilde{\omega}_{ij}| \leq C|\tilde{\omega}_{ij}| \end{aligned}$$

and so $\text{sign}(\omega_{ij}) = \text{sign}(C\tilde{\omega}_{ij}) = \text{sign}(\tilde{\omega}_{ij})$.

Therefore, the constructed equilibrium stress $\{\omega_{ij}\}$ is integral and positive. We conclude the proof with an upper bound on its size. Since $C < 4L^{2(\Delta_G - 1)}L^4$,

$$\begin{aligned} |\omega_{ij}| &\leq \left| \sum_{\tau=i,j,k,l} (C\alpha_\tau \pm 1) \Omega_{ij}^a(W_\tau) \right| \leq C|\tilde{\omega}_{ij}| + \sum_{\tau=i,j,k,l} |\Omega_{ij}^a(W_\tau)| \\ &\leq C \max |\tilde{\omega}_{ij}| + 4 \max |\Omega_{ij}^a(W_k)| \leq 4L^{2\Delta_G + 2} \cdot L^3 + 4L^{2\Delta_G - 2} = O(L^{2\Delta_G + 5}). \end{aligned}$$

□

Combining Theorem 6 and Theorem 3 leads to the following result:

Theorem 7. *Let (q_2, \dots, q_n) be an integer embedding of a simplicial 3d polytope with graph G_\uparrow , such that the orthogonal projection into the plane $\{z = 0\}$ gives a planar 2d embedding (p_2, \dots, p_n) with boundary face $(v_2v_3v_4)$. Then we can construct an embedding $(\phi_f)_{f \in F(G)}$ of a graph dual to $G = \text{Stack}(G_\uparrow; v_1; v_2v_3v_4)$ with integer coordinates bounded by*

$$|\phi_f| = O(n^2 \max |q_i|^{2\Delta_G + 13}).$$

We remark that the algorithms following the lifting approach generate embeddings that fulfill the conditions of the above theorem. Using a more technical analysis we can even show that the following stronger version of Theorem 7 holds. The proof of the theorem can be found in the full version of the paper.

Theorem 8. *Let G be a triangulation with at least one vertex of degree 3, and let (q_i) be an integer realization of G as a convex polytope. Then there is a realization $(\phi_f)_{f \in F(G)}$ of the dual graph G^* as a convex polytope with integer coordinates bounded by*

$$|\phi_f| < \max |q_i|^{O(\Delta_G)}.$$

References

1. Andrews, G.E.: A lower bound for the volume of strictly convex bodies with many boundary lattice points. *Trans. Amer. Math. Soc.* 99, 272–277 (1961)
2. Bárány, I., Rote, G.: Strictly convex drawings of planar graphs. *Documenta Math.* 11, 369–391 (2006)
3. Buchin, K., Schulz, A.: On the number of spanning trees a planar graph can have. In: de Berg, M., Meyer, U. (eds.) *ESA 2010, Part I. LNCS*, vol. 6346, pp. 110–121. Springer, Heidelberg (2010)
4. Das, G., Goodrich, M.T.: On the complexity of optimization problems for 3-dimensional convex polyhedra and decision trees. *Computational Geometry: Theory and Applications* 8(3), 123–137 (1997)
5. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. *Combinatorica* 10(1), 41–51 (1990)
6. Demaine, E.D., Schulz, A.: Embedding stacked polytopes on a polynomial-size grid. In: *Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1177–1187. ACM Press (2011)
7. Eades, P., Garvan, P.: Drawing stressed planar graphs in three dimensions. In: Brandenburg, F.J. (ed.) *GD 1995. LNCS*, vol. 1027, pp. 212–223. Springer, Heidelberg (1996)
8. Hopcroft, J.E., Kahn, P.J.: A paradigm for robust geometric algorithms. *Algorithmica* 7(4), 339–380 (1992)
9. Lovász, L.: Steinitz representations of polyhedra and the Colin de Verdière number. *J. Comb. Theory, Ser. B* 82(2), 223–236 (2001)
10. Maxwell, J.C.: On reciprocal figures and diagrams of forces. *Phil. Mag. Ser. 27*, 250–261 (1864)
11. Mor, A.R., Rote, G., Schulz, A.: Small grid embeddings of 3-polytopes. *Discrete & Computational Geometry* 45(1), 65–87 (2011)
12. Onn, S., Sturmfels, B.: A quantitative Steinitz’ theorem. *Beiträge zur Algebra und Geometrie* 35, 125–129 (1994)
13. Richter-Gebert, J.: *Realization Spaces of Polytopes. Lecture Notes in Mathematics*, vol. 1643. Springer (1996)
14. Rote, G., Santos, F., Streinu, I.: Expansive motions and the polytope of pointed pseudo-triangulations. *Discrete and Computational Geometry—The Goodman-Pollack Festschrift* 25, 699–736 (2003)
15. Schulz, A.: Drawing 3-polytopes with good vertex resolution. *Journal of Graph Algorithms and Applications* 15(1), 33–52 (2011)
16. Steinitz, E.: *Polyeder und Raumeinteilungen*. In: *Encyclopädie der mathematischen Wissenschaften*, vol. 3-1-2 (Geometrie), ch. 12, pp. 1–139. B.G. Teubner, Leipzig (1916)
17. Whitney, H.: Congruent graphs and the connectivity of graphs. *Amer. J. Math.* 54, 150–168 (1932)

Block Additivity of \mathbb{Z}_2 -Embeddings

Marcus Schaefer¹ and Daniel Štefankovič²

¹ School of Computing, DePaul University, Chicago, IL 60604
mschaefer@cs.depaul.edu

² Computer Science Department, University of Rochester, Rochester, NY 14627-0226
stefanko@cs.rochester.edu

Abstract. We study embeddings of graphs in surfaces up to \mathbb{Z}_2 -homology. We introduce a notion of genus mod 2 and show that some basic results, most noteworthy block additivity, hold for \mathbb{Z}_2 -genus. This has consequences for (potential) Hanani-Tutte theorems on arbitrary surfaces.

1 Introduction

A graph G *embeds* in a surface \mathcal{S} if it can be drawn in \mathcal{S} so that no pair of edges cross. In this paper we want to relax the embedding condition using \mathbb{Z}_2 -homology, that is, we are only interested in the parity of the number of crossings between independent edges; in terms of algebraic topology we are studying the “mod 2 homology of the deleted product of the graph” [1]. We say a graph \mathbb{Z}_2 -*embeds* in \mathcal{S} if it can be drawn in \mathcal{S} so that every pair of independent edges crosses evenly. This approach is inspired by two Hanani-Tutte theorems which, for the plane [2] and the projective plane [3], show that embeddability is equivalent to \mathbb{Z}_2 -embeddability. For other surfaces, it is only known that \mathbb{Z}_2 -embeddability is a necessary condition for embeddability. In this paper we want to lay the foundations for a study of \mathbb{Z}_2 -embeddings of graphs in surfaces which may, at some point, lead to a proof of the Hanani-Tutte theorem for arbitrary surfaces. Our main result is that if we define the notion of \mathbb{Z}_2 -genus as a homological invariant of \mathbb{Z}_2 -embeddings, then block additivity holds for \mathbb{Z}_2 -genus just as it does for the standard notion of genus (as proved by Battle, Harary, Kodama and Youngs for the orientable case, and by Stahl and Beineke in the non-orientable case, see [4, Section 4.4]). This implies that a counterexample to the Hanani-Tutte theorem on an arbitrary surface can be assumed to be 2-connected (Corollary 1).

2 \mathbb{Z}_2 -Embeddings

2.1 Definition

In the introduction we defined a \mathbb{Z}_2 -embedding in a surface \mathcal{S} , as a drawing of a graph G in which every pair of independent edges crosses evenly. In this section, we want to develop a more algebraic version of this definition, which separates the topology of the surface from the drawing. We start with the plane, and then add crosscaps and handles.

Pick an initial drawing D of $G = (V, E)$ in the plane. For edges $e, f \in E$ let $i_D(e, f)$ be the number of crossings of e and f in the drawing D . We want to extend the drawing to a surface \mathcal{S} with c crosscaps. Since we only plan to keep track of the parity of the number of crossings of independent edges and hence we use \mathbb{Z}_2 -homology. For each edge we have a vector $y_e \in \mathbb{Z}_2^c$ where $(y_e)_i = 1$ if e is pulled through the i -th crosscap an odd number of times (in a drawing, we can deform e so it passes through the i -th crosscap; this changes the crossing parity of e with any edge that passes through the i -th crosscap an odd number of times). We also allow changing the planar part of the drawing—for each edge we have a vector $x_e \in \mathbb{Z}_2^V$ where $(x_e)_v = 1$ indicates that we made an (e, v) -move, that is, we pull the edge e over v (this changes the crossing parity between e and any edge incident to v). We say that the initial drawing together with $\{x_e\}_{e \in E}$ and $\{y_e\}_{e \in E}$ is a \mathbb{Z}_2 -embedding of G in \mathcal{S} if for each pair of independent edges $e = \{u, v\}, f = \{s, t\}$ we have

$$i_D(e, f) + (x_e)_s + (x_e)_t + (x_f)_u + (x_f)_v + y_e^T y_f \equiv 0 \pmod{2}. \tag{1}$$

All congruences in this paper are modulo 2, so to simplify notation, we drop $(\text{mod } 2)$ from now on. See Figure 1 for a \mathbb{Z}_2 -embedding of K_5 in the projective plane, illustrating the effect of crosscap- and (e, v) -moves. This definition is equivalent to the more intuitive definition given in the introduction (see, for example, Levow [5, Theorem 3]). We say that the drawing is *orientable* if $y_e^T y_e \equiv 0$ for every $e \in E$ (that is, every y_e has an even number of ones).

Handles can be dealt with in the following way: For each handle we have three coordinates in y_e and the possible settings for these coordinates are 000, 110, 101, and 011. We extend the definitions of \mathbb{Z}_2 -embeddings and orientability given earlier to y_e containing handles. Note that each of the four vectors modeling an edge passing through handle has an even number of ones, so if the surface contains only handles, then any drawing on it is orientable by the earlier definition.

A handle and a crosscap are equivalent to three crosscaps (Dyck, see [6, Section 1.2.4]): in the \mathbb{Z}_2 -homology this corresponds to the following bijection (we replace the $3 + 1$ coordinates in y_e by 3 coordinates in y_e):

$$\begin{aligned} 0000 &\leftrightarrow 000, 0001 \leftrightarrow 111, 0110 \leftrightarrow 011, 0111 \leftrightarrow 100, \\ 1100 &\leftrightarrow 110, 1101 \leftrightarrow 001, 1010 \leftrightarrow 101, 1011 \leftrightarrow 010, \end{aligned} \tag{2}$$

where the first three coordinates on the left hand sides correspond to the handle. Note that (2) preserves the parity of the number of ones (and thus the orientability of the drawing). Also note that (2) is linear (add vector 111 times the last coordinate to the vector of the first three coordinates) and hence preserves the dimension of the space generated by the $\{y_e\}_{e \in E}$.

Remark 1 (\mathbb{Z}_2 -drawings). Call a drawing D of a graph G in the plane together with $\{x_e\}_{e \in E}$ and $\{y_e\}_{e \in E}$ a \mathbb{Z}_2 -drawing, and define $i_{D,x,y}(e, f) := i_D(e, f) + (x_e)_s + (x_e)_t + (x_f)_u + (x_f)_v + y_e^T y_f$. With this notion of \mathbb{Z}_2 -drawing, we can model drawings of graphs in a surface up to the \mathbb{Z}_2 -homology we are interested in: If D is

crosscaps into handles, the single crosscap left at the end is not used by any edge and hence can be discarded.

With this terminology, we can now define \mathbb{Z}_2 -homological variants of the genus and the Euler genus of a graph. We write $\mathbf{g}(G)$ and $\mathbf{eg}(G)$ for the traditional genus and Euler genus of G (following [4]).

Definition 1 (\mathbb{Z}_2 -genus and \mathbb{Z}_2 -Euler genus). *If a graph G has a \mathbb{Z}_2 -embedding in an orientable surface with h handles, but not in any surface with fewer handles, we write $\mathbf{g}_0(G) = h$ and call h the \mathbb{Z}_2 -genus of G . If G has a \mathbb{Z}_2 -embedding in a surface \mathcal{S} with c crosscaps and h handles, but not in any surface with a smaller value of $2h + c$, we write $\mathbf{eg}_0(G) = 2h + c$ and call $2h + c$ the \mathbb{Z}_2 Euler genus of G .*

By definition, we have $\mathbf{g}_0(G) \leq \mathbf{g}(G)$ and $\mathbf{eg}_0(G) \leq \mathbf{eg}(G)$, where $\mathbf{g}(G)$ is the genus of G and $\mathbf{eg}(G)$ is the Euler genus of G .

2.2 Basic Properties

We derive some basic properties of \mathbb{Z}_2 -embeddings. We call two graphs G and H *disjoint* if $V(G) \cap V(H) = \emptyset$.

Lemma 1. *Let G be a graph \mathbb{Z}_2 -embedded in a (possibly non-orientable) surface \mathcal{S} . Let C_1 and C_2 be two disjoint cycles in G . Then*

$$\sum_{e \in C_1, f \in C_2} y_e^T y_f \equiv 0. \tag{3}$$

Let $e_1 \in C_1$ and $e_2 \in C_2$. Suppose that all edges e in $C_1 \setminus \{e_1\}$ have $y_e = 0$ and all edges $f \in C_2 \setminus \{e_2\}$ have $y_f = 0$. Then

$$y_{e_1}^T y_{e_2} \equiv 0. \tag{4}$$

Proof. We have

$$\sum_{e \in C_1, f \in C_2} (i_D(e, f) + (x_e)_s + (x_e)_t + (x_f)_u + (x_f)_v + y_e^T y_f) \equiv 0, \tag{5}$$

where s, t are endpoints of f and u, v are endpoints of e (note that s, t, u and v vary over the terms in the sum). The equality in (5) follows from the disjointness of C_1 and C_2 (any $e \in C_1$ and $f \in C_2$ are independent and hence (1) has to be satisfied).

We have

$$\sum_{e \in C_1, f \in C_2} (i_D(e, f) + (x_e)_s + (x_e)_t + (x_f)_u + (x_f)_v) \equiv 0, \tag{6}$$

since (6) corresponds to a drawing in the plane (and two transversally intersecting cycles in the plane intersect evenly; the cycles have to intersect transversally since they are disjoint).

Combining (5) with (6) we obtain (3). Equation (4) is an immediate corollary (since terms in (3) other than $y_{e_1}^T y_{e_2}$ are zero). \square

Lemma 1 suggests that orthogonality plays a role in understanding \mathbb{Z}_2 -embeddings. We will need the following fact about vector spaces over finite fields (see [8, Section 2.3]).

Lemma 2. *Let A be a subspace of \mathbb{Z}_2^t . Then for $A^\perp := \{x \in \mathbb{Z}_2^t \mid (\forall y \in A) x^T y \equiv 0\}$ we have*

$$\dim A + \dim A^\perp = t.$$

Let A, B be subspaces of \mathbb{Z}_2^t with $A \subseteq B$. Then for $A^{\perp B} := \{x \in B \mid (\forall y \in A) x^T y \equiv 0\} = A^\perp \cap B$ we have

$$\dim A + \dim A^{\perp B} = \dim B + \dim \text{rad } B,$$

where $\text{rad } B := B^{\perp B}$ is the radical of B .

The dimension of the \mathbb{Z}_2 -embedding (the dimension of the space spanned by $\{y_e\}_{e \in E}$) is closely related to its \mathbb{Z}_2 -genus. In Lemma 3 we extend this result to \mathbb{Z}_2 -Euler genus.

Lemma 3. *Let G be a graph \mathbb{Z}_2 -embedded in a (possibly non-orientable) surface \mathcal{S} . Assume that the drawing is orientable. Let d be the dimension (over \mathbb{Z}_2) of the vector space generated by the edge vectors $\{y_e\}_{e \in E}$. Then G can be \mathbb{Z}_2 -embedded in an orientable surface of genus $\lfloor d/2 \rfloor$.*

Proof. In the light of Remark 2 we can assume that \mathcal{S} has t crosscaps (and no handles). We are going to remove the crosscaps from \mathcal{S} one by one. Let S be the space generated by the edge vectors $\{y_e\}_{e \in E}$. Let $d = \dim S$. Let $T = S^\perp$ be the space of vectors that are perpendicular to S . Assume that T contains a vector z such that $z \neq 0$ and $z^T z \equiv 0$ (the computations are in \mathbb{Z}_2). Rearranging coordinates, if necessary, we can assume that $z_1 = 1$. To each y_e for which $(y_e)_1 = 1$ we add z . This transformation has the following properties:

- it is linear ($y \mapsto y + y_1 z$) and hence the dimension of S cannot increase,
- it preserves orientability (since $(y + z)^T (y + z) \equiv y^T y$),
- it preserves the parity of the number of crossings for every independent pair of edges e, f (since $(y_e + z)^T (y_f + z) \equiv y_e^T y_f$ and $(y_e + z)^T y_f \equiv y_e^T y_f$; here we use the fact that $T = S^\perp$).

After the transformation, the first crosscap is not used by any edge and hence we can remove it thus decreasing t . We repeat the crosscap removal process as long as such a z exists. We distinguish three cases depending on whether the process stops with $d \leq t - 2$, $d = t - 1$ or $d = t$.

If $d \leq t - 2$, then T always contains $z \neq 0$ with $z^T z \equiv 0$ (since by a dimension argument there are two distinct vectors z_1, z_2 in $T \setminus \{0\}$ and then one of $z_1, z_2, z_1 + z_2$ satisfies $z^T z \equiv 0$) and hence we can always remove a crosscap in this case. Therefore, the process ends up with either $d = t$ or $d = t - 1$, $T = \langle z \rangle$, and $z^T z \equiv 1$. If $d = t$, then we convert the crosscaps back into $\lfloor d/2 \rfloor$ handles (if t is odd we end up with $(t - 1)/2$ handles, if t is even we add an extra crosscap

and end up with $t/2$ handles, in both cases use Remark 2 on being able to drop a crosscap in an orientable embedding).

The final case to handle is $d = t - 1$. Let k be the number of ones in z . W.l.o.g. the first k coordinates of z are 1 and the rest are 0. Note that k is odd and that $z^T y_e \equiv 0$ for each e (that is, if we restrict our attention to the first k crosscaps, the drawing is orientable). Hence we can convert the first k crosscaps into $(k - 1)/2$ handles. Then—as in the $d = t$ case—we convert the remaining $t - k$ crosscaps into $\lfloor (t - k)/2 \rfloor$ handles. In total, we have

$$(k - 1)/2 + \lfloor (t - k)/2 \rfloor = \lfloor (t - 1)/2 \rfloor = \lfloor d/2 \rfloor$$

handles. □

For non-orientable surfaces we have the following analogue of Lemma 3, replacing the notion of genus by Euler genus.

Lemma 4. *Let G be a graph \mathbb{Z}_2 -embedded in a (possibly non-orientable) surface \mathcal{S} . Let d be the dimension (over \mathbb{Z}_2) of the vector space generated by the edge vectors $\{y_e\}_{e \in E}$. Then G can be \mathbb{Z}_2 -embedded in a (possibly non-orientable) surface of Euler genus d .*

Proof. The proof is almost the same as the proof of Lemma 3. We first convert handles to crosscaps and work on a surface with crosscaps only. We again remove crosscaps one by one until we end up with $d = t$ or with $d = t - 1$, $T = \langle z \rangle$, and $z^T z \equiv 1$. In the case that $d = t$ we are done.

In the case $d = t - 1$ we assume, as in the proof of Lemma 3, that the first k coordinates of z are 1 and the rest are 0 and convert the first k crosscaps into $(k - 1)/2$ handles. We leave the remaining crosscaps as they are. The Euler genus of the resulting surface is $2((k - 1)/2) + t - k = t - 1 = d$. □

We end this section with a more complex move that allows us to zero out the labels of all edges in a spanning forest.

Lemma 5. *Suppose G is \mathbb{Z}_2 -embedded on a surface \mathcal{S} , and F is a spanning forest of G . Then there is a \mathbb{Z}_2 -embedding of G on \mathcal{S} in which all edges of F are labeled with zero vectors.*

Proof. By Remark 2 we can assume that \mathcal{S} is a surface with $c > 0$ crosscaps. Choose $z \in \mathbb{Z}_2^c$ and $v \in V$. Consider the following collection of moves: 1) add z to y_e for all e that are adjacent to v , and 2) for every f not adjacent to v and so that $y_f^T z \equiv 1$ perform an (f, v) -move. This collection of moves preserves the parity of the number of crossings between any pair of independent edges. Moreover, if z contains an even number of ones, the parity of the number of ones in no y -label is changed. Pick a root for each component of F , orient the edges of F away from the root, and process the edges in each component in a breadth-first traversal; for each edge e in this traversal, we turn its label into the zero vector, by performing the collection of moves above with $z = y_e$ and v the head of e . Note that this changes the label of e into the zero vector, without

affecting the labels of any edges that have already been processed (since F is a forest, and $y_f^T z \equiv 0$ for edges f already processed, because $y_f = 0$ for those edges). If the \mathbb{Z}_2 -embedding was orientable to begin with, it remains so, since z is chosen from the set of existing labels, all of which contain an even number of ones originally and throughout the relabeling. \square

3 Block Additivity mod \mathbb{Z}_2

As a warm-up we show the additivity of genus over connected components (a result that is nearly obvious for embeddings).

Lemma 6. *The \mathbb{Z}_2 -genus of a graph is the sum of the \mathbb{Z}_2 -genera of its connected components.*

Proof. Let G be a graph. Let $g := \mathbf{g}_0(G)$ be the \mathbb{Z}_2 -genus of G . By Remark 2 we have an orientable drawing of G on the surface with $t := 2g + 1$ crosscaps. Assume that G is the disjoint union of G_1 and G_2 . Let F_1 be a maximum spanning forest of G_1 and F_2 be a maximum spanning forest of G_2 . We can assume (see Lemma 5) that the y_e -labels for edges e in F_1 and F_2 are zero.

Let e_1 be an edge in $G_1 - F_1$ and let e_2 be an edge in $G_2 - F_2$. Let C_1 be the unique cycle in $F_1 + e_1$ and let C_2 be the unique cycle in $F_2 + e_2$. Note that C_1 and C_2 are disjoint (since G_1 and G_2 are disjoint) and hence by Lemma 1 we have

$$y_{e_1}^T y_{e_2} \equiv 0; \tag{7}$$

that is, the vectors y_{e_1} and y_{e_2} are perpendicular. Let S_1 be the vector space generated by the y_e -labels on the edges in G_1 and let S_2 be the vector space generated by the y_e -labels on the edges in G_2 . Then $S_1 \perp S_2$ and hence

$$\dim S_1 + \dim S_2 \leq t = 2g + 1. \tag{8}$$

By Lemma 3, we can \mathbb{Z}_2 -embed G_i in an orientable surface with $\lfloor (\dim S_i)/2 \rfloor$ handles. Note

$$\lfloor (\dim S_1)/2 \rfloor + \lfloor (\dim S_2)/2 \rfloor \leq g$$

and hence $\mathbf{g}_0(G_1) + \mathbf{g}_0(G_2) \leq \mathbf{g}_0(G)$. \square

Again, one also has the analogue of Lemma 6 for non-orientable surfaces.

Lemma 7. *The \mathbb{Z}_2 -Euler genus of a graph is the sum of the \mathbb{Z}_2 -Euler genera of its connected components.*

Proof. The proof is the same as the proof of Lemma 6 except the final part. We have

$$\dim S_1 + \dim S_2 \leq t, \tag{9}$$

where $t := \mathbf{eg}_0(G)$ is the \mathbb{Z}_2 -Euler genus of G . By Lemma 4 we can draw G_i in a surface with $\dim S_i$ crosscaps. Hence $\mathbf{eg}_0(G_1) + \mathbf{eg}_0(G_2) \leq \mathbf{eg}_0(G)$. \square

We are ready now to establish additivity of \mathbb{Z}_2 -genus and \mathbb{Z}_2 -Euler genus over 2-connected components (blocks).

Theorem 1. *The \mathbb{Z}_2 -genus of a graph is the sum of the \mathbb{Z}_2 -genera of its blocks. The \mathbb{Z}_2 -Euler genus of a graph is the sum of the \mathbb{Z}_2 -Euler genera of its blocks.*

Proof. There is a large shared part in the arguments for \mathbb{Z}_2 -genus and \mathbb{Z}_2 -Euler genus (only the initial setup and the final drawing step are different).

The initial setup for the \mathbb{Z}_2 -genus case is the following. Let $G = (V, E)$ be a connected graph (we can assume this by Lemma 6) and let $g := \mathbf{g}_0(G)$ be the \mathbb{Z}_2 -genus of G . Thus we have an orientable \mathbb{Z}_2 -embedding of G on the surface with $t := 2g + 1$ crosscaps. Let B be the subspace of \mathbb{Z}_2^t consisting of vectors with an even number of ones (we will keep our drawing orientable, that is, all the edge labels will be from B). Note that

$$\text{rad } B = \{0\}, \tag{10}$$

since each vector in $B \setminus \{0\}$ has a zero and a one. Let $\hat{t} := \dim B = t - 1$.

The initial setup for the \mathbb{Z}_2 -Euler genus case is the following. Let $G = (V, E)$ be a connected graph (Lemma 7), and let $g := \mathbf{eg}_0(G)$ be the \mathbb{Z}_2 -Euler genus of G . Thus we have a \mathbb{Z}_2 -embedding of G on the surface with t crosscaps. In this case we let $B := \mathbb{Z}_2^t$. (And we trivially have (10).) Let $\hat{t} := \dim B = t$.

Let v be a cut vertex of G . Let $G_1 = (V_1, E_1)$ be a block of G containing v and let $G_2 = (V_2, E_2)$ be the union of the remaining blocks (note that $V_1 \cap V_2 = \{v\}$). Let $T = (V, F)$ be a depth-first search (DFS) spanning tree of G with the exploration starting at v . We can assume $y_e = 0$ for the edges in F (see Lemma 5). The reason for taking a DFS spanning tree is that we will need the following property: if $e \in E \setminus F$ is not adjacent to v then the unique cycle in $F + e$ does not contain v .

Let S_i be the vector space generated by the y_e -labels of $e \in E_i$ that are not adjacent to v . Let Z_i be the vector space generated by the y_e -labels of $e \in E_i$ that are adjacent to v . See Figure 2. Our plan is to modify the y_e -labels of the edges adjacent to v (changing Z_1, Z_2) so that: 1) no new odd crossings between independent edges are introduced, and 2) after the modification $\dim(S_1 + Z_1) + \dim(S_2 + Z_2) \leq t$.

We can modify the y_e of an edge e adjacent to v by adding any vector in $O := (S_1 + S_2)^\perp$. This does not change the parity of the number of crossings between independent pairs of edges (for f that are not adjacent to v we have $y_f^T(y_e + z) \equiv y_f^T y_e$; and f which are adjacent to v are not independent of e). Note that the modification in the \mathbb{Z}_2 -genus case also preserves orientability (by choice of B).

We are going to modify the y_e -labels of edges adjacent to v (by adding vectors in O) as follows. For $i \in \{1, 2\}$ we do the following. Let $a := \dim(Z_i \cap S_i)$, $b := \dim(Z_i \cap (S_i + O))$, and $c := \dim(Z_i)$. Let z_1, \dots, z_c be a basis of Z_i such that 1) z_1, \dots, z_a is a basis of $Z_i \cap S_i$, 2) z_1, \dots, z_b is a basis of $Z_i \cap (S_i + O)$, and 3) $z_{a+1}, \dots, z_b \in Z_i \cap O$. Such a basis can be constructed as follows: first apply the Steinitz exchange lemma on a basis of $Z_i \cap S_i$ and a basis of $Z_i \cap O$ (the

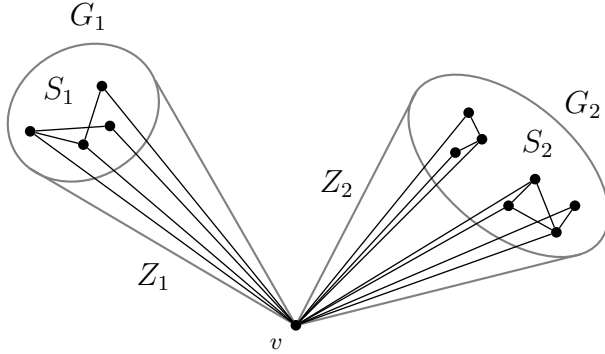


Fig. 2. Graph G with cutvertex v , block G_1 and union of remaining blocks G_2 . Vector space Z_i (S_i) is generated by labels of edges in G_i (not) incident to v .

basis of $Z_i \cap S_i$ will be extended by vectors in the basis of $Z_i \cap O$ to a basis of $Z_i \cap (O + S_i)$; then apply the Steinitz exchange lemma on the resulting basis and a basis of Z_i . For each edge in G_i adjacent to v we relabel $y_e = \alpha_1 z_1 + \dots + \alpha_c z_c$ by setting $\alpha_{a+1} = \alpha_{a+2} = \dots = \alpha_b = 0$ (note that this corresponds to adding an element of O to y_e). After the modification (which also changed Z_i) we have that z_1, \dots, z_a is a basis of $Z_i \cap (S_i + O)$ and also a basis of $Z_i \cap S_i$. Thus we have

$$Z_i \cap (S_i + O) = Z_i \cap S_i. \tag{11}$$

Let $e_1 \in E_1 \setminus F$ and let $e_2 \in E_2 \setminus F$ be such that e_2 is not adjacent to v . Let C_1 be the unique cycle in $F + e_1$ and let C_2 be the unique cycle in $F + e_2$. Note that C_2 does not contain v (since F is a DFS spanning tree and e_2 is not adjacent to v). Thus C_1 and C_2 are disjoint (C_1 is in G_1 , C_2 is in G_2 and $V_1 \cap V_2 = \{v\}$) and hence, by Lemma 1, we have

$$y_{e_1}^T y_{e_2} \equiv 0.$$

Thus $(Z_1 + S_1) \subseteq S_2^{\perp B}$ and since $O \subseteq S_2^{\perp B}$ (by the definition of O) we also have $(Z_1 + S_1 + O) \subseteq S_2^{\perp B}$. By symmetry we also have $(Z_2 + S_2 + O) \subseteq S_1^{\perp B}$ and hence, by Lemma 2 and (10), we obtain

$$\dim(Z_i + S_i + O) + \dim(S_{3-i}) \leq \hat{t}. \tag{12}$$

Thus we have (using $\dim(A + B) = \dim(A) + \dim(B) - \dim(A \cap B)$)

$$\dim(Z_i) + \dim(S_i + O) - \dim(Z_i \cap (S_i + O)) + \dim(S_{3-i}) \leq \hat{t},$$

and (11) yields

$$\dim(Z_i) + \dim(S_i + O) - \dim(Z_i \cap S_i) + \dim(S_{3-i}) \leq \hat{t}. \tag{13}$$

Adding (13) for $i = 1, 2$ and simplifying (again using $\dim(A + B) = \dim(A) + \dim(B) - \dim(A \cap B)$) we obtain

$$\dim(Z_1 + S_1) + \dim(Z_2 + S_2) + \dim(S_1 + O) + \dim(S_2 + O) \leq 2\hat{t}. \tag{14}$$

We have

$$\begin{aligned}
 & \dim(S_1 + O) + \dim(S_2 + O) \\
 &= \dim(S_1) + \dim(O) - \dim(S_1 \cap O) + \dim(S_2) + \dim(O) - \dim(S_2 \cap O) \\
 &= \dim(S_1 + S_2) + \dim(S_1 \cap S_2) + 2 \dim(O) - \dim(S_1 \cap O) - \dim(S_2 \cap O) \\
 &= \hat{t} + \dim(S_1 \cap S_2) + \dim(O) - \dim(S_1 \cap O) - \dim(S_2 \cap O) \\
 &\geq \hat{t} + \dim(S_1 \cap S_2 \cap O) + \dim(O) - \dim(S_1 \cap O) - \dim(S_2 \cap O) \\
 &= \hat{t} + \dim(O) - \dim((S_1 + S_2) \cap O) \geq \hat{t},
 \end{aligned}$$

where in the first, second, and fourth equality we used $\dim(A+B) + \dim(A \cap B) = \dim(A) + \dim(B)$; in the third equality we used $\dim(S_1 + S_2) + \dim(O) = \hat{t}$ (which follows from the definition of O and Lemma 2); in the first and the last inequality we used the monotonicity of dimension,

Plugging $\dim(S_1 + O) + \dim(S_2 + O) \geq \hat{t}$ into (14) we obtain

$$\dim(Z_1 + S_1) + \dim(Z_2 + S_2) \leq \hat{t} \leq t. \quad (15)$$

The \mathbb{Z}_2 -genus case of the lemma now follows from Lemma 3, using the argument from the proof of Lemma 6 (the final part after equation (8)) giving us \mathbb{Z}_2 -embeddings of G_1 and G_2 on two surfaces which have g handles total. In the \mathbb{Z}_2 -Euler genus case we apply Lemma 4, using the argument from the proof of Lemma 7 (the final part after equation (9)). \square

The Hanani-Tutte theorem for surface \mathcal{S} would—if true—state that if a graph has a \mathbb{Z}_2 -embedding on surface \mathcal{S} , then it can be embedded on \mathcal{S} . Theorem 1 implies that in a search for counterexamples we can concentrate on 2-connected graphs. For the projective plane this result was obtained using much simpler means in [3, Lemma 2.4].

Corollary 1. *A minimal counterexample to the Hanani-Tutte theorem on any surface \mathcal{S} is 2-connected.*

Proof. Suppose G is a minimal counterexample to Hanani-Tutte on some surface \mathcal{S} . If G is not connected, let G_1, \dots, G_k , $k \geq 2$ be its connected components. If \mathcal{S} is orientable, let g be the genus of \mathcal{S} . Then $g \geq \mathbf{g}_0(G) = \sum_{i=1}^k \mathbf{g}_0(G_i) = \sum_{i=1}^k \mathbf{g}(G_i) = \mathbf{g}(G)$, where the first equality is true by Lemma 6, and the second equality because G is minimal (and the third is a standard property of the genus of a graph). Therefore, $\mathbf{g}(G) \leq g$, so G can be embedded in \mathcal{S} , meaning it cannot be a counterexample. If \mathcal{S} is non-orientable we can make essentially the same argument with Lemma 7 replacing Lemma 6, and Euler genus replacing genus. If G is connected, but not 2-connected, we repeat the same argument with Theorem 1 replacing the two lemmas, and using block additivity of (Euler) genus to conclude that $\sum_{i=1}^k \mathbf{g}(G_i) = \mathbf{g}(G)$ where the G_i are the blocks of G having cutvertex v in common. \square

4 Questions

The (Euler) genus of a graph is an obvious upper bound on the \mathbb{Z}_2 -(Euler) genus of a graph, but are they always the same?

Conjecture 1. The \mathbb{Z}_2 -(Euler) genus of a graph equals its (Euler) genus.

The truth of this conjecture would imply the Hanani-Tutte theorem for arbitrary surfaces, so we have to leave the question open. The block additivity result from Section 3 implies that a minimal counterexample to the conjecture (if it exists) and, thereby, to the Hanani-Tutte theorem on an arbitrary surface, can be assumed to be 2-connected (since it cannot have a cut-vertex).

A much more modest goal than Conjecture 1 would be to bound the standard (Euler) genus in the \mathbb{Z}_2 -(Euler) genus: Are there functions f and g so that $\mathbf{g}(G) \leq f(\mathbf{g}_0(G))$ and $\mathbf{eg}(G) \leq g(\mathbf{eg}_0(G))$?

In the absence of a proof of Conjecture 1, we can ask what other results for (Euler) genus also hold for \mathbb{Z}_2 -(Euler) genus. For example, is the computation of the \mathbb{Z}_2 -(Euler) genus **NP**-hard (as it is for (Euler) genus [9])? And is \mathbb{Z}_2 -embeddability decidable in polynomial time for a fixed surface \mathcal{S} (as it is for embeddability [10])? One could also try to extend the block additivity result: if G_1 and G_2 are two edge-disjoint graphs with $|V(G_1) \cap V(G_2)| = 2$, is it true that $|\mathbf{g}_0(G) - (\mathbf{g}_0(G_1) + \mathbf{g}_0(G_2))| \leq 1$? (This inequality is known to be true for the standard genus, a result by Decker, Glover, Huneke, and Stahl, see [4, Section 4.4]).

References

1. Sarkaria, K.S.: A one-dimensional Whitney trick and Kuratowski's graph planarity criterion. *Israel J. Math.* 73(1), 79–89 (1991)
2. Chojnacki (Haim Hanani), C.: Über wesentlich unplättbare Kurven im dreidimensionalen Raume. *Fundamenta Mathematicae* 23, 135–142 (1934)
3. Pelsmajer, M.J., Schaefer, M., Stasi, D.: Strong Hanani–Tutte on the projective plane. *SIAM Journal on Discrete Mathematics* 23(3), 1317–1323 (2009)
4. Mohar, B., Thomassen, C.: *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore (2001)
5. Levow, R.B.: On Tutte's algebraic approach to the theory of crossing numbers. In: Hoffman, F., Levow, R.B., Thomas, R.S.D. (eds.) *Proceedings of the Third Southeastern Conference on Combinatorics, Graph Theory and Computing*, Boca Raton, Fla., Florida Atlantic University, pp. 315–314 (1972)
6. Stillwell, J.: *Classical topology and combinatorial group theory*. Second edn, 2nd edn. *Graduate Texts in Mathematics*, vol. 72. Springer, New York (1993)
7. de Longueville, M.: *A course in topological combinatorics*. In: *Universitext*, Springer, New York (2013)
8. Babai, L., Frankl, P.: *Linear algebra methods in combinatorics with applications to geometry and computer science*. Department of Computer Science, The University of Chicago (1992)
9. Thomassen, C.: The graph genus problem is NP-complete. *J. Algorithms* 10(4), 568–576 (1989)
10. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.* 12(1), 6–26 (1999)

Exploiting Air-Pressure to Map Floorplans on Point Sets

Stefan Felsner*

Institut für Mathematik, Technische Universität Berlin, Germany
felsner@math.tu-berlin.de

Abstract. We prove a conjecture of Ackerman, Barequet and Pinter. Every floorplan with n segments can be embedded on every set of n points in generic position. The construction makes use of area universal floorplans also known as area universal rectangular layouts.

The notion of area used in our context depends on a nonuniform density function. We, therefore, have to generalize the theory of area universal floorplans to this situation. The method is then used to prove a result about accommodating points in floorplans that is slightly more general than the conjecture of Ackerman et al.

1 Introduction

In our context a *floorplan* is a partition of a rectangle into a finite set of interior-disjoint rectangles. A floorplan is *generic* if it has no cross, i.e., no point where four rectangles of the partition meet. A *segment* of a floorplan is a maximal nondegenerate interval that belongs to the union of the boundaries of the rectangles. Segments are either horizontal or vertical. The segments of a generic floorplan are internally disjoint. Two floorplans F and F' are *weakly equivalent* if there exist bijections $\phi : S_H(F) \rightarrow S_H(F')$ and $\psi : S_V(F) \rightarrow S_V(F')$ between their horizontal and vertical segments such that segment s has an endpoint on segment t in F iff $\phi(s)$ has an endpoint on $\psi(t)$. A set P of points in \mathbb{R}^2 is *generic* if no two points from P have the same x or y coordinate. Section 2 provides a more comprehensive overview of definitions and notions related to floorplans.

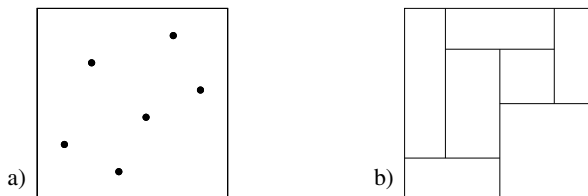


Fig. 1. A generic set of six points and a generic floorplan with six segments

Let P be a set of n points in a rectangle R and let F be a generic floorplan with n internal segments. A *cover map* from F to P is a floorplan F' weakly equivalent to F with outer rectangle R such that every internal segment of F' contains exactly one point from P . Figure 2 shows an example.

* Partially supported by ESF EuroGIGA projects Compose and GraDR.

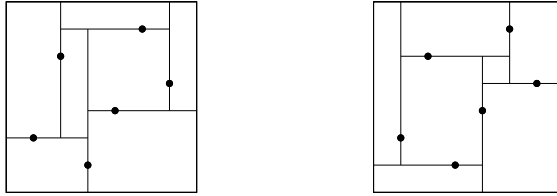


Fig. 2. Two cover maps from the floorplan of Fig. 1.b to the point set of Fig. 1.a

In this paper we answer a question of Ackerman et al. [1] by proving Theorem 1. The proof of the theorem and some variants and generalizations is the subject of Section 4.

Theorem 1. *If P is a generic set of n points and F is a generic floorplan with n internal segments, then there is a cover map from F to P .*

The proof is based on results about area representations of floorplans. The following theorem is known, it has been proven with quite different methods, see [14], [11], [5].

Theorem 2. *Let F be a floorplan with rectangles R_1, \dots, R_{n+1} , let A be a rectangle and let $w : \{1, \dots, n+1\} \rightarrow \mathbb{R}_+$ be a weight function with $\sum_i w(i) = \text{area}(A)$. There exists a unique floorplan F' contained in A that is weakly equivalent to F such that the area of the rectangle $R'_i = \phi(R_i)$ is exactly $w(i)$.*

In Section 3 we prove the generalization of Theorem 2 that will be needed for the proof of Theorem 1. In the generalized theorem (Theorem 3) the weight of a rectangle is measured as integral over some density function instead of the area.

2 Floorplans and Graphs

A *floorplan* is a dissection of a rectangle into a finite set of interiorly disjoint rectangles. From a given floorplan F we can obtain several graphs and additional structure. We introduce two of these and use them to define notions of equivalence for floorplans.

The Rectangular Dual. Let $R(F)$ be the set of rectangles of a floorplan F . It is convenient to include the enclosing rectangle in the set $R(F)$. The *rectangular dual* of F is the graph $G^*(F)$ with vertex set $R(F)$ and edges joining pairs of rectangles that share a boundary segment. Usually the notion of a rectangular dual is used in the other direction, i.e., it is assumed that a planar graph G is given and the quest is for a floorplan F such that $G = G^*(F)$. It is convenient to extend a floorplan F with four rectangles that frame F as shown in Figure 3. In the dual of an extended floorplan F_+ we omit the vertex that corresponds to the enclosing rectangle. With this twist in the definition of the dual we get: The dual $G_+^*(F)$ of the extended floorplan of a generic F is a 4-connected inner triangulation of a 4-gon. Indeed this is the characterization of the duals of extended generic floorplans. Buchsbaum et al. [3] and Felsner [6] provide many pointers to the literature related to floorplans and rectangular duals.

The Transversal Structure. The transversal structure (also known as regular edge labeling) associated to a floorplan F is an orientation and coloring of the edges of the extended dual $G_+^*(F)$.

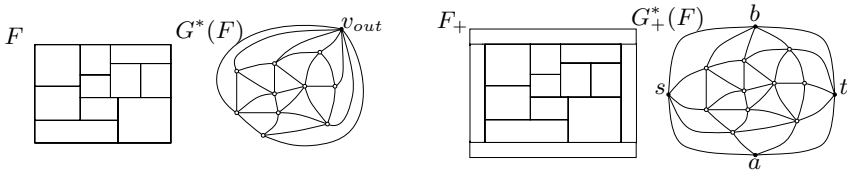


Fig. 3. A floorplan F , the extended floorplan F_+ , and the duals $G^*(F)$ and $G_+^*(F)$

Let G be an inner triangulation of a 4-gon with outer vertices s, a, t, b in counterclockwise order. A *transversal structure* for G is an orientation and 2-coloring of the inner edges of G such that two local conditions hold: 1) All edges incident to s, a, t and b are blue outgoing, red outgoing, blue ingoing, and red ingoing, respectively. 2) The edges incident to an inner vertex v come in clockwise order in four nonempty blocks consisting solely of red ingoing, blue ingoing, red outgoing, and blue outgoing edges, respectively.

Transversal structures have been studied in [9], [10], and in [12]. A proof of the following proposition can e.g. be found in [6].

Proposition 1. *Every transversal structure of an inner triangulation G of a 4-gon with outer vertices s, a, t, b is induced by a floorplan F with $G = G_+^*(F)$ and each floorplan F induces a transversal structure on $G_+^*(F)$.*

Figure 4 indicates the correspondence between floorplans and transversal structures.

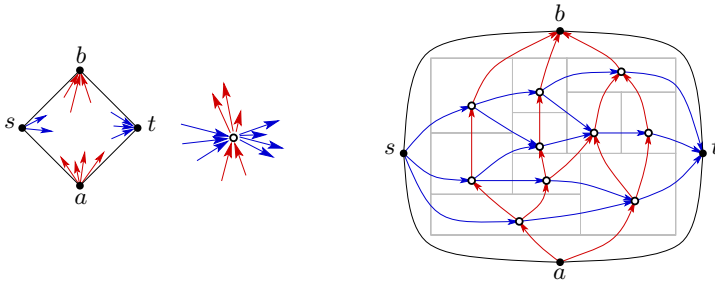


Fig. 4. The two local conditions and an example of a transversal structure together with a corresponding floorplan

The Segment Contact Graph. Recall that we call a floorplan *generic* if it has no cross, i.e., no point where four rectangles meet. A *segment* of a floorplan is a maximal non-degenerate interval that belongs to the union of the boundaries of the rectangles. In the generic case intersections between segments only occur between horizontal and vertical segments and they involve an endpoint of one of the segments, i.e., they are contacts. If a floorplan has a cross at point p we can break one of the two segments that contain p into two to get a system of interiorly disjoint segments.

The *segment contact graph* $G_{seg}(F)$ of a floorplan F is the bipartite planar graph whose vertices are the segments of F and edges correspond to contacts between segments. From Figure 5 we see that $G_{seg}(F)$ is indeed planar and that the faces of $G_{seg}(F)$ are in bijection with the rectangles of F and are uniformly of degree 4. Therefore $G_{seg}(F)$ is a maximal bipartite planar graph, i.e., a quadrangulation.

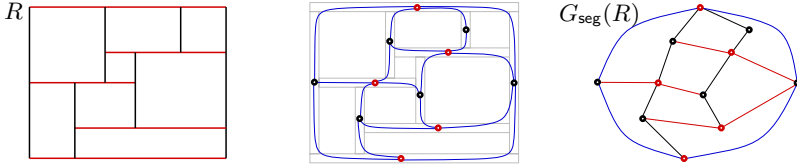


Fig. 5. A floorplan F and two drawings of its segment contact graph $G_{\text{seg}}(F)$

The Separating Decomposition. The separating decomposition associated to a floorplan is an orientation and coloring of the edges of the segment contact graph.

Let Q be a quadrangulation, we call the color classes of the bipartition white and black and name the two black vertices on the outer face s and t . A *separating decomposition* of Q is an orientation and coloring of the edges of Q with colors red and blue such that two conditions hold: 1) All edges incident to s are ingoing red and all edges incident to t are ingoing blue. 2) Every vertex $v \neq s, t$ is incident to a nonempty interval of red edges and a nonempty interval of blue edges. If v is white, then, in clockwise order, the first edge in the interval of a color is outgoing and all the other edges of the interval are incoming. If v is black, the outgoing edge is the last one in its color in clockwise order (see the left part of Figure 6).

Separating decompositions have been studied in [4], [8], and [7]. To us they are of interest because of the following proposition proved e.g. in [6].

Proposition 2. A floorplan F induces a separating decomposition on its segment contact graph $G_{\text{seg}}(F)$ and every separating decomposition of a planar quadrangulation Q corresponds to a floorplan F with $Q = G_{\text{seg}}(F)$.

The right part of Figure 6 indicates the correspondence between floorplans and separating-decompositions.

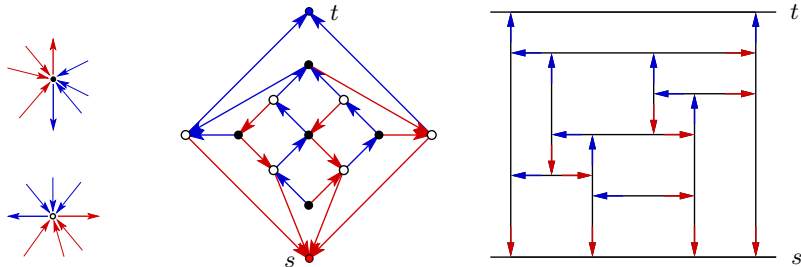


Fig. 6. The rule for black and white vertices. A separating decomposition S and a floorplan F corresponding to S .

2.1 Notions of Equivalence for Floorplans

Definition 1. Two floorplans are weakly equivalent if they induce the same separating decomposition.

Definition 2. Two floorplans are strongly equivalent if they induce the same transversal structure.

In the introduction we said that two floorplans F and F' are weakly equivalent if there exist bijections $\phi : S_H(F) \rightarrow S_H(F')$ and $\psi : S_V(F) \rightarrow S_V(F')$ between their horizontal and vertical segments such that segment s has an endpoint on segment t in F iff $\phi(s)$ has an endpoint on $\psi(t)$. We claim that this yields the same equivalence classes as Definition 1. Clearly, if F and F' induce the same separating decomposition then they are weakly equivalent in the above sense. For the converse first observe that the segment contact graphs of F and F' are isomorphic, i.e., $G_{\text{seg}}(F) = G_{\text{seg}}(F')$. Now define an orientation Q_F on $Q = G_{\text{seg}}(F)$ by orienting s to t iff segment s has an endpoint on segment t . Let $Q_{F'}$ be the orientation defined on Q using the segments of F' . Observe that $Q_F = Q_{F'}$. Since a separating decomposition is uniquely determined by the underlying 2-orientation (see, [4] or [7]) we conclude that F and F' induce the same separating decomposition, i.e., $SD_F = SD_{F'}$. This implies that F and F' are weakly equivalent in the sense of Definition 1.

Eppstein et al. [5] use the term *layout* instead of floorplan. Their equivalent layouts correspond to strongly equivalent floorplans and order-equivalent layouts to weakly equivalent floorplans. Asinowski et al. [2] study independent notions of R -equivalence and S -equivalence for floorplans.

3 Realizing Weighted Floorplans via Air-Pressure

In this section we prove a generalization of Theorem 2 to situations where the “area” of a rectangle is replaced by the mass defined through a density distribution.

Let $\mu : [0, 1]^2 \rightarrow \mathbb{R}_+$ be a density function on the unit square whose total mass is 1, i.e., $\int_0^1 \int_0^1 \mu(x, y) dx dy = 1$. We assume that μ can be integrated over axis aligned rectangles and all fibers μ_x and μ_y can be integrated over intervals. Moreover, we require that integrating μ over a nondegenerate rectangle and fibers over nondegenerate intervals always yields nonzero values. The *mass* of an axis aligned rectangle $R \subseteq [0, 1]^2$ is defined as $m(R) = \iint_R \mu(x, y) dx dy$.

Theorem 3. *Let $\mu : [0, 1]^2 \rightarrow \mathbb{R}_+$ be a density function of total mass 1. If F is a floorplan with rectangles R_1, \dots, R_{n+1} and $w : \{1, \dots, n+1\} \rightarrow \mathbb{R}_+$ a positive weight function with $\sum_1^{n+1} w(i) = 1$ then there exists a unique floorplan F' in the unit square that is weakly equivalent to F such that $m(R_i) = w(i)$ for each rectangle R_i .*

Our proof follows the air-pressure paradigm as proposed by Izumi, Takahashi and Kajitani [11]. We first describe the idea: Consider a realization of F in the unit square and compare the mass $m(R_i)$ to the intended mass $w(i)$. The quotient of these two values can be interpreted as the pressure inside the rectangle. Integrating this pressure along a side of the rectangle yields the force by which R_i is pushing against the segment that contains the side. The difference of pushing forces from both sides of a segment yields the effective force acting on the segment. The intuition is that shifting a segment in direction of the effective force yields a better balance of pressure in the rectangles. We will show that iterating such improvement steps drives the realization of F towards a situation with $m(R_i) = w(i)$ for all i , i.e., the procedure converges towards the floorplan F' whose existence we want to show.

In [11] the air-pressure paradigm was used for situations where the mass of a rectangle is its area. The authors observed fast convergence experimentally but they had no proof of convergence. Here we provide such a proof for the more general case of weights given by integrals over a density function.

A proof of Theorem 3 can also be given along the lines of the proof of Theorem 2 that has been given by Eppstein et al. in [5]. This approach has been taken by Schrenzenmaier [13]. The resulting proof is quite compact, however, it has the disadvantage of being purely existential. Schrenzenmaier also has a `java` implementation of the air-pressure approach that solves moderate size instances quickly.

Let $R_i = [x_l, x_r] \times [y_b, y_t]$ be a rectangle of F . Recall that the mass of R_i is $m(R_i) = \int_{x_l}^{x_r} \int_{y_b}^{y_t} \mu(x, y) dy dx$. The pressure $p(i)$ in R_i is the fraction of the intended mass $w(i)$ and the actual mass $m(R_i)$, i.e., $p(i) = \frac{w(i)}{m(R_i)}$. Let s be a segment of F and let R_i be one of the rectangles with a side in s . Let s be vertical with x -coordinate x_s and let $s \cap R_i$ span the interval $[y_b(i), y_t(i)]$. The (undirected) *force imposed on s by R_i* is the pressure $p(i)$ of R_i times the density dependent length of the intersection.

$$f(s, i) = \frac{w(i)}{m(R_i)} \int_{y_b(i)}^{y_t(i)} \mu(x_s, y) dy = p(i) \int_{y_b(i)}^{y_t(i)} \mu_{x_s}(y) dy.$$

The *force acting on s* is obtained as a sum of the directed forces imposed on s by incident rectangles.

$$f(s) = \sum_{R_i \text{ left of } s} f(s, i) - \sum_{R_i \text{ right of } s} f(s, i).$$

Symmetric definitions apply to horizontal segments.

Balance for Rectangles and Segments

A segment s is in *balance* if $f(s) = 0$. A rectangle R_i is in *balance* if $p(i) = 1$, i.e., if $w(i) = m(R_i)$.

Lemma 1. *If all rectangles R_i of F are in balance, then all segments are in balance.*

Proof. Since all rectangles are in balance we can eliminate the pressures from the definition of the $f(s, i)$. With this simplification we get for a vertical segment s

$$f(s) = \sum_{R_i \text{ left of } s} \int_{y_b(i)}^{y_t(i)} \mu_{x_s}(y) dy - \sum_{R_j \text{ right of } s} \int_{y_b(j)}^{y_t(j)} \mu_{x_s}(y) dy.$$

Hence $f(s) = M_s - M_s = 0$, where M_s is the integral of the fiber density μ_{x_s} along s . The symmetric argument applies to horizontal segments. \square

Interestingly, the converse of the lemma also holds.

Proposition 3. *If all segments of F are in balance, then all rectangles are in balance.*

Proof. Suppose that F balances all segments but not all rectangles. Choose some τ with $\min_i p(i) < \tau \leq \max_i p(i)$. Let T_τ be the union of all rectangles R_i whose pressure exceeds τ and let Γ_τ be the boundary of T_τ .

Claim. The boundary Γ_τ of T_τ contains no complete segment.

Suppose Γ_τ contains the vertical segment s such that T_τ is left of s . Let I be a nontrivial interval on s that is defined as intersection of a rectangle R_i that has its right edge on s and a rectangle R_j that has its left edge on s . The force acting on s along I is $p(i) \int_I \mu_{x_s}(y) dy - p(j) \int_I \mu_{x_s}(y) dy$. Since $\int_I \mu_{x_s}(y) dy > 0$ by our assumption on μ and $p(i) > p(j)$ by definition of T_τ the force is positive. This holds for every interval I on s , hence, the overall force $f(s)$ acting on s is also positive. This contradicts the assumption that s is in balance and completes the proof of the claim. \triangle

Let s_0 be any segment which contributes to Γ_τ . From the lemma we know that at some interior point of segment s_0 the boundary leaves s_0 and continues along another segment s_1 . Again, the boundary has to leave s_1 at some interior point to continue on s_2 . Because this procedure always follows the boundary of T which is a region defined by a union of rectangles in F the sequence of segments has to get back to segment s_0 , i.e., there is an k such that $s_k = s_0$.

From the definition of the separating decomposition SD_F corresponding to F we find that $s_0 \leftarrow s_1 \leftarrow s_2 \leftarrow \dots \leftarrow s_{k-1} \leftarrow s_0$ is a directed cycle in SD_F . The four segments of the enclosing square of F do not contribute to the boundary of T_τ , simply because they cannot belong to a directed cycle of SD_F .

Recall the assumption that F balances all segments but not all rectangles. Let s be the vertical segment with maximal x -coordinate among all vertical segments that contribute to a boundary Γ_τ for some τ . From the choice of s it is clear that T_τ is to the left of s . Consider the segment $s' = s_{k-1}$ following $s = s_0$ in the cycle $s_0 \leftarrow \dots \leftarrow s_{k-1} \leftarrow s_0$ in SD_F corresponding to Γ_τ . Left from the contact point p of s and s' the segment s' is part of the boundary Γ_τ of T_τ . From the choice of τ and s it follows that to the right of p the rectangles on both sides of s' have the same pressure $p(i)$. Otherwise the right part of s' would belong to some boundary $\Gamma_{\tau'}$ and the vertical segment following s' on $\Gamma_{\tau'}$ is in contradiction to the choice of s .

Now consider $f(s')$ and split the contributions to this force at p . Left of p the pressure on the side of T_τ exceeds the pressure from the other side. Right of p the rectangles on both sides of p have the same pressure. This shows that in contradiction to the assumption $f(s') \neq 0$. This completes the proof of the proposition. \square

Balancing Segments and Optimizing the Entropy

Proposition 4. *If a segment s of F is unbalanced, then we can keep all the other segments at their position and shift s parallel to a position where it is in balance. The resulting floorplan F' is weakly equivalent to F .*

Proof. We consider the case of a vertical segment, the horizontal case is symmetric. Let x_s be the x -coordinate of s . With S_- and S_+ we denote the sets of rectangles in F that touch s from the left respectively from the right. Let R_l be the rectangle with a left boundary of maximal x -coordinate x_l in S_- and let R_r be the rectangle with a right boundary of minimal x -coordinate x_r in S_+ . Note that if t satisfies $x_l < t < x_r$ then segment s can be shifted parallel to the position $x_s = t$ and the resulting floorplan is weakly equivalent to F .

For $t \in (x_l, x_r)$ we define $h(t)$ as the force acting on s when the segment it is shifted to $x_s = t$. We observe:

- The pressure $p(i)$ depends continuously on t for all rectangles $R_i \in S_- \cup S_+$.
- The value of $\int_I \mu_t(y) dy$ is a continuous function of t for all intervals I .

Hence, $h(t)$ is a continuous function. With t approaching x_l from the right the area of R_l tends to zero. Hence, the mass $m(R_l)$ also tends to zero and the pressure $p(l)$ tends to infinity. Since $\int_{y_b(t)}^{y_t(l)} \mu_t(y) dy > 0$ we conclude that $h(t) \rightarrow +\infty$ with $t \rightarrow x_l$. A similar reasoning involving R_r shows that $h(t) \rightarrow -\infty$ with $t \rightarrow x_r$. It follows that there is some $t_0 \in (x_l, x_r)$ with $h(t_0) = 0$. Hence, if we shift s to the position $x_s = t_0$ the force acting on s vanishes and s is in balance. \square

Definition 3. The entropy of a rectangle R_i of F is defined as $-w(i) \log p(i)$. The entropy of the floorplan F is

$$E = \sum -w(i) \log p(i)$$

The proof of Theorem 3 will be completed after showing the following

- (1) The entropy E is always nonpositive.
- (2) $E = 0$ if and only if all rectangles R_i of F are in balance.
- (3) Shifting an unbalanced segment s into its balance position increases the entropy.
- (4) The process of repeatedly shifting unbalanced segments into their balance position makes F converge to a floorplan F' such that the entropy of F' is zero.
- (5) The solution is unique.

The first two of these statements are shown in the next lemma.

Lemma 2. The entropy E is always nonpositive and $E = 0$ if and only if all rectangles R_i of F are in balance.

Proof. We use that $p(i) > 0$ and hence $\log p(i) \geq (1 - \frac{1}{p(i)}) = (1 - \frac{m(R_i)}{w(i)})$. For the entropy of R_i we get $-w(i) \log p(i) \leq -w(i)(1 - \frac{m(R_i)}{w(i)}) = m(R_i) - w(i)$. This yields

$$E = \sum_i -w(i) \log p(i) \leq \sum_i m(R_i) - \sum_i w(i) = 1 - 1 = 0$$

The equality $E = 0$ is equivalent to equality for each summand. Hence $0 = \log p(i) = (1 - \frac{m(R_i)}{w(i)})$ and $m(R_i) = w(i)$ for all i . \square

Lemma 3. Shifting an unbalanced segment s into its balance position increases the entropy.

Proof. We consider a vertical segment s as in the proof of Proposition 4 and assume $f(s) > 0$. Let t_0 be the first zero of $h(t)$ right of x_s . For all $t \in [x_s, t_0)$ the force $h(t)$ acting on s is positive, i.e., pushing s to the right.

Let $E(t)$ be the entropy of the floorplan when s is shifted towards the position $x_s = t_0$. We consider $E(t)$ as a function of t .

Claim. $\frac{d}{dt} E(t) = h(t)$.

Only rectangles touching s change their contribution to $E(t)$. Let $R_i = [x_l, t] \times [y_1, y_2]$ be a rectangle on the left of s , i.e., $R_i \in S_-$, and t is the x -coordinate of the right side of R_i . Hence

$$\begin{aligned} \frac{d}{dt}(-w(i) \log p(i)) &= -w(i) \frac{1}{p(i)} p'(i) = -w(i) \frac{m(R_i)}{w(i)} \frac{d}{dt} \frac{w(i)}{m(R_i)} = \\ -w(i) \frac{m(R_i)}{w(i)} w(i) \frac{-m'(R_i)}{m^2(R_i)} &= \frac{w(i)}{m(R_i)} m'(R_i) = \frac{w(i)}{m(R_i)} \frac{d}{dt} \int_{x_l}^t \int_{y_1}^{y_2} \mu(x, y) dy dx = \\ \frac{w(i)}{m(R_i)} \int_{y_1}^{y_2} \mu_t(y) dy &= p(i) \int_{y_1}^{y_2} \mu_t(y) dy. \end{aligned}$$

When $R_i \in S_+$ the mass $m(R_i)$ is decreasing with t so that $m'(R_i)$ is negative and $\frac{d}{dt}(-w(i) \log p(i)) = -p(i) \int_{y_1}^{y_2} \mu_t(y) dy$. Summing this over all rectangles incident to s we obtain that $\frac{d}{dt} E(t) = h(t)$. This is the claim. \triangle

While shifting s from the initial position x_s to t_0 we have $h(t) > 0$. The claim implies that the derivative of the entropy is positive and, hence, the entropy is increasing. \square

We continue with item (4) from our program. To prove this, however, we have to add a condition to the process of balancing segments. The iteration has to be performed such that no unbalanced segment can be ignored. A rule is called *nonignoring* if it complies with this condition. Here are two examples of nonignoring selection rules: 1) Choose the segment for balancing uniformly at random from the set of unbalanced segments. 2) Always choose the segment so that the increase of the entropy is as large as possible.

Proposition 5. *Let F_0, F_1, F_2, \dots be a sequence of floorplans where F_{i+1} is obtained from F_i by balancing an unbalanced segment from F_i . If the selection of segments is nonignoring, then there is a subsequence G_0, G_1, \dots of floorplans that has a limit $G = \lim G_i$ and the entropy of the floorplan G is zero.*

Proof. Let s_1, s_2, \dots, s_n be the inner segments of F . Floorplans that are weakly equivalent to F can be encoded by the coordinate vector of the segments. This vector z in \mathbb{R}^n has the value $z(i) = x_s$ if s_i is a vertical segment and $z(i) = y_s$ if s_i is horizontal. A sequence of floorplans is converging if the corresponding coordinate vectors converge.

Consider the sequence of coordinate vectors z_0, z_1, \dots of the given sequence of floorplans. Since each of the coordinates of these vectors is from the interval $(0, 1)$, there is a convergent subsequence. Let G_0, G_1, \dots be the corresponding convergent sequence of floorplans and let e_i be the entropy of G_i . From Lemma 3 we know that the e_i are an increasing sequence of negative numbers. Assume that the sequence e_i converges to $-a \neq 0$. Consider the limit $G = \lim G_i$. Since the entropy of G is $-a < 0$ there is an unbalanced rectangle R_j in G (Lemma 2) and, hence, there is an unbalanced segment s in G (Proposition 3). Let Δ be the increase of the entropy that comes from balancing s in G . Now, for all i greater than a sufficiently large N the floorplan G_i is so close to G that balancing s in G_i implies an increase of entropy of at least $\Delta/2$. For all i greater than a sufficiently large M we also have $e_i > -a - \Delta/2$. It follows that the unbalanced segment s was not used for balancing in any G_i with $i \geq \max(M, N)$. This is in contradiction to the assumption that the process is nonignoring. \square

Actually, a stronger statement is true. The full sequence F_0, F_1, F_2, \dots is also converging. To prove this we need the uniqueness from Proposition 6. In fact if G is the unique floorplan that is weakly equivalent to F and has $m(R_i) = w(i)$ for all i , then it follows from the continuity of the entropy that there is an $\varepsilon > 0$ such that all floorplans whose entropy is larger than $-\varepsilon$ have a coordinate vector that is δ_ε close to the coordinate vector of G . This implies that $\lim F_i = G$.

Proposition 6. *For every floorplan F with $n + 1$ rectangles and every positive weight function $w : \{1, \dots, n + 1\} \rightarrow \mathbb{R}_+$ with $\sum_i w(i) = 1$ there is a unique floorplan F' in the unit square that is weakly equivalent to F and has $m(R_i) = w(i)$ for all i .*

A proof of the proposition can be found in the paper by Eppstein et al. [5].

4 Accommodating Floorplans on Point Sets

Let P be a generic set of n points in a rectangle \mathcal{R} . Let F be a generic floorplan and S be a subset of the segments of F of size n . A *cover map* from (F, S) to P is a floorplan F' with outer rectangle \mathcal{R} that is weakly equivalent to F such that every segment from $S' = \phi(S)$ contains a point from P . The following is a generalization of Theorem 1.

Theorem 4. *If P is a generic set of n points in a rectangle \mathcal{R} and F is a generic floorplan with a prescribed subset S of the segments of size n , then there is a cover map F' from (F, S) to P .*

Proof. We use Theorem 3 as a tool for the proof. First we transform the point set P into a suitable density distribution $\mu = \mu_P$ inside \mathcal{R} . This density is defined as the sum of a uniform distribution μ_1 with $\mu_1(q) = 1/\text{area}(\mathcal{R})$ for all $q \in \mathcal{R}$ and a distribution μ_2 that represents the points of P . Choose some $\Delta > 0$ such that $\|p - p'\| > 3\Delta$ for all $p, p' \in P$. Define $\mu_2 = \sum_{p \in P} \mu_p$ where $\mu_p(q)$ takes the value $(\Delta^2 \pi)^{-1}$ on the disk $D_\Delta(p)$ of radius Δ around p and value 0 for q outside of this disk.

For a density ν defined on \mathcal{R} and a rectangle $R \subseteq \mathcal{R}$ let $\nu(R)$ be the integral of the density ν over R . Since $\mu_1(\mathcal{R}) = 1$ and $\mu_p(\mathcal{R}) = 1$ for all $p \in P$ the total mass of \mathcal{R} is $\mu(\mathcal{R}) = 1 + n$.

Transform the floorplan F into a floorplan F_S depending on the set S of segments that have to cover points of P : inflate every segment in S to form a thin rectangle. This description is not very formal but with a look at Figure 7 should make clear how to produce F_S from F and S . Let \mathcal{S} be the set of new rectangles obtained by inflating segments from S .

Define weights for the rectangles of F_S as follows. If F_S has r rectangles we define $w(R) = 1 + 1/r$ if $R \in \mathcal{S}$ and $w(R) = 1/r$ for all the rectangles of F_S that came from rectangles of F . The total weight, $\sum_R w(R) = 1 + n$ equals the total mass $\mu(\mathcal{R})$.

The data \mathcal{R} with μ and F_S with w constitute, up to scaling of \mathcal{R} and w , a set of inputs for Theorem 3. From the conclusion of the theorem we obtain a floorplan F'_S weakly equivalent to F_S such that $m(R) = \iint_R \mu(x, y) dx dy = w(R)$ for all rectangles.

The definition of the weight function w and the density μ is so that F'_S should be close to a cover map from (F, S) to P : Only the rectangles $R \in \mathcal{S}$ that have been constructed by inflating segments may contain a disk $D_\Delta(p)$ and each of these rectangles

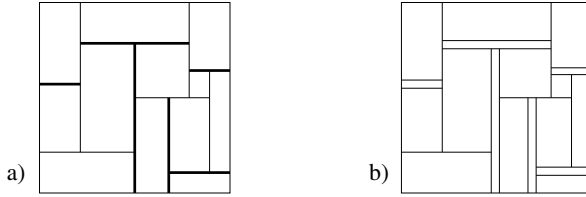


Fig. 7. Floorplans F with as prescribed subset S of segments (bold) and the floorplan F_S obtained by doubling the segments of S

may contain at most one of the disks. This suggests a correspondence $S \leftrightarrow P$. However, a rectangle $R \in S$ may use parts of several discs to accumulate mass. To find a correspondence between S and P we define a bipartite graph G whose vertices are the points in P and the rectangles in S :

- A pair (p, R) is an edge of G iff $R \cap D_\Delta(p) \neq \emptyset$ in F'_S .

The proof of the theorem will be completed by proving two claims: 1) G admits a perfect matching. 2) From F'_S and a perfect matching M in G we can produce a floorplan F' that realizes the cover map from (F, S) to P .

For the first of the claims we check Hall's matching condition: Consider a subset A of S . Since F_S is realizing the prescribed weights we have $m(A) = \mu(A) = \sum_{R \in A} \mu(R) = \sum_{R \in A} w(R) = |A|(1 + 1/r)$. Since $\mu_1(A) < 1$ and $\mu_p(A) \leq 1$ for all $p \in P$ there must be at least $|A|$ points $p \in P$ with $\mu_p(A) > 0$, these are the points that have an edge to a rectangle from A in G . We have thus shown that every set A of inflated segments is incident to at least $|A|$ points in G , hence, there is an injective mapping $\alpha : S \rightarrow P$ such that $R \cap D_\Delta(\alpha(R)) \neq \emptyset$ in F'_S for all $R \in S$.

To prove the second claim we have to construct a floorplan F' that realizes the cover map from (F, S) to P : Let s be a segment in S and let R_s be the rectangle in F'_S that corresponds to s . If s is horizontal we define s' to be the unique maximal horizontal segment in R_s whose y -coordinate is as close to the y -coordinate of the point $\alpha(R_s)$ as possible. For vertical segments we focus on the x -coordinate. For segments s of F that do not belong to S set $s' = s$. The collection $\{s' : s \text{ segment in } F\}$ of segments may fail to be a floorplan, see e.g. Figure 8.b. However, if s_1 and s_2 are segments of F such that s_1 has one of its endpoints on s_2 and $s_2 \in S$ then we can extend s'_1 into R_{s_2} to recover the contact with s'_2 . By doing this for all qualifying pairs s'_1, s'_2 we get a floorplan, see Figure 8.c. This floorplan is weakly equivalent to F but there may still be segments of S that do not quite cover the assigned point. By construction the distance from a segment to its assigned point is at most Δ . Since Δ is small compared to the distances of points in P we can shift all segments orthogonally to make them cover their assigned points. Again this may spoil the floorplan property, see e.g. Figure 8.d. However, enlarging or shortening of segments by an amount of at most Δ at the ends finally yields the floorplan F' that realizes the cover from (F, S) to P . □

The topic of [1] was the study of the number $Z(P)$ of rectangulations of a generic point set P . In our terminology this is the total number of cover maps from floorplans with n inner segments to a generic point set P with n points. Theorem 4 implies that this number is at least as large as the number of weak equivalence classes of floorplans.

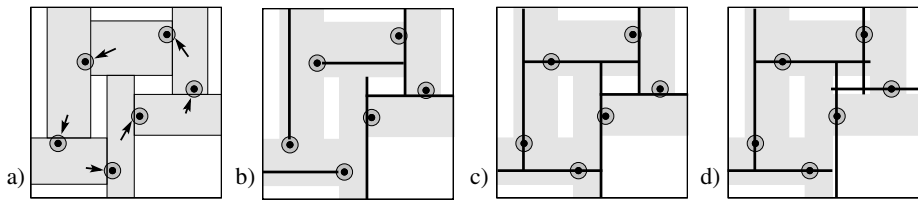


Fig. 8. a) A solution F'_S for the instance from Fig. 1. The arrows indicate a matching α . b) Segments $s \in S$ shifted to their optimal position in R_s . c) Enlarged segments recover the contacts. d) Some segments s are moved outside R_s to cover the corresponding points $\alpha(R_s)$. Small final adjustments (clipping and enlarging) yield F' .

This is the Baxter number B_{n+1} which is known to be of order $\Theta(8^{n+1}/(n+1)^4)$. In [1] an upper bound for $Z(P)$ of order $O(20^n/n^4)$ is shown. To improve this bound remains an intriguing problem.

References

1. Ackerman, E., Barequet, G., Pinter, R.Y.: On the number of rectangulations of a planar point set. *J. Combin. Theory Ser. A* 113, 1072–1091 (2006)
2. Asinowski, A., Barequet, G., Bousquet-Mélou, M., Mansour, T., Pinter, R.: Orders induced by segments in floorplan partitions and (2-14-3,3-41-2)-avoiding permutations (2010) (submitted), arXiv:1011.1889
3. Buchsbaum, A.L., Gansner, E.R., Procopiu, C.M., Venkatasubramanian, S.: Rectangular layouts and contact graphs. *ACM Trans. Alg.* 4, 28 pages (2008)
4. de Fraysseix, H., Ossona de Mendez, P.: On topological aspects of orientation. *Discr. Math.* 229, 57–72 (2001)
5. Eppstein, D., Mumford, E., Speckmann, B., Verbeek, K.: Area-universal and constrained rectangular layouts. *SIAM J. Computing* 41, 537–564 (2012)
6. Felsner, S.: Rectangle and square representations of planar graphs. In: Pach, J. (ed.) *Thirty Essays on Geometric Graph Theory*, pp. 213–248. Springer, New-York (2012)
7. Felsner, S., Fusy, É., Noy, M., Orden, D.: Bijections for Baxter families and related objects. *Journal of Comb. Theory A* 18, 993–1020 (2011)
8. Felsner, S., Huemer, C., Kappes, S., Orden, D.: Binary labelings for plane quadrangulations and their relatives. *Discr. Math. & Theor. Comp. Sci.* 12, 115–138 (2010)
9. Fusy, É.: *Combinatoire des cartes planaires et applications algorithmiques*. PhD thesis, LIX Ecole Polytechnique (2007), http://www.lix.polytechnique.fr/~fusy/Articles/these_eric_fusy.pdf
10. Fusy, É.: Transversal structures on triangulations: A combinatorial study and straight-line drawings. *Discr. Math.* 309, 1870–1894 (2009)
11. Izumi, T., Takahashi, A., Kajitani, Y.: Air-pressure model and fast algorithms for zero-wasted-area layout of general floorplan. *IEICE Trans. Fundam. Electron., Commun. and Comp. Sci.* E81-A, 857–865 (1998)
12. Kant, G., He, X.: Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theor. Comput. Sci.* 172, 175–193 (1997)
13. Schrenzenmaier, H.: *Ein Luftdruckparadigma zur Optimierung von Zerlegungen*. Bachelor's Thesis, TU Berlin (2013)
14. Wimer, S., Koren, I., Cederbaum, I.: Floorplans, planar graphs, and layouts. *IEEE Transactions on Circuits and Systems* 35, 267–278 (1988)

Superpatterns and Universal Point Sets

Michael J. Bannister, Zhanpeng Cheng,
William E. Devanny, and David Eppstein *

Department of Computer Science, University of California, Irvine, USA

Abstract. An old open problem in graph drawing asks for the size of a *universal point set*, a set of points that can be used as vertices for straight-line drawings of all n -vertex planar graphs. We connect this problem to the theory of permutation patterns, where another open problem concerns the size of *superpatterns*, permutations that contain all patterns of a given size. We generalize superpatterns to classes of permutations determined by forbidden patterns, and we construct superpatterns of size $n^2/4 + \Theta(n)$ for the 213-avoiding permutations, half the size of known superpatterns for unconstrained permutations. We use our superpatterns to construct universal point sets of size $n^2/4 - \Theta(n)$, smaller than the previous bound by a 9/16 factor. We prove that every proper subclass of the 213-avoiding permutations has superpatterns of size $O(n \log^{O(1)} n)$, which we use to prove that the planar graphs of bounded pathwidth have near-linear universal point sets.

1 Introduction

Fary's theorem tells us that every planar graph can be drawn with its edges as non-crossing straight line segments. As usually stated, this theorem allows the vertex coordinates of the drawing to be drawn from an uncountable and unbounded set, the set of all points in the plane. It is natural to ask how tightly we can constrain the set of possible vertices. In this direction, the *universal point set problem* asks for a sequence of point sets $U_n \subseteq \mathbf{R}^2$ such that every graph with n vertices can be straight-line embedded with vertices in U_n and such that the size of U_n is as small as possible.

So far the best known upper bounds for this problem have considered sets U_n of a special form: the intersection of the integer lattice with a convex polygon. In 1988 de Fraysseix, Pach and Pollack showed that a triangular set of lattice points within a rectangular grid of $(2n-3) \times (n-1)$ points forms a universal set of size $n^2 - O(n)$ [1,2], and in 1990 Schnyder found more compact grid drawings within the lower left triangle of an $(n-1) \times (n-1)$ grid [3], a set of size $n^2/2 - O(n)$. Using the method of de Fraysseix et al., Brandenburg found that a triangular subset of a $\frac{4}{3}n \times \frac{2}{3}n$ grid, of size $\frac{4}{9}n^2 + O(n)$, is universal [4]. Until now his bound has remained the best known.

On the other side, Dolev, Leighton, and Trickey [5] used the *nested triangles graph* to show that rectangular grids that are universal must have size at least $n/3 \times n/3$, or with a fixed choice of planar embedding and outer face $2n/3 \times 2n/3$. Thus, if we wish to find subquadratic universal point sets we must consider sets not forming a grid. However,

* This research was supported in part by the National Science Foundation under grants 0830403 and 1217322, and by the Office of Naval Research under MURI grant N00014-08-1-1015.

the known lower bounds that do not make this grid assumption are considerably weaker. In 1988 de Fraysseix, Pach and Pollack proved the first nontrivial lower bounds of $n + \Omega(\sqrt{n})$ for a general universal point set [1]. This was later improved to $1.098n - o(n)$ by Chrobak and Payne [2]. Finally, Kurowski improved the lower bound to $1.235n$ [6], which is still the best known lower bound.¹

With such a large gap between these lower bounds and Brandenburg's upper bound, obtaining tighter bounds remains an important open problem in graph drawing [8].

Universal point sets have also been considered for subclasses of planar graphs. For instance, every set of n points in general position (no three collinear) is universal for the n -vertex outerplanar graphs [9]. Universal point sets of size $O(n(\log n / \log \log n)^2)$ exist for simply-nested planar graphs (graphs that can be decomposed into nested induced cycles) [10], and planar 3-trees have universal point sets of size $O(n^{5/3})$ [11]. Based in part on the results in this paper, the graphs of line and pseudoline arrangements have been shown to have universal point sets of size $O(n \log n)$ [12].

In this paper we provide a new upper bound on universal point sets for general planar graphs, and improved bounds for certain restricted classes of planar graphs. We approach these problems via a novel connection to a different field of study than graph drawing, the study of patterns in permutations.² A permutation σ is said to *contain* the pattern π (also a permutation) if σ has a (not necessarily contiguous) subsequence whose elements are in the same relative order with respect to each other as the elements of π . The permutations that do not contain any pattern in a given set F of forbidden patterns are said to be *F-avoiding*; we define $S_n(F)$ to be the length- n permutations avoiding F . Researchers in permutation patterns have defined a *superpattern* to be a permutation that contains all length- n permutations among its patterns, and have studied bounds on the lengths of these patterns [14, 15], culminating in a proof by Miller that there exist superpatterns of length $n^2/2 + \Theta(n)$ [16]. We generalize this concept to an $S_n(F)$ -superpattern, a permutation that contains all possible patterns in $S_n(F)$; we prove that for certain sets F , the $S_n(F)$ -superpatterns are much shorter than Miller's bound.

As we show, the existence of small $S_n(213)$ -superpatterns leads directly to small universal point sets for arbitrary planar graphs, and the existence of small $S_n(F)$ -superpatterns for F containing 213 leads to small universal point sets for subclasses of the planar graphs. Our method constructs a universal set U that has one point for each element of the superpattern σ . It uses two different traversals of a depth-first-search tree of a canonically oriented planar graph G to derive a permutation $\text{cperm}(G)$ from G , and it uses the universality of σ to find $\text{cperm}(G)$ as a pattern in σ . Then, the positions of the elements of this pattern in σ determine the assignment of the corresponding vertices of G to points in U , and we prove that this assignment gives a planar embedding of G . A similar but simpler reduction uses S_n -superpatterns to construct universal point sets for *dominance drawings* of transitively reduced *st*-planar graphs.

¹ The validity of this result was originally questioned by Mondal [7], but later confirmed.

² A different connection between permutation patterns and graph drawing is being pursued independently by Bereg, Holroyd, Nachmanson, and Pupyrev, in connection with bend minimization in bundles of edges that realize specified permutations [13].

Specifically our contributions include proving the existence of:

- superpatterns for 213-avoiding permutations of size $n^2/4 + \Theta(n)$;
- universal point sets for planar graphs of size $n^2/4 - \Theta(n)$;
- universal point sets for dominance drawings of size $n^2/2 + \Theta(n)$;
- superpatterns for every proper subclass of the 213-avoiding permutations of size $O(n \log^{O(1)} n)$;
- universal point sets for graphs of bounded pathwidth of size $O(n \log^{O(1)} n)$; and
- universal point sets for simply-nested planar graphs of size $O(n \log n)$.

In addition, we prove that every superpattern for $\{213, 132\}$ -avoiding permutations has length $\Omega(n \log n)$, which in turn implies that every superpattern for 213-avoiding permutations has length $\Omega(n \log n)$. It was known that S_n -superpatterns must have quadratic length—otherwise they would not have enough length- n subsequences to cover all $n!$ permutations [14]—but such counting arguments cannot provide nonlinear bounds for $S_n(F)$ -superpatterns due to the now-proven Stanley–Wilf conjecture that $S_n(F)$ grows singly exponentially [17]. Instead, our proof finds an explicit set of $\{213, 132\}$ -avoiding permutations whose copies within a superpattern cannot share many elements.

2 Preliminaries

Let S_n denote the set of all *permutations* of the numbers from 1 to n . We will normally specify a permutation as a sequence of numbers: for instance, the six permutations in S_3 are 123, 132, 213, 231, 312, and 321. If π is a permutation, then we write π_i for the element in the i th position of π , and $|\pi|$ for the number of elements in π .

We say that a permutation π is a *subpattern* of a permutation σ of length n if there exists a sequence of integers $1 \leq \ell_1 < \ell_2 < \dots < \ell_{|\pi|} \leq n$ such that $\pi_i < \pi_j$ if and only if $\sigma_{\ell_i} < \sigma_{\ell_j}$. In other words, π is a subpattern of σ if π is order-isomorphic to a subsequence of σ . We say that a permutation σ *avoids* a permutation ϕ if σ does not contain ϕ as a subpattern. A *permutation class* is a set of permutations with the property that all patterns of a permutation in the class also belong to the class; every permutation class may be defined by a (not necessarily finite) set of *forbidden permutations*, the minimal patterns that do not belong to the class. Define $S_n(\phi_1, \dots, \phi_k)$ to be the set of all length- n permutations that avoid all of the (forbidden) patterns ϕ_1, \dots, ϕ_k . Given a set of permutations $P \subseteq S_n$, we define a *P-superpattern* to be a permutation σ with the property that every $\pi \in P$ is a subpattern of σ .

One of the most important permutation classes in the study of permutation patterns is the class of *stack-sortable permutations* [18], the permutations that avoid the pattern 231. Knuth’s discovery that these are exactly the permutations that can be sorted using a single stack [19] kicked off the study of permutation patterns. The 213-avoiding permutations that form the focus of our research are related to the 231-avoiding permutations by a simple transformation, the replacement of each value i in a permutation by the value $n + 1 - i$, that does not affect the existence or size of superpatterns.

Given a permutation π we define a *column* of π to be a maximal ascending run of π , and we define a *row* of π to be a maximal ascending run in π^{-1} , where a *run* is a contiguous monotone subsequence of the permutation. We define a *block* of π to be a

set of consecutive integers that appear contiguously (but not necessarily in order) in π . For instance, $\{3, 4, 5\}$ forms a block in 14352. (Our definition of rows and columns differs from that of Miller [16]: for our definition, the intersection of a row and column is a block that could contain more than one element, whereas in Miller’s definition a row and column intersect in at most one element.)

We define the *chessboard representation* of a permutation π to be an $r \times c$ matrix $M = \text{chessboard}(\pi)$, where r is number of rows in π and c is the number of columns in π , such that $M(i, j)$ is the number of points in the intersection of the i^{th} column and the j^{th} row of π . An example of a chessboard representation can be seen in Figure 1. To recover a permutation from its chessboard representation, start with the lowest row and work upwards assigning an ascending subsequence of values to the squares of each row in left to right order within each row. If a square has label i , allocate i values for it. Then, after this assignment has been made, traverse each column in left-to-right order, within each column listing in ascending order the values assigned to each square of the column. The sequence of values listed by this column traversal is the desired permutation.

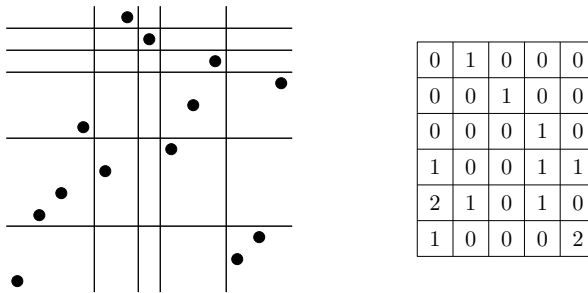


Fig. 1. The permutation $\pi = 1\ 4\ 5\ 8\ 6\ 13\ 12\ 7\ 9\ 11\ 2\ 3\ 10$ represented by its scatterplot (the points (i, π_i)) with lines separating its rows and columns (left), and by chessboard(π) (right)

3 From Superpatterns to Universal Point Sets

In this section, we show how 213-avoiding superpatterns can be turned into universal point sets for planar graphs. Let G be a planar graph. We assume G is *maximal planar*, meaning that no additional edges can be added to G without breaking its planarity; this is without loss of generality, because a point set that is universal for maximal planar graphs is universal for all planar graphs. Additionally, we assume that G has a fixed plane embedding; for maximal planar graphs, such an embedding is determined by the choice of which of the triangles of G is to be the outer face.

3.1 Canonical Representation

As in the grid drawing method of de Fraysseix, Pach and Pollack [1], we use *canonical representations* of planar graphs; these are sequences v_1, v_2, \dots, v_n of the vertices of the given maximal plane graph G with the following properties:

- $v_1v_2v_n$ is the outer triangle of the embedding, in clockwise order v_1, v_n, v_2 .
- Each vertex v_i with $i \geq 3$ has two or more earlier neighbors in the sequence, and these neighbors form a contiguous subset of the cyclic ordering of neighbors around v_i in the embedding of G .

Given a canonical representation, let G_k be the induced subgraph of G with vertex set $\{v_1, v_2, \dots, v_k\}$. Then G_k is necessarily 2-connected; its induced embedding has as its exterior face a simple cycle C_k containing v_k and the edge v_1v_2 , and the neighbors of v_k in G_k form an induced path in C_{k-1} .

As de Fraysseix, Pach and Pollack proved, every embedded maximal planar graph has at least one canonical representation. For the rest of this section, we will assume that the vertices v_i of the given maximal planar graph G are numbered according to such a representation. The definition of a canonical representation implies that the outer face of G is the triangle $v_1v_2v_n$; we will assume that this triangle is oriented so that v_1, v_n, v_2 are in clockwise order.

For each vertex v_i with $i > 2$, let $\text{parent}(v_i)$ be the most clockwise smaller-numbered neighbor of v_i . By following a path of edges from vertices to their parents, each vertex can reach v_1 , so these edges form a tree $\text{ctree}(G)$ having v_1 as its root; this same tree may also be obtained by orienting each edge of G from lower to higher numbered vertices, and then performing a depth-first search of the resulting oriented graph that visits the children of each vertex in clockwise order, starting from v_1 .³ For each vertex v_i of G , let $\text{pre}(v_i)$ be the position of v_i in a pre-order traversal of $\text{ctree}(G)$ that visits the children of each node in clockwise order, and let $\text{post}(v_i)$ be the position of v_i in a sequence of the nodes of $\text{ctree}(G)$ formed by reversing a post-order clockwise traversal. These two numbers may be used to determine the ancestor-descendant relationships in $\text{ctree}(G)$: a node v_i is an ancestor of a node v_j if and only if both $\text{pre}(v_i) < \text{pre}(v_j)$ and $\text{post}(v_i) < \text{post}(v_j)$ [20].

Lemma 1. *Let G be a canonically-represented maximal plane graph, and renumber the vertices of G in order by their values of $\text{post}(v_i)$. Then the result is again a canonical representation of the same embedding of G , and for each induced subgraph G_k of this new canonical representation, the clockwise ordering of the vertices along the exterior cycle C_k is in sorted order by the values of $\text{pre}(v_i)$.*

Proof. The fact that $\text{post}(v_i)$ gives a canonical representation comes from the fact that it is a reverse postorder traversal of a depth-first search tree. Reverse postorder traversal gives a topological ordering of every directed acyclic graph, from which it follows that every vertex in G has the same set of earlier neighbors when ordered by $\text{post}(v_i)$ as it did in the original ordering.

The statement on the ordering of the vertices of C_k follows by induction, from the fact that v_k has a larger value of $\text{pre}(v_i)$ than its earliest incoming neighbor (its parent in $\text{ctree}(G)$) and a smaller value than all of its other incoming neighbors. \square

³ Although we do not use this fact, $\text{ctree}(G)$ is also part of a Schnyder decomposition of G , together with a second tree rooted at v_2 connecting each vertex to its most counterclockwise earlier neighbor and a third tree rooted at v_n connecting each vertex to the later vertex whose addition removes it from C_k .

Let $\text{cperm}(G)$ be the permutation in which, for each vertex v_i , the permutation value in position $\text{pre}(v_i)$ is $\text{post}(v_i)$. That is, $\text{cperm}(G)$ is the permutation given by traversing $\text{ctree}(G)$ in preorder and listing for each vertex of the traversal the number $\text{post}(v_i)$.

Lemma 2. *For every canonically-represented maximal planar graph G , the permutation $\pi = \text{cperm}(G)$ is 213-avoiding.*

Proof. Let $i < j < k$ be an arbitrary triple of indexes in the range from 1 to n , corresponding to the vertices u_i , u_j and u_k . If π_j is not the smallest of these three values, then π_i , π_j , and π_k certainly do not form a 213 permutation pattern. If π_j is the smallest of these three values, then, since $\text{pre}(u_i) < \text{pre}(u_j)$ but $\text{post}(u_i) > \text{post}(u_j)$, u_i is not an ancestor or descendant of u_j , and u_j is an ancestor of u_k . Therefore u_i is also not an ancestor or descendant of u_k , from which it follows that $\pi_i > \pi_k$ and the pattern formed by π_i , π_j , and π_k is 312 rather than 213. Since the choice of indices was arbitrary, no three indices can form a 213 pattern and π is 213-avoiding. \square

We observe that $\text{cperm}(G)$ has some additional structure, as well: its first element is 1, its second element is n , and its last element is 2.

3.2 Stretching a Permutation

It is natural to represent a permutation σ by the points with Cartesian coordinates (i, σ_i) , but for our purposes we need to stretch this representation in the vertical direction; we use a transformation closely related to one used by Bukh, Matoušek, and Nivasch [21] for weak epsilon-nets, and by Fulek and Tóth [11] for universal point sets for plane 3-trees. Letting $q = |\sigma|$, we define

$$\text{stretch}(\sigma) = \{(i, q^{\sigma_i}) \mid 1 \leq i \leq q\}.$$

Lemma 3. *Let σ be an arbitrary permutation with $|\sigma| = q$, let p_i denote the point in $\text{stretch}(\sigma)$ corresponding to position i in σ , let i and j be two indices with $\sigma_i < \sigma_j$, and let m be the absolute value of the slope of line segment $p_i p_j$. Then $q^{\sigma_j - 1} \leq m < q^{\sigma_j}$.*

Proof. The minimum value of m is obtained when $|i - j| = q - 1$ and $\sigma_i = \sigma_j - 1$, for which $q^{\sigma_j - 1} = m$. The maximum value of m is obtained when $|i - j| = 1$ and $\sigma_i = 1$, for which $m = q^{\sigma_j} - q < q^{\sigma_j}$. \square

Lemma 4. *Let σ be an arbitrary permutation with $|\sigma| = q$, let p_i denote the point in $\text{stretch}(\sigma)$ corresponding to position i in σ , and let i , j , and k be three indices with $\max\{\sigma_i, \sigma_j\} < \sigma_k$ and $i < j$. Then the clockwise ordering of the three points p_i , p_j , and p_k is p_i, p_k, p_j .*

Proof. The result follows by using Lemma 3 to compare the slopes of the two line segments $p_i p_j$ and $p_i p_k$. \square

Lemma 5. *Let σ be an arbitrary permutation with $|\sigma| = q$, let p_i denote the point in $\text{stretch}(\sigma)$ corresponding to position i in σ , and let h , i , j , and k be four indices with $\max\{\sigma_h, \sigma_i, \sigma_j\} < \sigma_k$ and $h < j$. Then line segments $p_h p_j$ and $p_i p_k$ cross if and only if both $h < i < j$ and $\max\{\sigma_h, \sigma_j\} > \sigma_i$.*

Proof. A crossing occurs between two line segments if and only if the endpoints of each segment are on opposite sides of the line through the other segment. The endpoints of $p_i p_k$ are on opposite sides of line $p_h p_j$ if and only if the two triangles $p_h p_i p_j$ and $p_h p_k p_j$ have opposite orientations; with the assumption that σ_k is the largest of the three values, this is equivalent by Lemma 4 to the condition that σ_i is not the second-largest. The endpoints of $p_h p_j$ are on opposite sides of line $p_i p_k$ if and only if the two triangles $p_i p_h p_k$ and $p_i p_j p_k$ have opposite orientations; this is equivalent by Lemma 4 to the condition that $h < i < j$. \square

3.3 Universal Point Sets

If σ is any permutation, we define $\text{augment}(\sigma)$ to be a permutation of length $|\sigma| + 3$, in which the first element is 1, the second element is $|\sigma| + 3$, the last element is 2, and the remaining elements form a pattern of type σ . It follows from Lemma 2 that, if σ is an $S_{n-3}(213)$ -superpattern and if G is an arbitrary n -vertex maximal plane graph, then $\text{cperm}(G)$ is a pattern in $\text{augment}(\sigma)$.

Theorem 1. *Let σ be an $S_{n-3}(213)$ -superpattern, and let $U_n = \text{stretch}(\text{augment}(\sigma))$. Then U_n is a universal point set for planar graphs on n vertices.*

Proof. Let G be an arbitrary maximal plane graph, let v_1, v_2, \dots, v_n be a canonical representation of G , and let x_i denote a sequence of positions in $\text{augment}(\sigma)$ that form a pattern of type $\text{cperm}(G)$, with position x_i in $\text{augment}(\sigma)$ corresponding to position $\text{pre}(v_i)$ in $\text{cperm}(G)$. Let $q = |\text{augment}(\sigma)|$, and for each i , let $y_i = q^j$ where j is the value of $\text{augment}(\sigma)$ at position x_i . Embed G by placing vertex v_i at the point $(x_i, y_i) \in U_n$.

Let v_h, v_i, v_j , and v_k be four vertices in G such that $v_h v_j$ and $v_i v_k$ are edges in G . We may choose these indices in such a way that $\text{post}(v_k)$ is larger than the post values of the other three vertices. If these two edges crossed in the given embedding of G , then by Lemma 5 we would necessarily have $\text{pre}(v_h) < \text{pre}(v_i) < \text{pre}(v_j)$, and $\text{post}(v_i) < \max\{\text{post}(v_h), \text{post}(v_j)\}$. By Lemma 1, v_i would not be on the outside face of the graph induced by the vertices with post values at most $\max\{\text{post}(v_h), \text{post}(v_j)\}$, and could not be a neighbor of v_k in the canonical representation given by the post values. This contradiction shows that no crossing is possible, so the embedding is planar. \square

4 $S_n(213)$ -Superpatterns

In this section we construct a $S_n(213)$ -superpattern of size $n^2/4 + n + ((-1)^n - 1)/8$. An exhaustive computer search has shown that this size is minimal for $n \leq 6$. We start with a lemma about $S_n(213)$ -superpatterns with n rows and n columns (the minimal amount of each), demonstrating their recursive structure.

Lemma 6. *If σ is a $S_n(213)$ -superpattern and has n rows and n columns, then the permutation described by the intersection of columns $n - j + 1$ to $n - i + 1$ and rows i to j of σ is a $S_{j-i+1}(213)$ -superpattern.*

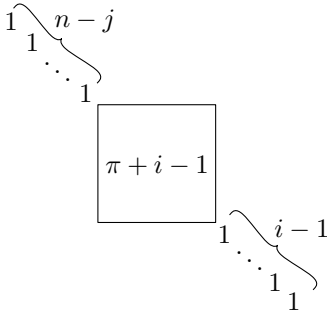


Fig. 2. The permutation τ constructed from π in the proof of Lemma 6

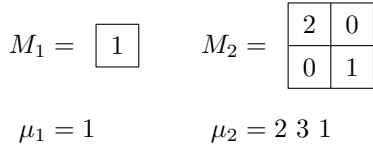


Fig. 3. The base case permutations for constructing μ_n for $n > 2$ and their chessboard representations

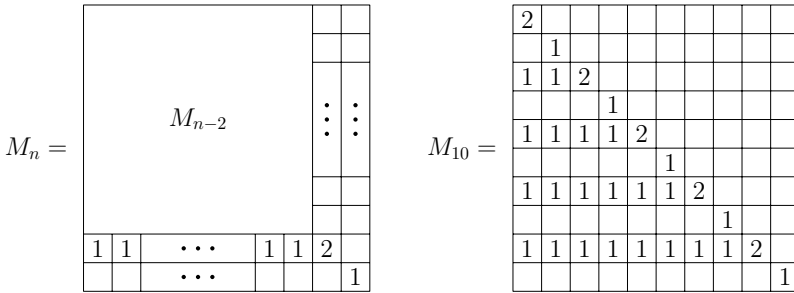


Fig. 4. The inductive construction of $\text{chessboard}(\mu_n)$ from $\text{chessboard}(\mu_{n-2})$. Cells of the matrix containing zero are shown as blank

Proof. Let π be an arbitrary 213-avoiding permutation of length $j - i + 1$ and consider the n -element 213-avoiding permutation

$$\tau = n(n - 1) \dots (j + 1)(\pi_1 + i - 1)(\pi_2 + i - 1) \dots (\pi_{j-i+1} + i - 1)(i - 1)(i - 2) \dots 321.$$

(See Fig. 2.) By the assumption that σ is a superpattern, τ has an embedding into σ . Because there are $n - j$ descents in τ before the first element of the form $\pi_i + i - 1$, this embedding cannot place any element $\pi_i + i - 1$ into the first $n - j$ columns of σ . Similarly because there are i descents in τ after the last element of the form $\pi_i + i - 1$, this embedding cannot place any element $\pi_i + i - 1$ into the last $i - 1$ columns of σ . By a symmetric argument, the elements of the form $\pi_i + i - 1$ cannot be embedded into the $i - 1$ lowest rows nor the $n - j$ highest rows of σ . Therefore these elements, which form a pattern of type π , must be embeddable into σ inclusively between column $n - j + 1$, column $n - i + 1$, row i , and row j . Since π was arbitrary, this part of σ must be universal for permutations of length $j - i + 1$, as claimed. \square

We define a permutation μ_n , which will be shown to be a $S_n(213)$ -superpattern, by describing $\text{chessboard}(\mu_n) = M_n$. In our construction M_n and μ_n have n columns and

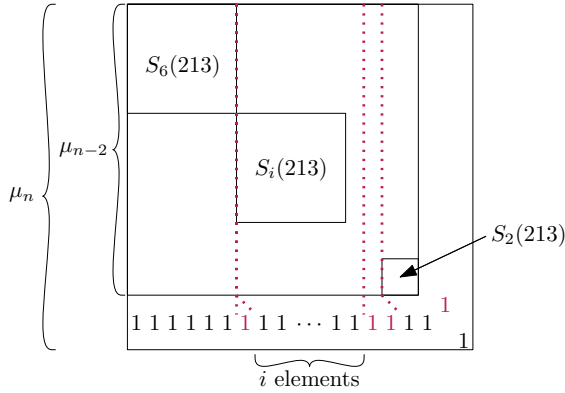


Fig. 5. A partial embedding of the red elements, showing where the remaining blocks can be fit into the columns of μ_{n-2}

n rows. The bottom two rows of M_n contain the values $M_n(n, 1) = M_n(i, 2) = 1$ for $1 \leq i \leq n - 2$, and $M_n(n - 1, 2) = 2$, with all other values in these rows zero. The values in the top $n - 2$ rows are given recursively by $M_n(1 : n - 2, 3 : n) = M_{n-2}$, again with all values outside this submatrix zero. The base cases of μ_1 and μ_2 are shown in Figure 3 and the inductive definition and an example are shown in Figure 4.

Theorem 2. *The permutation μ_n is a $S_n(213)$ -superpattern. Thus there exists a $S_n(213)$ -superpattern whose size is $n^2/4 + n + ((-1)^n - 1)/8$.*

Proof. It can be easily verified that μ_i is a $S_i(213)$ -superpattern when $1 \leq i \leq 2$. Let π be an arbitrary 213-avoiding permutation of length n . We will show that π can be embedded into μ_n .

Case 1: $\pi_n = 1$

Let $\pi_{i_1} \dots \pi_{i_k}$ be the second lowest row of π . Because π is 213-avoiding, $i_k = n - 1$ and for all j , $\pi_{i_j} = j + 1$. We embed this bottom row by mapping π_n to the bottom right element of μ_n and π_{i_j} to the i_j -th position of the second lowest row of μ_n .

Case 2: $\pi_n \neq 1$

Let $\pi_{i_1} \dots \pi_{i_k}$ be the lowest row of π . Similarly to Case 1, because π is 213-avoiding, $i_k = n$ and for all j , $\pi_{i_j} = j$. We embed this bottom row by mapping π_{i_j} to the i_j -th position of the second lowest row of μ_n .

These partial embeddings maintain the ordering of the lowest and possibly second lowest row of π . To finish the embedding, the remaining elements need to be fit into the copy of μ_{n-2} . Recall that a block of a permutation is a contiguous subsequence of consecutive values. Because π is 213-avoiding, the remaining elements of π form disjoint blocks that fit between the elements embedded so far. If one block is to the right of another in π , it has smaller values. Let π_{i_1} be the leftmost element that has been embedded on the second row of μ_n . Then there are $i_1 - 1$ elements before it in π and $i_1 - 1$ columns before where it was embedded in μ_n . By Lemma 6, these elements

can fit into the last $i_1 - 1$ rows of these columns. Let π_{i_j} and $\pi_{i_{j+1}}$ be two adjacent elements embedded in the second lowest row of μ_n . Between these two there are at least $i_{j+1} - i_j - 1$ columns of μ_{n-2} available: the column above π_{i_j} to the column before $\pi_{i_{j+1}}$. So again by Lemma 6, the $i_{j+1} - i_j - 1$ elements between π_{i_j} and $\pi_{i_{j+1}}$ can be fit into the last $i_{j+1} - i_j - 1$ rows of those columns. (See Fig. 5) Because $i_k = n - 1$ in Case 1 and n in Case 2, there is no block after π_{i_k} . Therefore π can be embedded into μ_n and μ_n is a $S_n(213)$ -superpattern. \square

Combining Theorem 2 with Theorem 1, the following is immediate:

Theorem 3. *The n -vertex planar graphs have universal point sets of size $n^2/4 - \Theta(n)$.*

5 Dominance Drawing

A *dominance drawing* of a directed acyclic graph [22] is a drawing of the graph in the plane such that each edge is directed upwards and to the right, such that the axis-aligned bounding box of every edge contains no vertices other than its endpoints, and such that no edge can be added to the drawing preserving these properties. The graphs with planar dominance drawings are exactly the transitively reduced *st*-planar graphs, i.e. the planar directed acyclic graphs in which there is one source and one sink, both on the outer face, and in which each edge forms the only directed path connecting its two endpoints.

If a graph has a dominance drawing D , then it has a drawing in which the points are in general position, and the points in this case can be thought of as representing a permutation π_D , where the positions of the elements in the permutation are the positions of the points in the sorted order by their x coordinates and the values of these elements are the positions in the sorted order by the y coordinates. Any two point sets with the same two sorted orders may be used as the basis for a dominance drawing combinatorially equivalent to D . In particular, if π_D appears as a pattern in another permutation σ , then the subset of the points (i, σ_i) corresponding to elements of π_D may be used to draw the same graph. This gives us the following result:

Theorem 4. *If σ is a superpattern for the length- n permutations, then the set of points (i, σ_i) is universal for dominance drawings of n -vertex transitively reduced *st*-planar graphs.*

Combining this result with Miller’s bound on superpatterns [16] shows that dominance drawings have universal point sets of size $n^2/2 + \Theta(n)$, half the size of the point sets given by previous methods based on $n \times n$ grids.

Not every permutation is of the form π_D for a planar dominance drawing D ; for instance the permutation 2143 corresponds to a drawing that has a crossing. However, every permutation π forms a pattern in a larger permutation σ that does define a planar dominance drawing, constructed from the Dedekind–MacNeille completion of a partially ordered set associated to π [23]. For this reason, the permutations that define planar dominance drawings have no forbidden patterns. However, in later research, we have shown that the dominance drawings of some other classes of graphs have forbidden patterns, leading to smaller universal sets for these drawings [24].

6 Additional Results

In the full version of the paper, we provide the following results.

- We prove that, for every 213-avoiding permutation ϕ , the $\{213, \phi\}$ -avoiding permutations have superpatterns of near-linear size. In particular, the $\{213, 312\}$ -avoiding permutations and the $\{213, 3412\}$ -avoiding permutations have superpatterns of linear size, and the minimum size of a superpattern for the $\{213, 132\}$ -avoiding permutations is $\Theta(n \log n)$ (despite these permutations being equinumerous with the $\{213, 312\}$ -avoiding permutations). We define the *Strahler number* of any 213-avoiding permutation from a forest derived from its chessboard representation, and we show that if a permutation ϕ has Strahler number s then the $\{213, \phi\}$ -avoiding permutations have superpatterns of size $O(n \log^{s-1} n)$.
- We prove that, for every integer w , there exists a pattern ϕ such that the planar graphs of pathwidth at most w correspond to a $\{213, \phi\}$ -avoiding permutation (using the same correspondence between graphs and permutations as in Section 3). As a consequence, the planar graphs of bounded pathwidth have universal point sets of size $O(n \log^{O(1)} n)$.
- We improve the bound of Angelini et al. [10] on universal point sets for simply-nested planar graphs from $O(n(\log n / \log \log n)^2)$ to $O(n \log n)$.

For space reasons we defer the proofs of these results to the full version of the paper.

7 Conclusion

In this paper we have constructed universal point sets for planar graphs of size $n^2/4 - \Theta(n)$, and of subquadratic size for graphs of bounded pathwidth. In the process of building these constructions we have provided a new connection between universal point sets and permutation superpatterns. We have also, for the first time, provided nontrivial upper bounds and lower bounds on the size of superpatterns for restricted classes of permutations. We leave the following problems open for future research:

- Which natural subclasses of planar graphs (beyond the bounded-pathwidth graphs) can be represented by permutations in a proper subclass of $S_n(213)$?
- Can we reduce the gap between our $O(n^2)$ upper bound and $\Omega(n \log n)$ lower bound for $S_n(213)$ -superpatterns?
- Our construction uses area exponential in n^2 ; how does constraining the area to a smaller bound affect the number of points in a universal point set?

References

1. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. *Combinatorica* 10(1), 41–51 (1990); Originally presented at STOC 1988
2. Chrobak, M., Payne, T.: A linear-time algorithm for drawing a planar graph on a grid. *Inform. Proc. Lett.* 54(4), 241–246 (1995)

3. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. 1st ACM–SIAM Symp. on Discrete Algorithms, pp. 138–148 (1990)
4. Brandenburg, F.J.: Drawing planar graphs on $\frac{8}{9}n^2$ area. In: Proc. Int. Conf. Topological and Geometric Graph Theory. Electronic Notes in Discrete Mathematics, vol. 31, pp. 37–40. Elsevier (2008)
5. Dolev, D., Leighton, T., Trickey, H.: Planar embedding of planar graphs. *Advances in Computing Research* 2, 147–161 (1984)
6. Kurowski, M.: A 1.235 lower bound on the number of points needed to draw all n -vertex planar graphs. *Inform. Proc. Lett.* 92(2), 95–98 (2004)
7. Mondal, D.: Embedding a Planar Graph on a Given Point Set. Master’s thesis, Dept. of Computer Science, U. of Manitoba (2012)
8. Demaine, E., O’Rourke, J.: Smallest Universal Set of Points for Planar Graphs (Problem 45). In: Demaine, E., Mitchell, J.S.B., O’Rourke, J. (eds.) *The Open Problems Project* (2002–2012)
9. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified positions. *Amer. Math. Monthly* 98(2), 165–166 (1991)
10. Angelini, P., Di Battista, G., Kaufmann, M., Mchedlidze, T., Roselli, V., Squarcella, C.: Small point sets for simply-nested planar graphs. In: van Kreveld, M., Speckmann, B. (eds.) *GD 2011. LNCS*, vol. 7034, pp. 75–85. Springer, Heidelberg (2011)
11. Fulek, R., Tóth, C.D.: Universal point sets for planar three-trees. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) *WADS 2013. LNCS*, vol. 8037, pp. 341–352. Springer, Heidelberg (2013)
12. Eppstein, D.: Drawing arrangement graphs in small grids, or how to play Planarity. In: Wismath, S., Wolff, A. (eds.) *GD 2013. LNCS*, vol. 8242, pp. 436–447. Springer, Heidelberg (2013)
13. Bereg, S., Holroyd, A.E., Nachmanson, L., Pupyrev, S.: Drawing permutations with few corners. In: Wismath, S., Wolff, A. (eds.) *GD 2013. LNCS*, vol. 8242, pp. 488–499. Springer, Heidelberg (2013)
14. Arratia, R.: On the Stanley–Wilf conjecture for the number of permutations avoiding a given pattern. *Elect. J. Comb.* 6, N1 (1999)
15. Eriksson, H., Eriksson, K., Linusson, S., Wästlund, J.: Dense packing of patterns in a permutation. *Ann. Comb.* 11(3–4), 459–470 (2007)
16. Miller, A.: Asymptotic bounds for permutations containing many different patterns. *J. Comb. Theory A* 116(1), 92–108 (2009)
17. Marcus, A., Tardos, G.: Excluded permutation matrices and the Stanley–Wilf conjecture. *J. Comb. Theory A* 107(1), 153–160 (2004)
18. Rotem, D.: Stack sortable permutations. *Discrete Math.* 33(2), 185–196 (1981)
19. Knuth, D.: Vol. 1: Fundamental Algorithms. In: *The Art of Computer Programming*. Addison-Wesley, Reading (1968)
20. Dietz, P.F.: Maintaining order in a linked list. In: Proc. 15th ACM Symp. on Theory of Computing, pp. 122–127 (1982)
21. Bukh, B., Matoušek, J., Nivasch, G.: Lower bounds for weak epsilon-nets and stair-convexity. *Israel J. Math.* 182, 199–208 (2011)
22. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: 4.7 Dominance drawings. In: *Graph Drawing: Algorithms for the Visualization of Graphs*, pp. 112–127. Prentice-Hall (1999)
23. Eppstein, D., Simons, J.A.: Confluent Hasse diagrams. In: van Kreveld, M., Speckmann, B. (eds.) *GD 2011. LNCS*, vol. 7034, pp. 2–13. Springer, Heidelberg (2011)
24. Bannister, M.J., Devanny, W.E., Eppstein, D.: Small superpatterns for dominance drawing (manuscript 2013)

Simultaneous Embedding: Edge Orderings, Relative Positions, Cutvertices^{*}

Thomas Bläsius, Annette Karrer, and Ignaz Rutter^{**}

Karlsruhe Institute of Technology (KIT)
firstname.lastname@kit.edu

Abstract. A simultaneous embedding of two graphs $G^{\circledast 1}$ and $G^{\circledast 2}$ with common graph $G = G^{\circledast 1} \cap G^{\circledast 2}$ is a pair of planar drawings of $G^{\circledast 1}$ and $G^{\circledast 2}$ that coincide on G . It is an open question whether there is a polynomial-time algorithm that decides whether two graphs admit a simultaneous embedding (problem SEFE).

In this paper, we present two results. First, a set of three linear-time preprocessing algorithms that remove certain substructures from a given SEFE instance, producing a set of equivalent SEFE instances without such substructures. The structures we can remove are (1) cutvertices of the *union graph* $G^{\circledast 1} \cup G^{\circledast 2}$, (2) cutvertices that are simultaneously a cutvertex in $G^{\circledast 1}$ and $G^{\circledast 2}$ and that have degree at most 3 in G , and (3) connected components of G that are biconnected but not a cycle.

Second, we give an $O(n^2)$ -time algorithm for SEFE where, for each pole u of a P-node μ (of a block) of the input graphs, at most three virtual edges of μ contain common edges incident to u . All algorithms extend to the sunflower case.

1 Introduction

A simultaneous embedding of two graphs $G^{\circledast 1}$ and $G^{\circledast 2}$ with common graph $G = G^{\circledast 1} \cap G^{\circledast 2}$ is a pair of planar drawings of $G^{\circledast 1}$ and $G^{\circledast 2}$, that coincide on G . The problem to decide whether a simultaneous embedding exists is called SEFE (simultaneous embedding with fixed edges). This definition extends to more than two graphs. For three graphs SEFE is NP-complete [7]. In the *sunflower case* it is required that every pair of input graphs has the same intersection. See [2] for a survey on SEFE and related problems.

There are two fundamental approaches to solving SEFE in the literature. The first approach is based on the characterization of Jünger and Schulz [10] stating that finding a simultaneous embedding of two graphs $G^{\circledast 1}$ and $G^{\circledast 2}$ with common graph G is equivalent to finding planar embeddings of $G^{\circledast 1}$ and $G^{\circledast 2}$ that induce the same embedding on G . The second very recent approach by Schaefer [11] is based on Hanani-Tutte-style redrawing results. One tries to characterize the existence of a SEFE via the existence of drawings where no two independent edges of the same graph cross an odd number of times. The existence of such drawings can be expressed using a linear system of boolean equations.

When following the first approach, we need two things to describe the planar embedding of the common graph G . First, for each vertex v , a cyclic order of incident edges

^{*} Partly done within GRADR – EUROGIGA project no. 10-EuroGIGA-OP-003.

^{**} Work was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

around v . Second, for every pair of connected components H and H' of G , the face f of H containing H' . We call this relationship the *relative position* of H' with respect to H . To find a simultaneous embedding, one needs to find a pair of planar embeddings that induce the same cyclic edge orderings (*consistent edge orderings*) and the same relative positions (*consistent relative positions*) on the common graph G .

Most previous results use the first approach but none of them considers both consistent edge orderings and relative positions. Most of them assume the common graph to be connected or to contain no cycles. The strongest results of this type are the two linear-time algorithms for the case that G is biconnected by Haeupler et al. [9] and by Angelini et al. [1] and a quadratic-time algorithm for the case where $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ are biconnected and G is connected [4]. In the latter result, SEFE is modeled as an instance of the problem SIMULTANEOUS PQ-ORDERING. On the other hand, there is a linear-time algorithm for SEFE if the common graph consists of disjoint cycles [3], which requires to ensure consistent relative positions but makes edge orderings trivially consistent.

The advantage of the second approach is that it implicitly handles both, consistent edge orderings and consistent relative positions, at the same time. Thus, the results by Schaefer [11] are the first that handle SEFE instances where the common graph consists of several, non-trivial connected components. He gives a polynomial-time algorithm for the cases where each connected component of the common graph is biconnected or has maximum degree 3. Although this approach is conceptionally simple, very elegant, and combines several notions of planarity within a common framework, it has two disadvantages. The running time of the algorithms are quite high and the high level of abstraction makes it difficult to generalize the results, e.g., to the sunflower case.

Contribution & Outline. In this paper, we follow the first approach and show how to enforce consistent edge orderings and consistent relative positions at the same time, by combining different recent approaches, namely the algorithm by Angelini et al. [1] and result on SIMULTANEOUS PQ-ORDERING [4] for consistent edge orderings and the result on disjoint cycles [3] for consistent relative positions. To handle relative positions of connected components to each other without knowing their embedding, we show that these relative positions can be expressed in terms of relative positions with respect to a cycle basis. In addition to that, we are able to handle certain cutvertices of $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$.

More precisely, we classify a vertex v to be a *union cutvertex*, a *simultaneous cutvertex*, and an *exclusive cutvertex* if v is a cutvertex of $G^{\textcircled{1}} \cup G^{\textcircled{2}}$, of $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ but not of $G^{\textcircled{1}} \cup G^{\textcircled{2}}$, and of $G^{\textcircled{1}}$ but not $G^{\textcircled{2}}$ or the other way around, respectively. We say that v has *common-degree* k if it is a common vertex with degree k in G . We present three preprocessing algorithms that simplify given instances of SEFE; see Section 3. They remove union cutvertices and simultaneous cutvertices with common-degree 3 (note that simultaneous cutvertices with common degree less than 3 cannot exist), and replace connected components of G that are biconnected by cycles. They run in linear time and can be applied independently. The latter algorithm together with the linear-time algorithm for disjoint cycles [3] improves the result by Schaefer [11] for instances where every connected component of G is biconnected to linear time and the sunflower case.

In Section 4 we show how to solve instances that have *common P-node degree* 3 and contain neither union nor simultaneous cutvertices in quadratic time. An instance has common P-node degree k if, for each pole u of a P-node μ (of a block) of the input

graphs, at most k virtual edges of μ contain common edges incident to u . Together with the preprocessing steps, this includes the case where every connected component of G is biconnected, has maximum degree 3, or is outerplanar with maximum degree 3 cutvertices. As before, this approach also applies to the sunflower case.

2 Preliminaries

Connectivity & SPQR-trees. A graph is *connected* if there exists a path between any pair of vertices. A *separating k -set* is a set of k vertices whose removal disconnects the graph. Separating 1-sets and 2-sets are *cutvertices* and *separation pairs*, respectively. A connected graph is *biconnected* if it has no cut vertex and *triconnected* if it has no separation pair. The maximal biconnected components of a graph are called *blocks*. The *split components* with respect to a separating k -set are the maximal subgraphs that are not disconnected by removing the separating k -set.

The SPQR-tree \mathcal{T} of a biconnected graph G represents the decomposition of G along its *split pairs*, where a split pair is either a separating pair or a pair of adjacent vertices [6]. We consider the SPQR-tree to be unrooted, representing embeddings on the sphere, i.e., planar embeddings without a designated outer face.

Let $\{s, t\}$ be a split pair and let H_1 and H_2 be two subgraphs of G such that $H_1 \cup H_2 = G$ and $H_1 \cap H_2 = \{s, t\}$. Consider the tree consisting of two nodes μ_1 and μ_2 associated with the graphs $H_1 + \{s, t\}$ and $H_2 + \{s, t\}$, respectively. These graphs are called *skeletons* of the nodes μ_i , denoted by $\text{skel}(\mu_i)$, and the special edge $\{s, t\}$ is a *virtual edge*. The edge connecting the nodes μ_1 and μ_2 associates the virtual edges in $\text{skel}(\mu_1)$ and $\text{skel}(\mu_2)$ with each other. The *expansion graph* of a virtual edge $\{s, t\}$ is the subgraph of G it represents, that is in $\text{skel}(\mu_1)$ the expansion graph of $\{s, t\}$ is H_2 and the expansion graph of $\{s, t\}$ in $\text{skel}(\mu_2)$ is H_1 . A combinatorial embedding of G uniquely induces a combinatorial embedding of $\text{skel}(\mu_1)$ and $\text{skel}(\mu_2)$ and vice versa.

Applying this kind of decomposition systematically yields the SPQR-tree \mathcal{T} . The skeletons of the internal nodes of \mathcal{T} are either a cycle (S-node), a bunch of parallel edges (P-node) or a triconnected planar graph (R-node). The leaves are Q-nodes, and their skeleton consists of two vertices connected by a virtual and a normal edge. Thus, the only possible embedding choices are flipping skeletons of R-nodes and ordering the edges in skeletons of P-nodes. The SPQR-tree can be computed in linear time [8].

Let \mathcal{T}^\circledast by the SPQR-tree of a block of G^\circledast in an instance of SEFE and let G be the common graph. Let further μ be a P-node of \mathcal{T}^\circledast . We say that μ has *common P-node degree k* if both vertices in $\text{skel}(\mu)$ are incident to common edges in the expansion graphs of at most k virtual edges (note that these can be different edges for the two vertices). We say that G^\circledast has common P-node degree k if each P-node in the SPQR-tree of each block of G^\circledast has common P-node degree k . If this is the case for G^\circledast and G^\circledast , we say that the instance of SEFE has common P-node degree k .

PQ-trees. A PQ-tree, originally introduced by Booth and Lueker [5], is a tree, whose inner nodes are either P-nodes or Q-nodes (note that these P-nodes have nothing to do with the P-nodes of the SPQR-tree). The order of edges around a P-node can be ordered arbitrarily, the edges around a Q-node are fixed up to a flip. In this way, a PQ-tree represents a set of orders on its leaves. A rooted PQ-tree represents linear orders, an unrooted

PQ-tree represents cyclic orders (in most cases we consider unrooted PQ-trees). Given a PQ-tree T and a subset S of its leaves, there exists another PQ-tree T' representing exactly the orders represented by T where the elements in S are consecutive. The tree T' is the *reduction* of T with respect to S . The *projection* of T to S is a PQ-tree with leaves S representing exactly the orders on S that are represented by T .

The problem SIMULTANEOUS PQ-ORDERING has several PQ-trees sharing some leaves as input, that are related by identifying some of their leaves [4]. More precisely, every instance is a directed acyclic graph, where each node is a PQ-tree, and each arc (T, T') has the property that there is an injective map from the leaves of the *child* T' to the leaves of the *parent* T . For each PQ-tree in such an instance, one wants to find an order of its leaves such that for every arc (T, T') the order chosen for the parent T is an extension of the order chosen for the child T' (with respect to the injective map). We will later use instances of SIMULTANEOUS PQ-ORDERING to express relations between orderings of edges around vertices.

3 Preprocessing Algorithms

In this section, we present several algorithms that can be used as a preprocessing of a given SEFE instance. The result is usually a set of SEFE instances that admit a solution if and only if the original instance admits one. The running time of the preprocessing algorithms is linear, and so is the total size of the equivalent set of SEFE instances. Each of the preprocessing algorithms removes certain types of structures from the instance, in particular from the common graph. Namely, we show that we can eliminate union cutvertices, simultaneous cutvertices with common-degree 3, and connected components of G that are biconnected but not a cycle. None of these algorithms introduces new cutvertices in G or increases the degree of a vertex. Thus, the preprocessing algorithms can be successively applied to a given instance, removing all the claimed structures.

Let (G°, G^\ominus) be a SEFE instance with common graph $G = G^\circ \cap G^\ominus$. We can equivalently encode such an instance in terms of its *union graph* $G_\cup = G^\circ \cup G^\ominus$, whose edges are labeled $\{1\}$, $\{2\}$, or $\{1, 2\}$, depending on whether they are contained exclusively in G° , exclusively in G^\ominus , or in G , respectively. Any graph with such an edge coloring can be considered as a SEFE instance. Since sometimes the coloring version is more convenient, we use these notions interchangeably throughout this section.

3.1 Union Cutvertices and Simultaneous Cutvertices

It is not hard to see that union cutvertices of a SEFE instance can be used to split it into independent instances. A simultaneous cutvertex with common-degree 3 can be modified as in Fig. 1, yielding an equivalent instance. Exhaustively applying these ideas, yields the following results; proofs are omitted due to space constraints.

Theorem 1. *There is a linear-time algorithm that decomposes a SEFE instance into an equivalent set of SEFE instances that do not contain union cutvertices.*

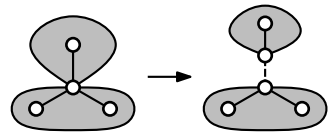


Fig. 1. A simultaneous cutvertex with common-degree 3. The gray regions are the split components of G° , the new dashed edge belongs to G^\ominus .

Theorem 2. *Let $(G^{\circledast}, G^{\circledcirc})$ be an instance of SEFE such that every simultaneous cutvertex has common-degree 3. An equivalent instance without simultaneous cutvertices can be computed in linear time.*

3.2 Connected Components that are Biconnected

Let $(G^{\circledast}, G^{\circledcirc})$ be a SEFE instance. Throughout this section, we assume without loss of generality that G^{\circledast} and G^{\circledcirc} are connected [3] and that the common graph G is an induced subgraph of G^{\circledast} and G^{\circledcirc} . The latter can be achieved by subdividing each exclusive edge once, which clearly does not alter the existence of a SEFE.

Let C be a connected component of G that is a cycle. A *union bridge* of G^{\circledast} and G^{\circledcirc} with respect to C is a connected component of $G_{\cup} - C$ together with all its attachment vertices on C . Similarly, there are $\textcircled{1}$ -bridges and $\textcircled{2}$ -bridges, which are connected components of $G^{\circledast} - C$ and $G^{\circledcirc} - C$ together with their attachment vertices on C , respectively. We say that two bridges B_1 and B_2 *alternate* if there are attachments a_1, b_1 of B_1 and attachments a_2, b_2 of B_2 , such that the order along C is $a_1 a_2 b_1 b_2$. We have the following lemma.

Lemma 1. *Let G^{\circledast} and G^{\circledcirc} be two planar graphs and let C be a connected component of the common graph that is a cycle. Then the graphs G^{\circledast} and G^{\circledcirc} admit a SEFE where C is the boundary of the outer face if and only if (i) each union bridge admits a SEFE together with C and (ii) no two $\textcircled{1}$ -bridges of C alternate for $i = 1, 2$.*

Proof. Clearly the conditions are necessary; we prove sufficiency. Let B_1, \dots, B_k be the union bridges with respect to C , and let $(\mathcal{E}_1^{\circledast}, \mathcal{E}_1^{\circledcirc}), \dots, (\mathcal{E}_k^{\circledast}, \mathcal{E}_k^{\circledcirc})$ be the corresponding simultaneous embeddings of B_i together with C , which exist by condition (i). Note that each union bridge is connected, and hence all its edges and vertices are embedded on the same side of C . After possibly flipping some of the embeddings, we may assume that each of them has C with the same clockwise orientation as the outer face.

We now glue $\mathcal{E}_1^{\circledast}, \dots, \mathcal{E}_k^{\circledast}$ to an embedding $\mathcal{E}^{\circledast}$ of G^{\circledast} , which is possible by condition (ii). In the same way, we find an embedding $\mathcal{E}^{\circledcirc}$ of G^{\circledcirc} from $\mathcal{E}_1^{\circledcirc}, \dots, \mathcal{E}_k^{\circledcirc}$. We claim that $(\mathcal{E}_1, \mathcal{E}_2)$ is a SEFE of G^{\circledast} and G^{\circledcirc} . For the consistent edge orderings, observe that any common vertex v with common-degree at least 3 is contained, together with all neighbors, in some union bridge B_i . The compatibility of the edge ordering follows since $(\mathcal{E}_i^{\circledast}, \mathcal{E}_i^{\circledcirc})$ is a SEFE. Concerning the relative position of a vertex v and some common cycle C' , we note that the relative positions clearly coincide in $\mathcal{E}^{\circledast}$ and $\mathcal{E}^{\circledcirc}$ for $C = C'$. Otherwise C' is contained in some union bridge. If v is embedded in the interior of C' in one of the two embeddings, then it is contained in the same union bridge as C' , and the compatibility follows. If this case does not apply, it is embedded outside of C' in both embeddings, which is compatible as well. \square

Now consider a connected component C of the common graph G of a SEFE instance such that C is biconnected. Such a component is called *2-component*. If C is a cycle, it is a *trivial 2-component*. We define the union bridges, and the $\textcircled{1}$ - and $\textcircled{2}$ -bridges of G^{\circledast} and G^{\circledcirc} with respect to C as above. We call an embedding \mathcal{E} of C together with an assignment of the union bridges to its faces *admissible* if and only if, (i) for each union bridge, all attachments are incident to the face to which it is assigned, and (ii) no

two ①- or ②-bridges that are assigned to the same face alternate. For a union bridge B , let C_B denote the cycle consisting of the attachments of B in the ordering of an arbitrary cycle of G containing all the attachments. It can be shown that this cycle is uniquely determined. Let G_B denote the graph consisting of B and C_B . We call these graphs the *union bridge graphs*.

Lemma 2. *Let $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ be two connected planar graphs and let C be 2-component of the common graph G . Then the graphs $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ admit a SEFE if and only if (i) C admits an admissible embedding, and (ii) each union bridge graph admits a SEFE. If a SEFE exists, the embedding of C can be chosen as an arbitrary admissible embedding.*

Proof. Clearly, a SEFE of $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ defines an embedding of C and a bridge assignment that is admissible. Moreover, it induces a SEFE of each union bridge graph.

Conversely, assume that C admits an admissible embedding and each union bridge graph admits a SEFE. We obtain a SEFE of $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ as follows. Embed C with the admissible embedding and consider a face f of this embedding with facial cycle C_f . Let B_1, \dots, B_k denote the union bridges that are assigned to this face, and let $(\mathcal{E}_1^{\textcircled{1}}, \mathcal{E}_1^{\textcircled{2}}), \dots, (\mathcal{E}_k^{\textcircled{1}}, \mathcal{E}_k^{\textcircled{2}})$ be simultaneous embeddings of the bridge graphs G_B . By subdividing the cycle C_B , in each of the embeddings, we may assume that the outer face of each B_i in the embedding $(\mathcal{E}_i^{\textcircled{1}}, \mathcal{E}_i^{\textcircled{2}})$ is the facial cycle C_f with the same orientation in each of them. By Lemma 1, we can hence combine them to a single SEFE of all union bridges whose outer face is the cycle C_f . We embed this SEFE into the face f of C . Since we can treat the different faces of C independently, applying this step for each face yields a SEFE of $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ with the claimed embedding of C . \square

Lemma 2 suggests a simple strategy for reducing SEFE instances containing non-trivial 2-components. Namely, take such a component, construct the corresponding union bridge graphs, where C occurs only as a cycle, and find an admissible embedding of C . Finding an admissible embedding for C can be done as follows. To enforce the non-overlapping attachment property, replace each ①-bridge of C by a *dummy ①-bridge* that consists of a single vertex that is connected to the attachments of that bridge via edges in $E^{\textcircled{1}}$. Similarly, we replace ②-bridges, which are connected to attachments via exclusive edges in $E^{\textcircled{2}}$. We seek a SEFE of the resulting instance (where the common graph is biconnected), additionally requiring that dummy bridges belonging to the same union bridge are embedded in the same face. We also refer to such an instance as *SEFE with union bridge constraints*. A slight modification of the algorithm by Angelini et al. [1] can decide the existence of such an embedding in polynomial time. It then remains to treat the bridge graphs. Exhaustively applying Lemma 2 results in a set of SEFE instances where each 2-component is trivial.

Linear-Time Decomposition. We now show that the set of instances resulting from exhaustively applying Lemma 2 can be computed in linear time.

Theorem 3. *Given a SEFE instance, an equivalent set of instances of total linear size such that each 2-component of these instances is trivial can be computed in linear time.*

Let G be a planar graph and let C_1, \dots, C_k be connected components of G . We are interested in simultaneously determining for each component C_i the number of incident bridges, and for each such bridge its attachment vertices. For this, we introduce

the notion of *subbridges*. A subbridge of G with respect to C_1, \dots, C_k is a maximal connected subgraph of G that does not become disconnected by removing all vertices of one component C_i . It is readily seen that each C_i -bridge B contains a unique subbridge S incident to C_i and that the attachments of S at C_i are exactly the attachments of B at C_i . We will thus rather work with the subbridges than the actual bridges as they represent the same information but in a more compact way. Our reduction now works in three phases.

1. Compute for each 2-component of G the number of ①-, ②-bridges, for each such bridge its attachments, and the grouping of these bridges into union bridges.
2. Find for each 2-component an admissible embedding with respect to its bridges.
3. Compute for each subbridge of G with respect to its 2-components a corresponding instance where each 2-component has been replaced by a suitable cycle.

The correctness of this approach descends from Lemma 2, since the set of instances computed by the procedure is exactly the one that can be obtained by exhaustively applying this lemma. The details of the implementation of this procedure are deferred to the full version of this paper. Here we only sketch the main ideas. For step 1, we exploit the fact that, after contracting each connected component of G that is biconnected to a single vertex, (almost) every such component is a cutvertex, and the union subbridges are essentially the blocks of the resulting graph. We can then traverse for each cutvertex its incident edges and label them by the block (subbridge) containing them. This allows us to construct the dummy-bridges and union bridges that are solved in step 2. For step 2, we modify the algorithm due to Angelini et al.[1]. Augmenting it such that it computes admissible embedding in polynomial time is straightforward. Achieving linear running time is quite technical and, like the linear version of the original algorithm, requires some intricate data structures. Step 3 is finally implemented by taking the admissible embeddings from step 2. We then traverse each such face exactly one, and construct, during this traversal, the corresponding cycles in all incident subbridges that are embedded in this face.

4 Instances with Common P-Node Degree 3

We consider instances of SEFE that have common P-node degree 3. Recall that a simultaneous embedding must induce consistent edge orderings and consistent relative positions on the common graph. We show how to address both requirements separately, by formulating necessary and sufficient constraints using linear equations over \mathbb{F}_2 . Both resulting systems of equations share all variables representing embedding choices. Satisfying both sets of linear equations at the same time then solves SEFE.

Before we can follow this strategy, we need to address one problem. The relative position of a component H' of G with respect to another connected component H , denoted by $\text{pos}_H(H')$, is the face of H containing H' . However, the set of faces of H depends on the embedding of H . To be able to handle relative positions independently from edge orderings, we need to express the relative positions independently from faces.

4.1 Relative Positions with Respect to a Cycle Basis

A *generalized cycle* C in a graph H is a subset of its edges such that every vertex of H has an even number of incident edges in C . The *sum* $C + C'$ of two generalized cycles is the symmetric difference between the edge sets, i.e., an edge e is contained in $C + C'$ if and only if it is contained in C or in C' but not in both. The resulting edge set $C + C'$ is again a generalized cycle. The set of all generalized cycles in H is a vector space over \mathbb{F}_2 . A basis of this vector space is called *cycle basis* of H .

Instead of considering the relative position $\text{pos}_H(H')$ of a connected component H' with respect to another component H , we choose a cycle basis \mathcal{C} of H and show that the relative positions of H' with respect to the cycles in \mathcal{C} suffice to uniquely define $\text{pos}_H(H')$, independent from the embedding of H . We assume H to be biconnected. All results can be extended to connected graphs by using a cycle basis for each block.

Let C_0, \dots, C_k be the set of facial cycles with respect to an arbitrary planar embedding of H . The set $\mathcal{C} = \{C_1, \dots, C_k\}$ obtained by removing one of the facial cycles is a cycle basis of G . A cycle basis that can be obtained in this way is called *planar cycle basis*. In the following we assume all cycle bases to be planar cycle bases. Moreover, we consider all cycles to have an arbitrary but fixed orientation, which has the effect, that $\text{pos}_C(p)$ for any cycle C and any point p can have either the value LEFT or RIGHT.

Theorem 4. *Let H be a planar graph embedded on the sphere, let p be a point on the sphere, and let $\mathcal{C} = \{C_1, \dots, C_k\}$ be an arbitrary planar cycle basis of H . Then the face containing p is determined by the relative positions $\text{pos}_{C_i}(p)$ for $1 \leq i \leq k$.*

Proof (sketch). Clearly, the point p and the face f containing p have to lie on the same side of each of the cycles in \mathcal{C} . It remains to show that the face with this property is unique. Let C be the facial cycle of f and let $C = C_1 + \dots + C_\ell$ be the linear combination of basis cycles of \mathcal{C} . The *position vector* of a point p with respect to the facial cycle C is $\text{pos}(p) = (\text{pos}_{C_1}(p), \dots, \text{pos}_{C_\ell}(p))$. It can be seen that inside f , the vector $\text{pos}(p)$ has a different parity of values LEFT than outside, which shows, that no other face can have the same relative positions with respect to all cycles in \mathcal{C} . \square

4.2 Consistent Edge Orderings

We first assume that the graphs G^\circledast and G^\circledcirc are biconnected. There exists an instance of SIMULTANEOUS PQ-ORDERING that has a solution if and only if G^\circledast and G^\circledcirc admit embeddings with consistent edge ordering [4]. This solution is based on the *PQ-embedding representation*, an instance of SIMULTANEOUS PQ-ORDERING representing all embeddings of a biconnected planar graph. We describe this embedding representation and show how to simplify it for instances that have common P-node degree 3.

For each vertex v^\circledast of G^\circledast , the PQ-embedding representation, denoted by $D(G^\circledast)$, contains the *embedding tree* $T(v^\circledast)$ having a leaf for each edge incident to v^\circledast , representing all possible orders of edges around v^\circledast . For every P-node μ^\circledast in the SPQR-tree \mathcal{T}^\circledast of G^\circledast that contains v^\circledast in $\text{skel}(\mu^\circledast)$ there is a P-node in $T(v^\circledast)$ representing the choice to reorder the virtual edges in $\text{skel}(\mu^\circledast)$. Similarly, for every R-node μ^\circledast in \mathcal{T}^\circledast containing v^\circledast there is a Q-node in $T(v^\circledast)$ whose flip corresponds to the flip of $\text{skel}(\mu^\circledast)$. As the orders of edges around different vertices of G^\circledast cannot be chosen independently

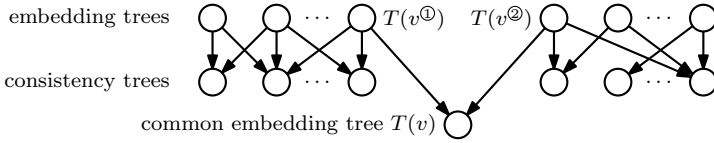


Fig. 2. The Q-embedding representations of G^{\odot} and G^{\otimes} and one common embedding tree

from each other, so called *consistency trees* are added as common children to enforce Q-nodes stemming from the same R-node in \mathcal{T}^{\odot} to have the same flip and P-nodes stemming from the same P-node to have consistent (i.e., opposite) orders. Every solution of the resulting instance corresponds to a planar embedding of G^{\odot} and vice versa [4].

As we are only interested in the order of common edges, we modify $D(G^{\odot})$ by projecting each PQ-tree to the leaves representing common edges. As G^{\odot} and G^{\otimes} have common P-node degree 3, all P-nodes of the resulting PQ-trees have degree 3 and can be assumed to be Q-nodes representing a binary decision. We call the resulting instance *Q-embedding representation* and denote it by $D(G^{\odot})$. Let μ^{\odot} be an R-node of the SPQR-tree \mathcal{T}^{\odot} whose embedding has influence on the ordering of common edges around a vertex. It is not hard to see that the Q-embedding representation contains a consistency tree consisting of a single Q-node representing the flip of $\text{skel}(\mu^{\odot})$. We associate the binary variable $\text{ord}(\mu^{\odot})$ with this decision. For a P-node μ^{\odot} we get a similar result. Let u^{\odot} and v^{\odot} be the nodes in $\text{skel}(\mu^{\odot})$. If the consistency tree enforcing a consistent decision in the embedding trees $T(u^{\odot})$ and $T(v^{\odot})$ has degree 3, its flip represents the embedding decision for $\text{skel}(\mu^{\odot})$ and we again get a binary variable $\text{ord}(\mu^{\odot})$. Otherwise, this consistency tree contains two or less leaves and can be ignored. Then the choices for the Q-nodes corresponding to μ^{\odot} in $T(u^{\odot})$ and $T(v^{\odot})$ are independent and we get one binary variable for each of these Q-nodes. We denote these variables by $\text{ord}(\mu_u^{\odot})$ and $\text{ord}(\mu_v^{\odot})$. We call these variables *PR-ordering variables*.

For a common vertex v occurring as v^{\odot} and v^{\otimes} in G^{\odot} and G^{\otimes} , respectively, we can ensure a consistent edge ordering by adding a so called *common embedding tree* $T(v)$ as child of the embedding trees $T(v^{\odot})$ and $T(v^{\otimes})$ in the Q-embedding representations of G^{\odot} and G^{\otimes} ; see Fig. 2. We get the following lemma.

Lemma 3. *Let G^{\odot} and G^{\otimes} be two biconnected graphs with common P-node degree 3. Requiring the common edges to be ordered consistently is equivalent to satisfying a system of linear equations M_{ord} over \mathbb{F}_2 with the following properties.*

- (i) *All equations in M_{ord} are of the type $x + y = c$ for $c \in \mathbb{F}_2$.*
- (ii) *M_{ord} contains all PR-ordering variables.*
- (iii) *M_{ord} has linear size and can be computed in linear time.*

In the following, we extend this result to the case where we allow exclusive cutvertices. Let $B_1^{\odot}, \dots, B_k^{\odot}$ be the blocks of G^{\odot} and let $B_1^{\otimes}, \dots, B_\ell^{\otimes}$ be the blocks of G^{\otimes} . We say that embeddings of these blocks have *blockwise consistent edge orderings* if for every pair of blocks B_i^{\odot} and B_j^{\otimes} sharing a vertex v the edges incident to v they share are ordered consistently. To have consistent edge orderings, it is obviously necessary to have blockwise consistent edge orderings.

When composing the embeddings of two blocks that share a cutvertex, the edges of each of the two blocks have to appear consecutively (note that this is no longer true for three or more blocks), which leads to another necessary condition. Let v be an exclusive cutvertex of G^\circledast . Then v is contained in a single block of G^\circledast whose embedding induces an order O^\circledast on all common edges incident to v . For every pair of blocks B_i^\circledast and B_j^\circledast containing v , the edges in B_i^\circledast must appear consecutively in the order of the edges incident to v in B_i^\circledast and B_j^\circledast that is induced by O^\circledast . If this is true for every exclusive cutvertex, we say that the embeddings have *pairwise consecutive blocks*.

Lemma 4. *Two graphs without simultaneous cutvertices admit embeddings with consistent edge orderings if and only if their blocks admit embeddings that have blockwise consistent edge orderings and pairwise consecutive blocks.*

To extend Lemma 3 to the case where we allow exclusive cutvertices, we enforce blockwise consistent edge orderings and pairwise consecutive blocks by adding additional PQ-trees to the above instance of SIMULTANEOUS PQ-ORDERING. As before, we get direct access to the embedding chosen for each block, via the PR-ordering variables. We want to get access to the ordering of common edges around a cutvertex v of G^\circledast in a similar way. Let B^\circledast be a block that contains the common edge e incident to v and let e_1 and e_2 be two common edges incident to v that are contained in a different block. We use the *cutvertex-ordering variable* $\text{ord}(e_1, e_2, B^\circledast)$ to denote the order of e_1 , e_2 , and e . Note that this is independent from the choice of the edge e of B^\circledast . To decrease the number of variables, we only consider those variables that are *required* by a cycle basis \mathcal{C} , where $\text{ord}(e_1, e_2, B^\circledast)$ is required by \mathcal{C} if e_1 and e_2 share a cycle in \mathcal{C} .

Lemma 5. *Given two graphs without union or simultaneous cutvertices with common P-node degree 3, requiring the common edges to be ordered consistently is equivalent to satisfying a system of linear equations M_{ord} with the following properties.*

- (i) *All equations in M_{ord} are of the type $x + y = 0$ or $x + y = 1$.*
- (ii) *M_{ord} contains all PR-ordering variables and all cutvertex-ordering variables required by a cycle basis of the common graph.*
- (iii) *M_{ord} has size $O(\min\{n^2, n\Delta^2\})$ (where Δ is the maximum degree in the common graph) and can be computed in linear time in its size.*

4.3 Consistent Relative Positions

In this section, we present a system of linear equations M_{pos} containing the PR-ordering and cutvertex-ordering variables such that satisfying M_{pos} is equivalent to requiring consistent relative positions for an instance of SEFE. Let H and H' be two connected components of the common graph G . To represent the relative position $\text{pos}_{H'}(H)$ of H with respect to H' , we use the relative positions $\text{pos}_C(H)$ of H with respect to cycles C in the cycle basis of H' (Theorem 4). To get binary variables, we use $\text{pos}_C(H) = 0$ if H lies to the right of C and $\text{pos}_C(H) = 1$ if H lies to the left of C . When we consider the graph G^\circledast containing G , it is known that the value of $\text{pos}_C(H)$ is determined by a single, very local embedding decision of G^\circledast [3]. In the following we consider the three possible cases that $\text{pos}_C(H)$ is determined by an R-node, by a P-node or by a cutvertex.

R-Node. If $\text{pos}_C(H)$ is determined by an R-node μ , then C induces a cycle κ in $\text{skel}(\mu)$ and parts of H are contained in a virtual edge ε not contained in κ . The relative position of H with respect to C is the same as the position of ε with respect to κ [3]. As the value of $\text{pos}_\kappa(\varepsilon)$ changes, when the embedding of $\text{skel}(\mu)$ changes, we can simply set $\text{pos}_C(H) + \text{ord}(\mu) = c$ (where $c \in \mathbb{F}_2$ depends on the reference embedding of $\text{skel}(\mu)$ and the orientation of C). Note that this implicitly ensures the consistency of all relative positions that are determined by the embedding of $\text{skel}(\mu)$.

P-Node. If $\text{pos}_C(H)$ is determined by the embedding of $\text{skel}(\mu)$ of a P-node μ , then C induces a cycle κ (of length 2) in $\text{skel}(\mu)$ and H is completely contained in a single edge ε of $\text{skel}(\mu)$ not belonging to κ . Again $\text{pos}_C(H)$ in G is the same as $\text{pos}_\kappa(\varepsilon)$ in $\text{skel}(\mu)$. However, this time the embedding choices of $\text{skel}(\mu)$ are more complicated than to flip or not to flip. Thus, we have to consider all relative positions decided by the embedding of $\text{skel}(\mu)$ at once, to get the dependencies between them.

We only consider the case where the common graph induces paths between the vertices of $\text{skel}(\mu)$ in the expansion graphs of three edges $\varepsilon_1, \varepsilon_2$, and ε_3 (all other cases are simpler as μ has common P-node degree 3). Cycles in the cycle basis \mathcal{C} can induce three different cycles, namely $\kappa_{1,2}, \kappa_{2,3}$, and $\kappa_{1,3}$ consisting of the virtual edges $(\varepsilon_1, \varepsilon_2), (\varepsilon_2, \varepsilon_3)$, and $(\varepsilon_1, \varepsilon_3)$, respectively. For every virtual edge $\varepsilon \neq \varepsilon_i$, we get the three variables $\text{pos}_{\kappa_{1,2}}(\varepsilon), \text{pos}_{\kappa_{2,3}}(\varepsilon)$, and $\text{pos}_{\kappa_{1,3}}(\varepsilon)$ determining the position of ε with respect to these three cycles. Recall that the variable $\text{ord}(\mu)$ determines the ordering of the three edges $\varepsilon_1, \varepsilon_2$, and ε_3 . Consider the case that $\text{ord}(\mu) = 0$ and assume that the reference order of $\varepsilon_1, \varepsilon_2$, and ε_3 as well as the orientation of the cycles $\kappa_{1,2}, \kappa_{2,3}$, and $\kappa_{1,3}$ is as shown in Fig. 3. Then either all three relative positions have the value 0 (which corresponds to RIGHT), or exactly two relative positions have the value 1. Thus, a combination of values for the positions $\text{pos}_{\kappa_{1,2}}(\varepsilon), \text{pos}_{\kappa_{2,3}}(\varepsilon)$, and $\text{pos}_{\kappa_{1,3}}(\varepsilon)$ is possible if and only if there is an even number of 1s. When setting $\text{ord}(\mu) = 1$, there need to be an odd number of 1s. This yields the constraint $\text{ord}(\mu) + \text{pos}_{\kappa_{1,2}}(\varepsilon) + \text{pos}_{\kappa_{2,3}}(\varepsilon) + \text{pos}_{\kappa_{1,3}}(\varepsilon) = 0$ (a different reference embedding or different orientations of the cycles may lead to a 1 on the right-hand side of the equation). As for R-nodes we set $\text{pos}_C(H) + \text{pos}_{\kappa_{i,j}}(\varepsilon) = c$ ($c \in \mathbb{F}_2$), if C induces $\kappa_{i,j}$ in $\text{skel}(\mu)$ and H is contained in the expansion graph of ε .

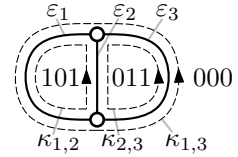


Fig. 3. A P-node with the three cycles $\kappa_{1,2}, \kappa_{2,3}$, and $\kappa_{1,3}$ (dashed)

Cutvertex. The relative position $\text{pos}_C(H)$ is determined by a cutvertex v of G^\circledast if C contains v and H lies in a split component S (with respect to v) different from the split component containing C . We can either embed the whole split component S to the left or to the right of C . For this decision, we introduce the variable $\text{pos}_C(S)$. Clearly, we get $\text{pos}_C(H) = \text{pos}_C(S)$ for every connected component H of G in S . Moreover, there are no further constraints on the relative positions determined by the embedding at the cutvertex v [4]. In case the split component S contains a common edge incident to v , fixing $\text{pos}_C(S)$ is equivalent fixing the cutvertex-ordering variable $\text{ord}(e_1, e_2, B)$, where e_1 and e_2 are the edges in C incident to v and B is the block in S containing v . Thus, we get $\text{ord}(e_1, e_2, B) + \text{pos}_C(S) = c$ ($c \in \mathbb{F}_2$). Together with the constraints for the relative positions determined by the P- and R-nodes, we get the following lemma.

Lemma 6. *Let G^{\circledast} and G^{\circledast} be two graphs without union or simultaneous cutvertices with common P -node degree 3. Requiring the relative positions to be consistent is equivalent to satisfying a system of linear equations M_{pos} with the following properties.*

- (i) *All equations in M_{pos} are of the type $x + y = c$ (with $c \in \mathbb{F}_2$) except for a linear number of equations of size 4.*
- (ii) *M_{pos} contains all PR -ordering variables and all cutvertex-ordering variables required by a cycle basis of G .*
- (iii) *M_{pos} has quadratic size and can be computed in quadratic time.*

Lemma 5 and Lemma 6 yield the following theorem. We obtain the quadratic running time by first eliminating all equations of size 1, and then solving the remaining system of $O(n)$ linear equations of size 4 with the algorithm by Wiedemann [12].

Theorem 5. *SEFE can be solved in quadratic time for two graphs without union or simultaneous cutvertex with common P -node degree 3.*

References

1. Angelini, P., Di Battista, G., Frati, F., Patrignani, M., Rutter, I.: Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph. *Journal of Discrete Algorithms* 14, 150–172 (2012)
2. Bläsius, T., Kobourov, S.G., Rutter, I.: Simultaneous Embedding of Planar Graphs. In: *Handbook of Graph Drawing and Visualization*, pp. 349–381. Chapman and Hall/CRC (2013)
3. Bläsius, T., Rutter, I.: Disconnectivity and relative positions in simultaneous embeddings. In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 31–42. Springer, Heidelberg (2013)
4. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. In: *Proc. 24th ACM-SIAM Sympos. Discrete Algorithm, SODA 2013*, pp. 1030–1043. ACM (2013)
5. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences* 13, 335–379 (1976)
6. Di Battista, G., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. *Algorithmica* 15(4), 302–318 (1996)
7. 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)
8. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) *GD 2000*. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
9. Haeupler, B., Jampani, K., Lubiw, A.: Testing simultaneous planarity when the common graph is 2-connected. *J. Graph Algorithms Appl.* 17(3), 147–171 (2013)
10. Jünger, M., Schulz, M.: Intersection graphs in simultaneous embedding with fixed edges. *Journal of Graph Algorithms and Applications* 13(2), 205–218 (2009)
11. Schaefer, M.: Toward a theory of planarity: Hanani-tutte and planarity variants. In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 162–173. Springer, Heidelberg (2013)
12. Wiedemann, D.: Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory* 32(1), 54–62 (1986)

Sketched Graph Drawing: A Lesson in Empirical Studies

Helen C. Purchase

School of Computing Science, University of Glasgow, UK

Abstract. This paper reports on a series of three similar graph drawing empirical studies, and describes the results of investigating subtle variations on the experimental method. Its purpose is two-fold: to report the results of the experiments, as well as to illustrate how easy it is to inadvertently make conclusions that may not stand up to scrutiny. While the results of the initial experiment were validated, instances of speculative conclusions and inherent bias were identified. This research highlights the importance of stating the limitations of any experiment, being clear about conclusions that are speculative, and not assuming that (even minor) experimental decisions will not affect the results.

Keywords: graph sketching, empirical studies, replication, limitations.

1 Introduction

This paper reports on a series of three similar experiments with a common aim: to determine the graph drawing layout principles favored when participants draw graphs. The first experiment of the series was peer-reviewed and published in a reputable journal [1].

Typically in information visualization or HCI research, researchers run an experiment, form conclusions, publish, and then move on to the next interesting question. The second and third experiments reported here arose from a reflective critique of the first experiment. In this case, the experimenter (the author of this paper) did not “publish and move on”, but explored subtle aspects of the experimental design, attempting to replicate and confirm the initial results.

The focus of this paper is therefore two-fold: to present the experiments and their results, but also to describe a process whereby revisiting empirical work has highlighted interesting facts about the process of conducting empirical studies.

This motivation for this paper relates to the idea of reflective critique, an idea borrowed from the practice of action learning projects. Action learning projects [2] do not aim to address specified research questions, and their approach is formative rather than summative, resulting in a cycle of continuous improvement. This approach is unlike the typical experimental research project, where a research question is clearly defined, a methodology is designed to address it, data is collected, and final conclusions made and published. In contrast, an action learning methodology encourages honest reflection on outcomes, and these reflections are fed into another cycle of the process.

This paper, therefore, reports on the results of two experiments conducted ‘after the fact’, i.e. after conclusions from the first experiment had been disseminated, with the latter two experiments being designed as a result of reflection and critique.

2 Background

The design of automatic graph layout algorithms tend to be based on common ‘aesthetic principles’, for example, the elimination of edge crossings or the minimisation of adjacent edge angles. Early experimental research investigating how graphs may best be laid out tended to use a task-based performance approach (e.g. [3,4,5]), and established key findings such as the fact that a high occurrence of crossed edges reduced performance and prominent depiction of symmetry increased performance.

More recent empirical research has instead focussed on the manner in which participants create their own visual layout of relational information as a graph drawing. Van Ham and Rogowitz [6] (HR08) asked participants to adjust manually the layout of existing graph drawings. They used four graphs of 16 nodes, each with two clusters separated by one, two, three and four edges respectively. The graphs were presented in a circular and a spring layout [7], giving a total of eight starting diagrams, shown in random order. They collected 73 unique drawings, and found that most participants separated the two clusters, that the human drawings contained 60% fewer edge crossings than the automatically produced drawings, and that humans did not value uniform edge length as much as the spring algorithm did.

Dwyer et. al [8] (D+09) performed a similar hands-on experiment, asking participants to lay out two social networks, each with a circular initial arrangement. Participants were encouraged to lay the graphs out in a way that would best support the identification of cliques, chains, cut nodes and leaf nodes. With a focus on the process of layout rather than on the product, the only observation that they make about the graphs produced is that users removed edge crossings.

The first experiment in the series reported here (**Experiment 1**, [1]) was designed to address a similar research goal as HR08 and D+09, using a different methodology. The research question is *Which graph drawing layout principles do people favour when creating their own drawings of graphs?*

There are five main design features of Experiment 1 that differentiate it from HR08 and D+09, the *differentiating design features* (DDFs):

- DDF-1.** The participants had to both draw the graph, as well as lay it out, a more complex task than both HR08 and D+09;
- DDF-2.** The participants drew the graphs from scratch, so were not biased by any initial layout (both HR08 and D+09 used an initial configuration);
- DDF-3.** A sketching tool was used, so the physical drawing process was unhindered by an intermediate editing interface;
- DDF-4.** Video data was collected, so both the process and product of creation were able to be analysed (this was done by D+09, but not HR08);
- DDF-5.** Layout preferences were discussed with the participants in a post-experiment interview (this was done by D+09, but not HR08).

Four graphs were used, two practise graphs and two experimental graphs. Data on product, process and preferences were collected¹.

¹ Note that the publication arising from Experiment 1 also included graphs drawn in a point-and-click mode, but only the sketched graph experiment is considered here.

The several conclusions of **Experiment 1** as published [1] included:

CONC-1 The layout principles that participants favoured during the process of laying out their drawings were often not evident in the final product.

CONC-2 The principle of fixing nodes and edges to an underlying unit grid was prominent.

2.1 Reflective Critique: Issues Arising

After peer-review, publication, presentation, and independent citation of this first experiment and its results, audience members at two seminars suggested some subtle variations on the experimental method: not new research questions, simply small amendments. As is typical in such situations, this author responded that such variations could be addressed as part of ‘future work’.

It was not necessary to investigate these issues (the paper had already been published after all), but they led the experimenter to reflect on the research, and to question to what extent results might be biased by a method.

Three issues arose as a result of this reflection:

- If participants compromise their layout design during creation of the drawing (CONC-1), does this mean that they are not happy with their final product? **Experiment 2** addressed the issue of whether participants were satisfied with their final drawing, or whether they expressed disappointment that they were unable to conform to their desired layout.
- If participants favour a grid-based layout (CONC-2), would they prefer a drawing laid out using an algorithm that aligns nodes and edges to an underlying grid to their own? **Experiment 2** investigated whether the participants preferred their own sketched graph drawing to a similar one that conformed to a grid layout.
- Was the tendency to favour a grid layout and straight lines (CONC-2) a consequence of the way in which the graph information stimuli were presented as a list of edges? **Experiment 3** investigated whether representing the graph structure in an alternative text format also produces results that favour grid layout.

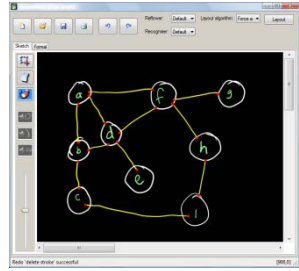
3 The Experiments

3.1 Experimental Process for all Three Experiments

The primary research question in all three of these graph sketching experiments is: *Which graph drawing layout principles do people favour when creating their own drawings of graphs?* Participants were asked to draw graphs and their drawings were analysed for evidence of common graph drawing layout principles.

Equipment: A graph-drawing sketch tool, SketchNode [9] was used, allowing nodes, edges and node labels to be sketched, edited and moved with a stylus on a tablet screen, laid flat.

Task: Participants were given a textual description of a graph and asked to draw it in SketchNode, with the instruction to *Please draw this graph as best as you can so to make it “easy to understand”*. They were deliberately not given any further instruction as to what “easy to understand” means, nor primed with any information about common graph layout principles (e.g. minimising edge crossings, use of straight edges, etc.). They had as long as they liked to draw and adjust the layout of the graphs.



Graphs: Two experimental graphs were used in all three experiments: graph A (10 nodes, 14 edges) and graph B (10 nodes, 18 edges).

Experimental method: Participants completed required ethical procedures and provided demographic information. All the relevant features of SketchNode were demonstrated, and participants were given the opportunity to practice and ask questions. Two practice graphs ensured participants were comfortable with the task and system.

The two experimental graphs A and B, were then presented to the participants, with the graph edges presented in different random order for each participant, and counterbalanced between participants. At the end of the experiment, the participants were asked “Why did you arrange the graphs in the way you did?” in a recorded interview.

The participants: Participants in all three experiments were of a similar profile: a mixture of students and non-students, of both sexes, with around half the student participants in each experiment studying some form of computer science.

3.2 Differences between the Experiments

In **Experiment 1**, the most important differentiating feature in comparison with prior research was the way the graphs were presented (DDF-2). HR08 and D+09 presented their graphs as graph drawings which already had some layout properties (circular and spring in HR08 and circular in D+09). So as not to bias the participants towards any layout principles, the graphs in Experiment 1 were presented in textual form, as a list of edges (Table 1, column 1).

Experiments 2 and 3 addressed four issues arising from Experiment 1.

Compromised layout: CONC-1 suggests participants may not have been entirely happy with their final drawing, as they had been obliged to compromise their favoured layout as the graph became more complex. **Experiment 2** investigated the extent of this compromise, and whether participants acknowledged it.

The first research question for this experiment was “Do participants like the layout of their final product?” We speculated they would express dissatisfaction with their final product. Once they had drawn their graph, we asked them to indicate on a scale of 1–5 how “happy” they were with their drawing.

Preference for a grid: CONC-2: Experiment 1 suggested a grid layout was favoured; we anticipated participants would prefer a grid layout to their own.

The second research question was “*Do participants prefer their sketched drawing to be laid out in a grid format to their own layout?*”

Experiment 2 used the two automatic graph layout algorithms in SketchNode: spring (based on Fruchterman and Reingold [10]) and grid (placing nodes and edges on the lines and vertices of an underlying unit grid). After layout, the visual sketched appearance of the nodes and labels remains the same, so the resultant diagram can be directly compared with original sketched drawing.

At the end of the sketching stage of Experiment 2, the participant’s own drawing was laid out using these two algorithms. Participants were asked to rank the three drawings according to their preference.

In an attempt to eliminate any personal bias or recency effects, a willing subset of the participants chose between hand-drawn and the two automatically laid out drawings two weeks after the experiment.

Validation: As both changes to the experimental method for Experiment 2 were post-experiment activities, Experiment 2 also served as a means to validate the results of Experiment 1.

Effect of graph format: One of the main differentiating features between Experiment 1 and prior research was that the graph information was presented in abstract form, rather than as a graph drawing (DDF-2), and the participants drew the graphs from scratch. Here we investigated whether even this abstract form had produced a bias.

The research question for this experiment was “*Does the format in which the graph structure is represented affect the layout of the sketched graph drawings produced?*” For **Experiment 3**, we presented the graphs as an adjacency list (Table 1, column 3), and followed exactly the same process as Experiment 1. This format is visually quite different from the simple list of edges, as each edge is not clearly specified as an individual pair, and it is more obvious which nodes have a higher degree. We wished to investigate whether a format that does not focus on the individual node pairs (as in Experiment 1) would still result in user-sketched drawings that conform to a grid structure.

Table 1 shows the differences between the experiments, as well as those factors that remained the same. Figure 1 shows example sketches from all three experiments.

4 Data Analysis

Compromised layout (Experiment 2): The participants in Experiment 2 indicated how happy they were with their own sketches (five-point scale, 5=perfectly happy). Graph A’s mean: 4.14, graph Bs’ mean: 3.5; both graphs together: 3.91.

Participants were asked what they didn’t like about their drawings, and how they would improve them. None mentioned that they would have liked to conform to a grid layout; most comments related to local issues like the size and shape of the nodes, and connections between the nodes and edges. The few comments that referred to overall layout of the drawing were concerned with spreading the nodes out, symmetry and circular layout. There were also several comments about the need to plan in advance.

Table 1. Summary of the differences between the three experiments

	Experiment 1	Experiment 2	Experiment 3
experimental graphs	A ($n = 10, e = 14$) B ($n = 10, e = 18$)	as Experiment 1	as Experiment 1
experimental task	After introduction and training activities, participants sketched the two graphs	as Experiment 1	as Experiment 1
equipment	SketchNode	as Experiment 1	as Experiment 1
number of participants	17	22	26
form of graph presented to participants	(A, D) (A, C) (B, D) (C, D) (B, C) (B, E) (C, E) (E, J) (F, G) (J, F) (F, I) (G, I) (J, H) (I, H)	as Experiment 1	S U R Q S V W Z Y X V R U W Y U Z T R Q S X Z
post-experiment discussion	none	participants indicated how happy they were with their drawing	as Experiment 1
post-experiment ranking	none	participants ranked sketched drawings against associated spring and grid drawings	as Experiment 1

Preference for a grid (Experiment 2): Once participants had sketched their graph, the two graph layout algorithms (Section 3.2 above) were applied to their drawing.

A three-way-set (TWS) is a set of three drawings: a participant’s sketched drawing, and two versions of this sketch produced by the algorithms. Each participant has a TWS-GA and TWS-GB. Figure 2 shows a TWS for one of the participant’s graphs.

Participants ranked the drawings in their own TWS-GA and TWS-GB at the end of Experiment 2. After two weeks, we contacted all participants for a follow-up ranking experiment; fourteen took part. They ranked their own TWS-GA and TWS-GB (as before), as well as the TWS-GA and TWS-GB for two other participants. They were not told their own drawings were included in these sets (Table 2).

The data were analysed with Friedman tests with adjusted pairwise comparisons. The only significant results related to graph B (highlighted in Table 2):

² The notational convention is: graph A drawn by participant 3 is 3A; graph B drawn by participant number 8 is 8B. The suffix -1, -2 or -3 indicates experiment 1, 2 or 3.

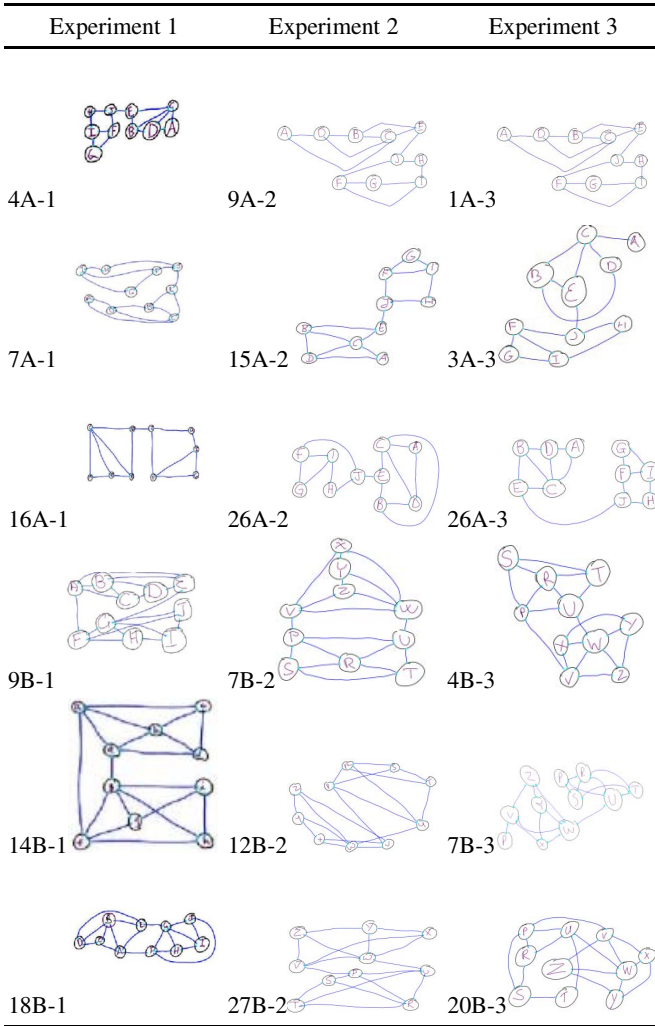


Fig. 1. Example drawings from all three experiments²

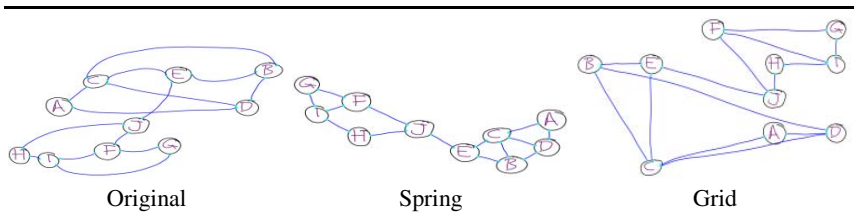


Fig. 2. The TWS-GA for participant 17. Note that the hand-drawn nature and form of the nodes and their labels is retained.

Table 2. Mean ranking of TWSs (3=most preferred). Significant results in italics.

	number of participants : TWSs	mean rank: sketch	mean rank: spring	mean rank: grid	% times sketch preferred	% times grid preferred
After drawing the graph (own TWS)	22:22	GA 2.45	1.77	1.77	50%	23%
		GB <i>2.54</i>	<i>1.41</i>	2.05	68%	23%
Two weeks later (own TWS)	14:14	GA 2.07	1.86	2.07	36%	29%
		GB 1.79	1.79	2.43	29%	43%
Two weeks later (own + two other TWSs)	14:42	GA 2.00	1.79	2.21	36%	36%
		GB <i>1.90</i>	<i>1.60</i>	<i>2.50</i>	26%	55%

- immediately after the experiment, the participants’ ranking for their own sketched graph was higher than that for the spring layout ($p < 0.001$);
- two weeks after the experiment, the participants’ ranking for the grid layout was higher than the spring layout ($p < 0.001$), and higher than their own drawing ($p = 0.019$) in the set of three sketched graphs that included their own.

Validation (Experiment 2): Experiment 1 found that participants appeared to favour a grid layout and horizontal and vertical edges in their sketched graph drawings. We analysed the 44 graph drawings from Experiment 2 for the following key layout features (Table 3):

- *Number of edge crossings:* points outside of the node boundaries where one or more edges cross.
- *Number of straight lines.* A visual assessment as to whether an edge was intended to be straight was agreed by two independent coders.
- *Number of vertical or horizontal edges, and grid structure.* A visual assessment was agreed by two independent coders as to whether edges were intended to be horizontal or vertical, and whether drawings had been drawn with a grid in mind.

Independent samples two-tailed t-tests were conducted using the 34 drawings from Experiment 1 and the 44 drawings from Experiment 2 (Table 3).

Effect of graph format (Experiment 3): We analysed the 52 graph drawings from Experiment 3 using the same layout features as for Experiments 1 and 2.

Independent samples two-tailed t-tests were conducted using the 34 drawings from Experiment 1 and the 52 drawings from Experiment 3 were used (Table 3).

In the interviews, as before, no participants spoke directly of a grid layout; there were comments about local features (size of the nodes, the need for straight lines), and layout (crossings, symmetry, distance between nodes). When asked why they drew the graph as they did, many said the adjacency list itself suggested the structure of the drawing: they ‘worked from top to bottom.’

Table 3. Comparing key layout features in the graphs produced in the experiments.

	E1 <i>n</i> = 34	E2 <i>n</i> = 44	E3 <i>n</i> = 52	validation:			
				E1 v. E2		effect of format: E1 vs E3	
				<i>t</i>	<i>p</i>	<i>t</i>	<i>p</i>
Number of crosses/drawing	2.21	4.05	2.38	1.410	0.163	0.216	0.829
Number of crosses/drawing (excl. outliers)	1.24	2.05	1.62	1.400	0.166	0.776	0.440
% HV edges	33.6%	28.0%	23.3%	1.232	0.222	2.842	<i>0.006^a</i>
% straight edges	88.1%	85.8%	90.3%	0.472	0.638	0.541	0.590
Drawn with grid in mind	10(29%)	7(16%)	4(8%)	1.433	0.156	2.478	<i>0.007^b</i>

^a More HV edges in Experiment 1. ^b More grids in Experiment 1

5 Results

Compromised layout: We wished to see whether participants expressed any dissatisfaction with their drawings as a result of having to compromise the layout while creating the drawing: a stated conclusion of Experiment 1.

In general, the participants were satisfied with the layout of their own drawings: their satisfaction ratings were high; they expressed little dissatisfaction, and did not mention that they would have liked to have been able to conform to a grid.

This is an instance of *stating a conclusion based on insufficient data*. In Experiment 1, observation of the videos (by two independent coders) suggested that participants conformed to a grid in the initial stages of drawing, but that this layout feature was abandoned later in the drawing process. This conclusion was, however, simply suggestive, and there was no qualitative interview data to back it up. It appears that, even if layout compromises had been made, participants were not aware of having done so.

Preference for a grid: We wished to see whether participants would prefer a grid-based layout to their own layout.

While initially they preferred their own drawings in Experiment 2, when the recency effect of having just drawn the graph was eliminated, the grid layout was ranked as substantially better than the sketches for the more complex graph. However, none of the participants mentioned a grid formation or horizontal or vertical edges in their comments.

It appears that participants know what they like when they see it (and when it is not in competition with a drawing that they know is their own), but cannot independently articulate the layout features that contribute to what they like.

This is an instance of *stating an incorrect conclusion based on qualitative data*. Despite the Experiment 1 drawings being analyzed by two independent coders who formed the same conclusion (that participants preferred a grid), this inferred conclusion does not hold when more direct data is collected.

Validation: The fact that there are no significant differences in the values for the key metrics for Experiment 2 suggests validation of the results of Experiment 1: when the

graph is presented as a list of edges, participants tend to favour a grid layout and straight lines.

This is an instance of *validating data*. Using identical methodologies for experiments 1 and 2, we would expect similar values for the dependent data variables.

Effect of graph format: We wished to see whether the form in which the graph stimuli are presented would affect the results.

The drawings produced in Experiment 3 from adjacency list stimuli did not favour a grid layout; there is a significant difference on the key metrics of horizontal and vertical edges, and grid formation, between Experiments 1 and 3.

The format in which the relational graph information is presented to the participants thus affects the final layout of their drawing, a result echoing experimental results on visual metaphors [11]. The participants spoke of ‘following the table from top to bottom’ in Experiment 3; it is likely that the participants in Experiment 1 followed the edge list from left to right.

There is an irony here: Experiment 1 presented the graph as a list of edges so to address possible layout biases in HR08 and D+09, who presented graphs as drawings. It seems even using a simple list of edges can introduce a bias.

This is an instance of *unintended bias*. Even a simple (and seemingly innocuous) decision in Experiment 1 introduced a factor that biased the results.

Table 4. Summary and comparison of the findings for the three experiments

	Experiment 1	Experiment 2	Experiment 3
compromised layout	assumed, from video data	<i>not proven</i>	not investigated
preferring a grid over own drawing	assumed, from sketch graph drawing data	<i>some support after elimination of recency effects</i>	not investigated
validation	not applicable	<i>validation of key graph drawing metrics</i>	not appropriate
effect of graph format	not considered	not investigated	<i>effect found</i>

6 Discussion

6.1 Implications for this Research

Our speculation that participants prefer the result of a grid-based algorithm over their own drawing was partially confirmed, but only when the ‘personal pride’ factor had diminished over time, and for the more complex graph. It is still clear, however, that while participants might prefer a grid layout in both creation and recognition, they are less able to articulate this preference.

The most surprising result is that how the graph is presented to the participants has a significant effect on the form of their drawing — this was something that had not been

considered originally. It suggests that the only way bias may be eliminated entirely would be by asking the participant to draw a graph based on their internal cognitive structure, and not on an externally visible form. This may mean describing a scenario to the participant while attempting to avoid verbal bias (for example, a social network) and then asking the participant to draw the graph representing the relational information.

Of course, this story could not simply end here, as there are several outstanding issues to address about all three of these experiments. Do these results extend to larger graphs? What would happen if the participants were all novice computer users? Or if a digital whiteboard were used? Or if participants were told that the graphs related to a domain (e.g. a transport network or a circuit diagram)? Or if they were explicitly advised to plan in advance? Further experiments would no doubt shed more light on these initial studies (and would probably reveal some unexpected results).

6.2 Implications for Experimental Research

The results of our investigation of subtle experimental variations suggest that:

- There will always be a bias relating to the manner in which information is presented to an experimental participant;
- Even if the results of an experiment are validated by repeating it, these results may still be compromised by bias;
- Firm experimental conclusions need validation through replication and multiple sources of data.

We did not set out to investigate the effect of experimental subtleties: our original aim was not to run a series of comparative experiments. If it had been, then we might have conducted a broader experiments-within-an-experiment study, preferably using the same participants throughout, and followed a systematic process of enquiry. No: we initially set out to investigate what happens when participants draw graphs from scratch — and we published a peer-reviewed paper in a reputable journal presenting the results of this study, as is common practice: researchers run an experiment, collect data, form conclusions, and publish. And then typically move on to their next experiment.

The contribution of this paper is therefore broader than the simple ‘run an experiment and report’ model: by reflecting on and critiquing our own experimental work, and investigating issues arising from the critique, we have demonstrated the limitations of this common practice.

It is rare that researchers repeat an experiment with subtle variations — doing so has revealed that there is still much to learn about the nature of user-sketched graphs, that even a carefully-conducted experiment may have flaws, that there is value in not simply moving on to the next ‘big’ question, and that repeating an experiment so as to investigate subtleties may produce surprising results. All experiments have limitations — no experiment can ever be perfect. This paper demonstrates the importance of acknowledging these limitations, of validating results where possible, and of reading published experimental results with healthy critical attitude.

Acknowledgments. SketchNode is the result of significant efforts of Beryl Plimmer and Hong Yul, of the University of Auckland. The experiments were conducted by

Christopher Pilcher, Rosemary Baker, Anastasia Giachanou, and Gareth Renaud. John Hamer assisted with data analysis, coding and editing. Ethical approval was given by the Universities of Auckland and Glasgow.

References

1. Purchase, H.C., Pilcher, C., Plimmer, B.: Graph drawing aesthetics — created by users not algorithms. *IEEE Trans. Vis. and Computer Graphics* 18(1), 81–92 (2012)
2. Kemmis, S., McTaggart, R.: *The Action Research Planner*, 3rd edn. Deakin University Press (1988)
3. Huang, W.: Using eye-tracking to investigate graph layout effects. In: Hong, S.H., Ma, K.L. (eds.) *Proc. Asia Pacific Symp. on Visualisation*, pp. 97–100. IEEE (2007)
4. Ware, C., et al.: Cognitive measurements of graph aesthetics. *Information Visualization* 1(2), 103–110 (2002)
5. Purchase, H.C., Cohen, R.F., James, M.: An experimental study of the basis for graph drawing algorithms. *ACM J. Experimental Algorithmics* 2(4), 1–17 (1997)
6. van Ham, F., Rogowitz, B.E.: Perceptual organisation in user-generated graph layouts. *IEEE Trans. Vis. and Computer Graphics* 14(6), 1333–1339 (2008)
7. Gansner, E.R., Koren, Y., North, S.C.: Graph Drawing by Stress Majorization. In: Pach, J. (ed.) *GD 2004. LNCS*, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
8. Dwyer, T., et al.: A comparison of user-generated and automatic graph layouts. *IEEE Trans. on Vis. and Computer Graphics* 15(6), 961–968 (2009)
9. Plimmer, B., Purchase, H.C., Yang, H.Y.: SketchNode: Intelligent sketching support and formal diagramming. In: Brereton, M., Viller, S., Kraal, B. (eds.) *OzChi Conference*, Brisbane, Australia, pp. 136–143 (2010)
10. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. *Software — Practice and Experience* 21(11), 1129–1164 (1991)
11. Ziemkiewicz, C., Kosara, R.: The shaping of information by visual metaphors. *IEEE Trans. on Vis. and Computer Graphics* 14(6), 1269–1276 (2008)

Many-to-One Boundary Labeling with Backbones

Michael A. Bekos¹, Sabine Cornelsen², Martin Fink³,
Seok-Hee Hong⁴, Michael Kaufmann¹, Martin Nöllenburg⁵,
Ignaz Rutter⁵, and Antonios Symvonis⁶

¹ Institute for Informatics, University of Tübingen, Germany
{bekos,mk}@informatik.uni-tuebingen.de

² Department of Computer and Information Science, University of Konstanz
sabine.cornelsen@uni-konstanz.de

³ Lehrstuhl für Informatik I, Universität Würzburg, Germany
martin.a.fink@uni-wuerzburg.de

⁴ School of Information Technologies, University of Sydney
shhong@it.usyd.edu.au

⁵ Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
{noellenburg,rutter}@kit.edu

⁶ School of Applied Mathematics and Physical Sciences, NTUA, Greece
symvonis@math.ntua.gr

Abstract. In this paper we study *many-to-one boundary labeling with backbone leaders*. In this model, a horizontal backbone reaches out of each label into the feature-enclosing rectangle. Feature points associated with this label are linked via vertical line segments to the backbone. We present algorithms for label number and leader-length minimization. If crossings are allowed, we aim to minimize their number. This can be achieved efficiently in the case of fixed label order. We show that the corresponding problem in the case of flexible label order is NP-hard.

1 Introduction

Boundary labeling was developed by Bekos et al. [2] as a framework and an algorithmic response to the poor quality (feature occlusion, label overlap) of specific labeling applications. In boundary labeling, labels are placed at the boundary of a rectangle and are connected to their associated features via arcs referred to as *leaders*. Leaders attach to labels at *label ports*. A survey by Kaufmann [4] presents different boundary labeling models that have been studied so far.

In *many-to-one boundary labeling* each label is associated to more than one feature point. This model was formally introduced by Lin et al. [7], who assumed that each label has one port for each connecting feature point (see Fig. 1a) and showed that several crossing minimization problems are NP-complete and, subsequently, developed approximation and heuristic algorithms. In a variant of this model, referred to as *boundary labeling with hyperleaders*, Lin [6] resolved the multiple port issue by joining together all leaders attached to a common label with a vertical line segment in the track-routing area (see Fig. 1b). At the cost of label duplications, leader crossings could be eliminated.

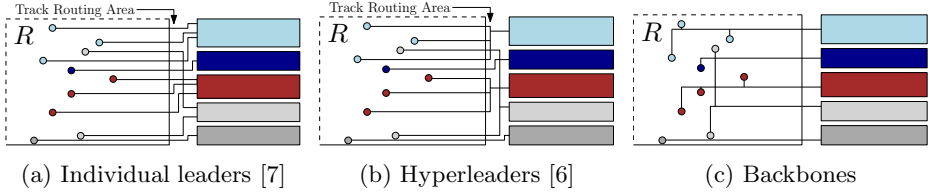


Fig. 1. Different types of many-to-one labelings

We study *many-to-one boundary labeling with backbone leaders* (for short, *backbone labeling*). In this model, a horizontal backbone reaches out of each label into the feature-enclosing rectangle. Feature points that need to be connected to a label are linked via vertical line segments to the label’s backbone (*backbone leaders*; see Fig. 1c). Formally, we are given a set $P = \{p_1, \dots, p_n\}$ of n points in an axis-aligned rectangle R , where each point $p \in P$ is assigned a color $c(p)$ from a color set C . We also assume that the points are in general position and sorted in decreasing order of y-coordinates, with p_1 being the topmost. Our goal is to place colored labels to the left or right side of R and assign each point $p \in P$ to a label $l(p)$ of color $c(p)$. A *backbone labeling* for a set of colored points P is a set \mathcal{L} of colored labels and a mapping of each point $p \in P$ to some $c(p)$ -colored label in \mathcal{L} , so that (i) each point is connected to a label of the same color, and (ii) there are no backbone leader overlaps. A *crossing-free* backbone labeling is one without leader crossings.

The number of labels of a specific color may be unlimited or bounded by $K \geq |C|$. If $K = |C|$, all points of the same color are associated with a common label. One may restrict the maximum number of allowed labels for each color in C separately by specifying a *color vector* $\mathbf{k} = (k_1, \dots, k_{|C|})$. A backbone labeling that satisfies all of the restrictions on the number of labels is called *feasible*.

Our goal is to find feasible backbone labelings that optimize different quality criteria. We study three different quality criteria, *label number minimization* (Section 2), *total leader length minimization* (Section 3), and *crossing minimization* (Section 4). The first two require crossing-free leaders. We consider both *finite backbones* and *infinite backbones*. Finite backbones extend horizontally from the label to the furthest point connected to the backbone, whereas infinite backbones span the whole width of the rectangle (thus one could use duplicate labels on both sides). Our algorithms also vary depending on whether the order of the labels is fixed or flexible and whether more than one label per color class can be used. Note that, due to space constraints some of our proofs are only sketched. Detailed proofs can be found in the technical report [1].

2 Minimizing the Total Number of Labels

In this section we minimize the total number of labels in a crossing-free solution, i.e., we set $K = n$ so that there is effectively no upper bound on the number of

labels. We first consider the case of infinite backbones and present an important observation on the structure of crossing-free labelings.

Lemma 1. *Let p_i, p_{i+1} be two vertically consecutive points. Let p_j ($j < i$) be the first point above p_i with $c(p_j) \neq c(p_i)$, and let $p_{j'}$ ($j' > i + 1$) be the first point below p_{i+1} with $c(p_{j'}) \neq c(p_{i+1})$, if such points exist. In any crossing-free backbone labeling with infinite backbones, p_i and p_{i+1} are vertically separated by at most 2 backbones and any separating backbone has color $c(p_i), c(p_{i+1}), c(p_j)$, or $c(p_{j'})$.*

Sketch of Proof. In a crossing-free solution any infinite backbone splits the drawing into two independent subinstances above and below the backbone. Clearly, a backbone traversing a point has to be of the same color. On the other hand, we can check that a backbone lying between two points p_i and p_{i+1} can only have color $c(p_i), c(p_{i+1})$, or the color of the next point of distinct color above p_i or of the one below p_{i+1} . \square

Clearly, if all points have the same color, one label always suffices. Even in an instance with two colors, one label per color is enough. However, if a third color is involved, then many labels may be required. We sketch how to find an optimum solution in $O(n)$ time. First, we replace any maximal set of identically colored consecutive points by the topmost point in the set. One can show that an optimum solution of the original instance can be easily obtained from an optimum solution of the reduced instance, in which no two consecutive points have the same color. We solve the reduced instance using dynamic programming.

Theorem 1. *Let $P = \{p_1, p_2, \dots, p_n\}$ be an input point set consisting of n points sorted from top to bottom. Then, a crossing-free labeling of P with the minimum number of infinite backbones can be computed in $O(n)$ time.*

Sketch of Proof. We store a table nl of values $nl(i, cur, c_{\text{bak}}, c_{\text{free}})$ representing the minimum number of backbones needed above or at point p_i such that the lowest backbone is c_{bak} -colored, the lowest backbone goes through p_i if the flag $cur = \text{true}$ and lies above p_i otherwise, and the (single) point between p_i and the lowest backbone (if $cur = \text{false}$) has color c_{free} . By careful case analysis, we can see that any entry of the table can be computed in constant time. \square

We now consider finite backbones. First, note that we can slightly shift the backbones in a given solution so that backbones are placed only in gaps between points. We number the gaps from 0 to n where gap 0 is above and gap n is below all points. Suppose a point p_l lies between a backbone of color c in gap g and a backbone of color c' in gap g' with $0 \leq g < l \leq g' \leq n$ such that both backbones horizontally extend to at least the x-coordinate of p_l . Let $R(g, g', l)$ be the rectangle bounded by these two backbones, the vertical line through p_l and the right side of R . Suppose all points except the ones in $R(g, g', l)$ are already labeled. An optimum solution for connecting the points in R cannot reuse any backbone except for the two backbones in gaps g and g' ; hence, it is independent of the rest of the solution. We use this observation for solving the problem by dynamic programming.

Theorem 2. *Given a set P of n colored points and a color set C , we can compute a feasible labeling of P with the minimum number of finite backbones in $O(n^4|C|^2)$ time.*

Sketch of Proof. For $0 \leq g \leq g' \leq n$, $l \in \{g, \dots, g'\} \cup \{\emptyset\}$, and two colors c and c' let $T[g, c, g', c', l]$ be the minimum number of additional labels that are needed for labeling all points in the rectangle $R(g, g', l)$ under the assumption that there is a backbone of color c in gap g , a backbone of color c' in gap g' , between these two backbones there is no backbone placed yet, and they both extend to the left of p_l . Note that for $l = \emptyset$ the rectangle is empty and $T[g, c, g', c', \emptyset] = 0$. Finally, let $\bar{c} \notin C$ be a dummy color, and let $p_{\bar{l}}$ be the leftmost point. Then, the value $T[0, \bar{c}, n, \bar{c}, \bar{l}]$ is the minimum number of labels needed for labeling all points. By careful case analysis, we can compute each of the $(n+1) \times |C| \times (n+1) \times |C| \times (n+1)$ entries of table T in $O(n)$ time. \square

3 Length Minimization

In this section we minimize the total length of all leaders in a crossing-free solution, either including or excluding the horizontal lengths of the backbones. We distinguish between a global bound K on the number of labels or a color vector \mathbf{k} of individual bounds per color. We first consider the case of infinite backbones and use a parameter λ to distinguish the two minimization goals, i.e., we set $\lambda = 0$, if we want to minimize only the sum of the length of all vertical segments and we set λ to be the width of the rectangle R if we also take the length of the backbones into account. We further assume that $p_1 > \dots > p_n$ are the y-coordinates of the input points.

Single Color. If all points have the same color, we seek for a set of at most K y-coordinates where we draw the backbones and connect each point to its nearest one, i.e., we must solve the following problem: Given n points with y-coordinates $p_1 > \dots > p_n$, find a set S of at most K y-coordinates that minimizes

$$\lambda \cdot |S| + \sum_{i=1}^n \min_{y \in S} |y - p_i|. \tag{1}$$

Note that we can optimize the value in Eq. (1) by choosing $S \subseteq \{p_1, \dots, p_n\}$. Hence, the problem can be solved in $O(Kn)$ time if the points are sorted according to their y-coordinates using the algorithm of Hassin and Tamir [3]. Note that the problem corresponds to the K -median problem if $\lambda = 0$.

Multiple Colors. If the input points have different colors, we can no longer assume that all backbones go through one of the given n points. However, by Lemma 1, it suffices to add between any pair of vertically consecutive points two additional candidates for backbone positions, plus one additional candidate

above all points and one below all points. Hence, we have a set of $3n$ candidate lines at y -coordinates

$$p_1^- > p_1 > p_1^+ > p_2^- > p_2 > p_2^+ > \dots > p_n^- > p_n > p_n^+ \tag{2}$$

where for each i the values p_i^- and p_i^+ are as close to p_i as the label heights allow. Clearly, a backbone through p_i can only be connected to points with color $c(p_i)$. If we use a backbone through p_i^- (or p_i^+ , respectively), it will have the same color as the first point below p_i (or above p_i , respectively) that has a different color than p_i . Hence, the colors of all candidates are fixed or the candidate will never be used as a backbone. For an easier notation, we denote the i th point in Eq. (2) by y_i and its color by $c(y_i)$. We solve the problem using dynamic programming.

Theorem 3. *A minimum length backbone labeling with infinite backbones for n points with $|C|$ colors can be computed in $O(n^2 \cdot \prod_{i=1}^{|C|} k_i)$ time if at most k_i labels are allowed for color i , $i = 1, \dots, |C|$ and in $O(n^2 \cdot K)$ time if in total at most K labels are allowed.*

Sketch of Proof. For each $i = 1, \dots, 3n$, and for each vector $\mathbf{k}' = (k'_1, \dots, k'_{|C|})$ with $k'_1 \leq k_1, \dots, k'_{|C|} \leq k_{|C|}$, let $L(i, \mathbf{k}')$ denote the minimum length of a feasible backbone labeling of $p_1, \dots, p_{\lfloor \frac{i+1}{3} \rfloor}$ using k'_c infinite backbones of color c for $c = 1, \dots, |C|$ such that the bottommost backbone is at position y_i , if such a labeling exists. Otherwise $L(i, \mathbf{k}') = \infty$. One can show that the values $L(i, \mathbf{k}')$ can be computed recursively in $\mathcal{O}(n^2 \prod_{i=1}^{|C|} k_i)$ time in total. Let S be the set of candidates y_i such that all points below y_i have the same color as y_i . Then, we can compute the minimum total length of a backbone labeling of p_1, \dots, p_n with at most k_c , $c = 1, \dots, |C|$ labels per color c by the following formula:

$$\min_{y_i \in S \cup \{p_n^+\}, k'_1 \leq k_1, \dots, k'_{|C|} \leq k_{|C|}} \left(L(i, k'_1, \dots, k'_{|C|}) + \sum_{\frac{i+2}{3} \leq x \leq n} (y_i - p_x) \right).$$

If we bound the total number of labels by K , we obtain a similar dynamic program with the corresponding values $L(i, k)$, $i = 1, \dots, 3n$, $k < K$. □

We now turn our attention to the case of finite backbones and sketch how to modify the dynamic program for minimizing the total number of labels (see Theorem 1) to minimize the total leader length.

Theorem 4. *Given a set P of n colored points, a color set C , and a label bound K (or color vector \mathbf{k}), we can compute a feasible labeling of P with finite backbones that minimizes the total leader length in time $O(n^7 |C|^2 K^2)$ (or $O(n^7 |C|^2 (\prod_{c \in C} k_c)^2)$).*

Sketch of Proof. We change the meaning of an entry in the table T to denote the additional length of segments and backbones needed for labeling the points of the subinstance. Moreover, the precise positions of backbones matter for length

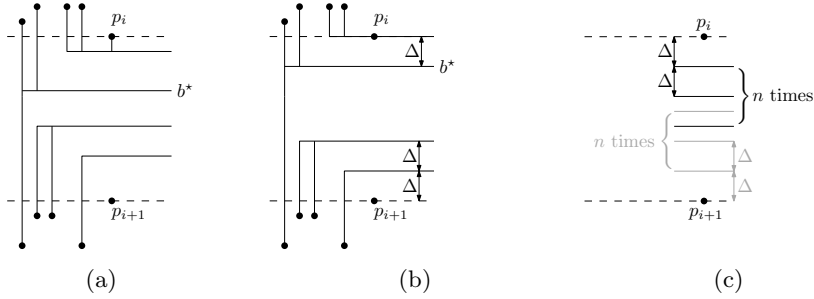


Fig. 2. (a) Longest backbone b^* splitting the backbones between p_i and p_{i+1} . (b) Backbones placed with the minimum leader length. (c) Candidate positions for backbones inside the gap

minimization. A clear candidate set is the set of the y-coordinates of input points which may be used by a backbone of the same color. We can also identify candidates for backbones inside a gap between points p_i and p_{i+1} . We observe that the longest backbone b^* inside the gap splits all other backbones lying between p_i and p_{i+1} ; see Fig. 2a. The backbones above b^* connect only to points above and, hence, must be placed as close to p_i as possible for length minimization. Symmetrically, the backbones below b^* connect only to the bottom and must be placed as close to p_{i+1} as possible.

For avoiding overlaps and to accommodate labels with fixed heights, we enforce a minimum distance $\Delta > 0$ between pairs of backbones, as well as backbones and differently colored points. Then, for the labels close to p_i , we get a sequence of consecutive candidate positions with distance Δ below p_i ; see Fig. 2b and 2c. Symmetrically, there is such a sequence above p_{i+1} . Any such sequence contains up to n points (less if the gap is too small). Note that the two sequences might overlap; we can, however, easily ensure that no two backbones with distance less than Δ are used in the dynamic program. To address entries in T we use the $O(n^2)$ candidate positions (input points and positions in gaps) instead of the gaps; no position can be used twice.

As a final step, we integrate the global value K or the color vector \mathbf{k} as a bound on the allowed numbers of labels. To this end, we add additional dimensions for K or for $k_c, c \in C$ to the table that specify the remaining available numbers of labels in the subinstance. \square

4 Crossing Minimization

In this section we allow crossings between backbone leaders, which generally allows us to use fewer labels. We concentrate on minimizing the number of crossings for the case $K = |C|$, i.e., one label per color, and distinguish fixed and flexible label orders.

4.1 Fixed y-Order of Labels

In this part, we assume that the color set C is ordered and we require that for each pair of colors $i < j$, the i -colored label is above the j -colored label.

Infinite Backbones. Observe that it is possible to slightly shift the backbones of a solution without increasing the number of crossings so that no backbone contains a point. So, the backbones can be assumed to be in the gaps between vertically consecutive points; we number the gaps from 0 to n , as in Section 2.

Theorem 5. *Given a set P of n colored points and an ordered color set C , a backbone labeling with one label per color, labels in the given color order, infinite backbones, and minimum number of crossings can be computed in $O(n|C|)$ time.*

Proof. Suppose that we fix the position of the i -th backbone to gap g . For $1 \leq i \leq |C|$ and $0 \leq g \leq n$, let $\text{cross}(i, g)$ be the number of crossings of the vertical segments of the non- i -colored points when the color- i backbone is placed at gap g . Note that this number depends only on the y-ordering of the backbones, which is fixed, and not on their actual positions. So, we can precompute the table cross , using dynamic programming, as follows. All table entries of the form $\text{cross}(\cdot, 0)$ can be clearly computed in $O(n)$ time. Then, $\text{cross}(i, g) = \text{cross}(i, g-1) + 1$, if the point between gaps $g-1$ and g has color j and $j > i$. In the case where the point between gaps $g-1$ and g has color j and $j < i$, $\text{cross}(i, g) = \text{cross}(i, g-1) - 1$. If it has color i , then $\text{cross}(i, g) = \text{cross}(i, g-1)$. From the above, it follows that the computation of table cross takes $O(n|C|)$ time.

Now, we use another dynamic program to compute the minimum number of crossings. Let $T[i, g]$ denote the minimum number of crossings on the backbones $1, \dots, i$ in any solution subject to the condition that the backbones are placed in the given ordering and backbone i is positioned in gap g . Clearly $T[0, g] = 0$ for $g = 0, \dots, n$. Moreover, we have $T[i, g] = \min_{g' \leq g} T[i-1, g'] + \text{cross}(i, g)$. Having pre-computed table cross and assuming that for each entry $T[i, g]$, we also store the smallest entry of row $T[i, \cdot]$ to the left of g , each entry of table T can be computed in constant time. Hence, table T can be filled in time $O(n|C|)$. Then, the minimum crossing number is $\min_g T[|C|, g]$. A corresponding solution can be found by backtracking in the dynamic program. \square

Finite Backbones. We can easily modify the approach used for infinite backbones to minimize the number of crossings for finite backbones, if the y-order of labels is fixed, as the following theorem shows.

Theorem 6. *Given a set P of n colored points and an ordered color set C , a backbone labeling with one label per color, labels in the given order, finite backbones, and minimum number of crossings can be computed in $O(n|C|)$ time.*

Proof. We present a dynamic program similar to the one presented in the proof of Theorem 5. Recall that all points of the same color are routed to the same label and the order of the labels is fixed, i.e., the label of the i -colored points is above the label of the j -colored points, when $i < j$. Here, the computation of the number of crossings when fixing a backbone at a certain position should take into consideration that the backbones are not of infinite length. Recall that the dynamic program could precompute these crossings, by maintaining an $n \times |C|$ table cross, in which each entry $\text{cross}(i, g)$ corresponds to the number of crossings of the non- i -colored points when the color- i -backbone is placed at gap g , for $1 \leq i \leq |C|$ and $0 \leq g \leq n$. In our case, $\text{cross}(i, g) = \text{cross}(i, g - 1) + 1$, if the point between gaps $g - 1$ and g is right of the leftmost of the i -colored points and has color j s.t. $j > i$. In the case, where the point between gaps $g - 1$ and g is right of the leftmost of the i -colored points and has color j and $j < i$, $\text{cross}(i, g) = \text{cross}(i, g - 1) - 1$. Otherwise, $\text{cross}(i, g) = \text{cross}(i, g - 1)$. Again, all table entries of the form $\text{cross}(\cdot, 0)$ can be clearly computed in $O(n)$ time. \square

4.2 Flexible y-Order of Labels

In this part the order of labels is no longer given and we need to minimize the number of crossings over all label orders. While there is an efficient algorithm for infinite backbones, the problem is NP-complete for finite backbones.

Infinite Backbones. We give an efficient algorithm for the case that there are $K = |C|$ fixed label positions y_1, \dots, y_K on the right boundary of R , e.g., uniformly distributed.

Theorem 7. *Given a set P of n colored points, a color set C , and a set of $|C|$ fixed label positions, we can compute in $O(n + |C|^3)$ time a feasible backbone labeling with infinite backbones that minimizes the number of crossings.*

Proof. First observe that if the backbone of color $k, 1 \leq k \leq |C|$ is placed at position $y_i, 1 \leq i \leq |C|$, then the number of crossings created by the vertical segments leading to this backbone is fixed, since all label positions will be occupied by an infinite backbone. This crossing number $\text{cr}(k, i)$ can be determined in $O(n_k + |C|)$ time, where n_k is the number of points of color k . In fact, by a sweep from top to bottom, we can even determine all crossing numbers $\text{cr}(k, \cdot)$ for backbone $k, 1 \leq k \leq |C|$ in time $O(n_k + |C|)$. Now, we construct an instance of a weighted bipartite matching problem, where for each position $y_i, 1 \leq i \leq |C|$ and each backbone $k, 1 \leq k \leq |C|$, we establish an edge (k, i) of weight $\text{cr}(k, i)$. In total, this takes $O(n + |C|^2)$ time. The minimum-cost weighted bipartite matching problem can be solved in time $O(|C|^3)$ with the Hungarian method [5] and yields a backbone labeling with the minimal number of crossings. \square

Finite Backbones. Next, we consider the variant with finite backbones and prove that it is NP-hard to minimize the number of crossings. For simplicity, we

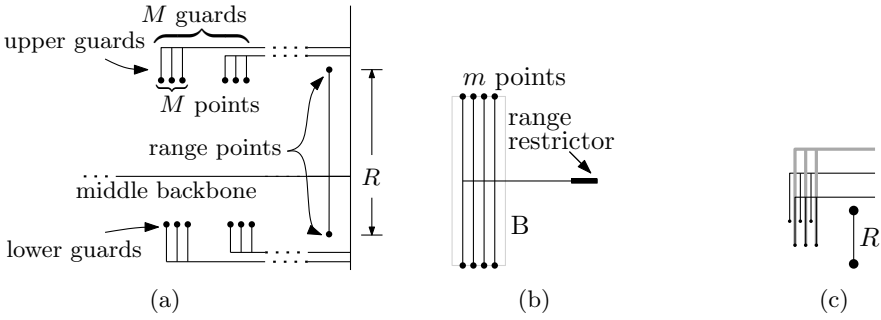


Fig. 3. (a) The range restrictor gadget, (b) a blocker gadget, (c) crossings caused by a pair of an upper and a lower guard that are positioned on the same side outside range R .

allow points that share the same x - or y -coordinates. This can be remedied by a slight perturbation. Our arguments do not make use of this special situation, and hence carry over to the perturbed constructions. We first introduce a number of gadgets that are required for our proof and sketch their properties.

The first one is the *range restrictor gadget*. Its construction consists of the middle backbone, whose position will be restricted to a given range R , and an upper and a lower *guard gadget* that ensure that positioning the middle backbone outside range R creates many crossings. We assume that the middle backbone is connected to at least one point further to the left such that it extends beyond all points of the guard gadgets. The middle backbone is connected to two *range points* whose y -coordinates are the upper and lower boundary of the range R . Their x -coordinates are such that they are on the right of the points of the guard gadgets. A *guard* consists of a backbone that connects to a set of M points, where $M > 1$ is an arbitrary number. The M points of a guard lie left of the range points. The upper guard points are horizontally aligned and lie slightly below the upper bound of range R . The lower guard points are placed such that they are slightly above the lower bound of range R . We place M upper and M lower guards such that the guards form pairs for which the guard points overlap horizontally. The upper (resp. lower) guard gadget is formed by the set of upper (resp. lower) guards. We call M the *size* of the guard gadgets. The next lemma shows properties of the range restrictor.

Lemma 2. *The backbones of the range restrictor can be positioned such that there are no crossings. If the middle backbone is positioned outside the range R , there are at least $M - 1$ crossings.*

Proof. The first statement is illustrated in Fig. 3a. To prove the second statement, assume to the contrary that the middle backbone is positioned outside range R , say w.l.o.g. below range R , and that there are fewer than $M - 1$ crossings. Observe that all guards must be positioned above the middle backbone, as a guard below the middle backbone would create M crossings, namely between the middle backbone and the segments connecting the points of the guard to its

backbone. So, the middle backbone is the lowest. Now observe that any guard that is positioned below the upper range point crosses the segment that connects this range point to the middle backbone. To avoid having $M - 1$ crossings, at least $M + 1$ guards (both upper and lower) must be positioned above range R . Hence, there is at least one pair consisting of an upper and a lower guard that are both positioned above range R . This, independent of their ordering, creates at least $M - 1$ crossings, a contradiction; see Fig. 3c, where the two alternatives for the lower guard are drawn in black and bold gray, respectively. \square

Let B be an axis-aligned rectangular box and R a small interval that is contained in the range of y -coordinates spanned by B . A *blocker gadget* of width m consists of a backbone that connects to $2m$ points, half of which are on the top and bottom side of B , respectively. A range restrictor gadget is used to restrict the backbone of the blocker to the range R ; see Fig. 3b. Note that, due to the range restrictor, this drawing is essentially fixed. We say that a backbone *crosses* the blocker gadget if its backbone crosses box B . It is easy to see that any backbone that crosses a blocker gadget creates m crossings, where m is the width of the blocker. We are now ready to present the NP-hardness reduction.

Theorem 8. *Given a set of n input points in k different colors and an integer Y it is NP-complete to decide whether a backbone labeling with one label per color and flexible y -order of the labels that has at most Y leader crossings exists.*

Proof. The proof is by reduction from the NP-complete Fixed Linear Crossing Number problem [8]: Given a graph $G = (V, E)$, a bijective function $f: V \rightarrow \{1, \dots, |V|\}$, and an integer Z , is there a drawing of G with the vertices placed on a horizontal line (*spine*) in the order specified by f and the edges drawn as semi-circles above or below the spine so that there are at most Z crossings? Masuda et al. [8] showed that the problem is NP-complete, even if G is a matching.

Let G be a matching. Then, the number of vertices is even and we can assume that the vertices $V = \{v_1, \dots, v_{2n}\}$ are indexed in the order specified by f , i.e., $f(v_i) = i$ for all i . We also direct each edge $\{v_i, v_j\}$ with $i < j$ from v_i to v_j . Let $\{u_1, \dots, u_n\}$ be the ordered source vertices and let $\{w_1, \dots, w_n\}$ be the ordered sink vertices; see Fig. 4a. In our reduction we will create an edge gadget for every edge of G . The gadget consists of five blocker gadgets and one *side selector gadget*. Each of the six sub-gadgets uses its own color and thus defines one backbone. The edge gadgets are ordered from left to right according to the sequence of source vertices (u_1, \dots, u_n) ; see Fig. 4b.

The edge gadgets are placed symmetrically with respect to the x -axis. We create $2n + 1$ special rows above the x -axis and $2n + 1$ special rows below, indexed by $-(2n + 1), -2n, \dots, 0, \dots, 2n, 2n + 1$. The gadget for an edge (v_i, v_j) uses five blocker gadgets (denoted as *central*, *upper*, *lower*, *upper gap*, and *lower gap* blockers) in two different columns to create two small gaps in rows j and $-j$, see the hatched blocks in the same color in Fig. 4b. The upper and lower blockers extend vertically to rows $2n + 1$ and $-2n - 1$. The gaps are intended to create two alternatives for routing the backbone of the side selector. Every backbone that starts left of the two gap blockers is forced to cross at least one of

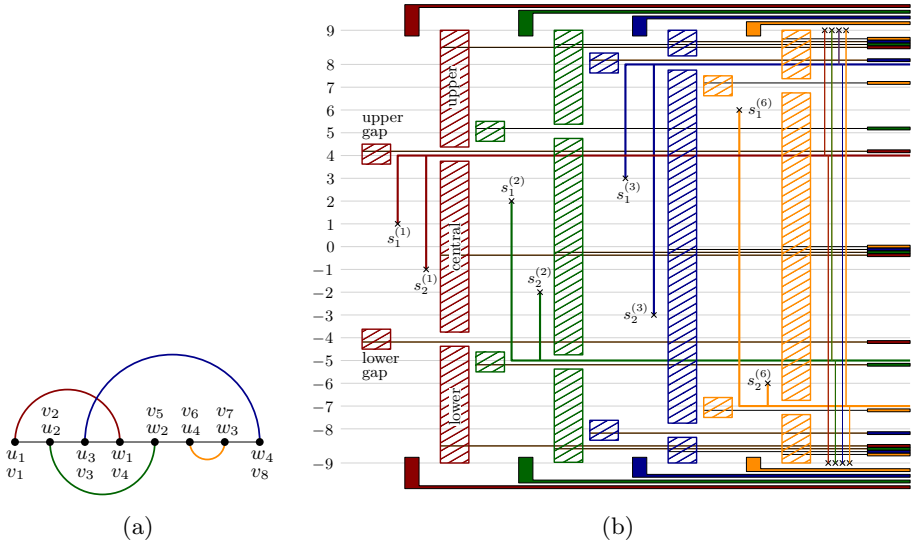


Fig. 4. (a) An input instance with four edges, (b) Sketch of the reduction for the graph of Fig. 4a. Hatched rectangles correspond to blockers, thick segments to side selectors, and filled shapes to guard gadgets or range restrictor gadgets.

these five blocker gadgets as long as it is vertically placed between rows $2n + 1$ and $-2n - 1$. The blockers have width $m = 8n^2$. Their backbones are fixed to lie between rows 0 and -1 for the central blocker, between rows $2n$ and $2n + 1$ ($-2n$ and $-2n - 1$) for the upper (resp. lower) blocker, and between rows j and $j + 1$ ($-j$ and $-j - 1$) for the upper (resp. lower) gap blocker.

The *side selector* consists of two horizontally spaced *selector points* $s_1^{(i)}$ and $s_2^{(i)}$ in rows i and $-i$ located between the left and right blocker columns. They have the same color and thus define one joint backbone that is supposed to pass through one of the two gaps in an optimal solution. The n edge gadgets are placed from left to right in the order of their source vertices; see Fig. 4b. The backbone of every selector gadget is vertically restricted to the range between rows $2n + 1$ and $-2n - 1$ in any optimal solution by augmenting each selector gadget with a range restrictor gadget. So, we add two more points for each selector to the right of all edge gadgets, one in row $2n + 1$ and one in row $-2n - 1$. They are connected to the selector backbone. In combination with a corresponding upper and lower guard gadget of size $M = \Omega(n^4)$ between the two selector points $s_1^{(i)}$ and $s_2^{(i)}$ this achieves the range restriction according to Lemma 2.

We now claim that in a crossing-minimal labeling the backbone of the selector gadget for every edge (v_i, v_j) passes through one of its two gaps in rows j or $-j$. The proof of this claim is based on three different options for placing a selector backbone: (a) outside its range restriction, i.e., above row $2n + 1$ or below row $-2n - 1$, (b) between rows $2n + 1$ and $-2n - 1$, but outside one of the two

gaps, and (c) in rows j or $-j$, i.e., inside one of the gaps. By this claim and the fact that violating any range restriction immediately causes M crossings, we can assume that every backbone adheres to the rules, i.e., stays within its range as defined by the range restriction gadgets or passes through one of its two gaps.

One can show that an optimal solution of the backbone labeling instance I_G created for a matching G with n edges has $X + 2Z$ crossings, where X is a constant depending on G , and Z is the minimum number of crossings of G in the Fixed Linear Crossing Number problem. The detailed proof is based on carefully counting crossings in four different cases, depending on which types of backbones and vertical segments intersect. It turns out that almost all crossings are fixed (yielding the number X), except for those of selector backbones with vertical selector segments for which the two underlying edges (v_i, v_j) and (v_k, v_l) with $i < k$ are *interlaced*, i.e., $i < k < j < l$ holds (yielding the number $2Z$). Note that we can guess an order of the backbones and apply Theorem 6 to compute the minimum crossing number, which concludes the NP-completeness proof. \square

Acknowledgements. This work was started at the Bertinoro Workshop on Graph Drawing 2013. M. Nöllenburg received financial support by the Concept for the Future of KIT. The work of M. A. Bekos and A. Symvonis is implemented within the framework of the Action “Supporting Postdoctoral Researchers” of the Operational Program “Education and Lifelong Learning” (Action’s Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State. We also acknowledge partial support by GRADR – EUROGIGA project no. 10-EuroGIGA-OP-003.

References

1. Bekos, M.A., Cornelsen, S., Fink, M., Hong, S., Kaufmann, M., Nöllenburg, M., Rutter, I., Symvonis, A.: Many-to-one boundary labeling with backbones. CoRR (2013), arXiv:1308.6801
2. Bekos, M.A., Kaufmann, M., Symvonis, A., Wolff, A.: Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry* 36(3), 215–236 (2007)
3. Hassin, R., Tamir, A.: Improved complexity bounds for location problems on the real line. *Operations Research Letters* 10(7), 395–402 (1991)
4. Kaufmann, M.: On map labeling with leaders. In: Albers, S., Alt, H., Näher, S. (eds.) *Efficient Algorithms*. LNCS, vol. 5760, pp. 290–304. Springer, Heidelberg (2009)
5. Kuhn, H.W.: The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2(1-2), 83–97 (1955)
6. Lin, C.C.: Crossing-free many-to-one boundary labeling with hyperleaders. In: Proc. IEEE Pacific Visualization Symp. (PacificVis 2010), pp. 185–192. IEEE (2010)
7. Lin, C.C., Kao, H.J., Yen, H.C.: Many-to-one boundary labeling. *Journal of Graph Algorithms and Applications* 12(3), 319–356 (2008)
8. Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Crossing minimization in linear embeddings of graphs. *IEEE Trans. Computers* 39(1), 124–127 (1990)

Streamed Graph Drawing and the File Maintenance Problem

Michael T. Goodrich and Paweł Pszozna

Dept. of Computer Science, University of California, Irvine

Abstract. In *streamed graph drawing*, a planar graph, G , is given incrementally as a data stream and a straight-line drawing of G must be updated after each new edge is released. To preserve the mental map, changes to the drawing should be minimized after each update, and Binucci *et al.* show that exponential area is necessary for a number of streamed graph drawings for trees if edges are not allowed to move at all. We show that a number of streamed graph drawings can, in fact, be done with polynomial area, including planar streamed graph drawings of trees, tree-maps, and outerplanar graphs, if we allow for a small number of coordinate movements after each update. Our algorithms involve an interesting connection to a classic algorithmic problem—the *file maintenance problem*—and we also give new algorithms for this problem in a framework where bulk memory moves are allowed.

1 Introduction

In the *streamed graph drawing* framework, which was introduced by Binucci *et al.* [4,3], a graph, G , is incrementally presented as a data stream of its vertices and edges, and a drawing of G must be updated after each new edge is released. So as to preserve the *mental map* [6,9] of the drawing, this framework also requires that changes to the drawing of G should be minimized after each update. Indeed, to achieve this goal, Binucci *et al.* took the extreme position of requiring that once an edge is drawn no changes can be made to that edge. They showed that, under this restriction, exponential area is necessary and sufficient for planar drawings of trees under various orderings for how the vertices and edges of the trees are presented.

In light of recent results regarding the mental map [1], however, we now know that moving a small number of vertices or edges in a drawing of a graph does not significantly affect readability in a negative way. Therefore, in this paper, we choose to relax the requirement that there are no changes to the drawing of the graph after an update and instead allow a small number of coordinate movements after each such update. In this paper, we study planar streamed graph drawing schemes for trees, tree-maps, and outerplanar graphs, showing that polynomial area is achievable for such streamed graph drawings if small changes to the drawings are allowed after each update. Our results are based primarily on an interesting connection between streamed graph drawing and a classic algorithmic problem, the *file maintenance problem*.

In the *file maintenance problem* [12], we wish to maintain an ordered set, S , of n elements, such that each element, x in S , is assigned a unique integer label, $L(x)$, in the range $[0, N]$, where x comes before y if and only if $L(x) < L(y)$. In the classic version of this problem, N is restricted to be $O(n)$, with the motivation that the integer labels are addresses or pseudo-addresses for memory locations where the elements of S are stored¹. If N is only restricted to be polynomial in n , then this is known as the *online list labeling problem* [2,5]. In either case, the set, S , can be updated by issuing a command, `insertAfter(x, y)`, where y is to be inserted to be immediately after $x \in S$ in the ordering, or `insertBefore(x, y)`, where y is to be inserted to be immediately before $x \in S$ in the ordering. The goal is to minimize the number of elements in S needing to be relabeled as a result of such an update.

Previous Related Results. For the file maintenance problem, Willard [12] gave a rather complicated solution that achieves $O(\log^2 n)$ relabelings in the worst case after each insertion, and this result was later simplified by Bender *et al.* [2]. For the online list labeling problem, Dietz and Sleator [5] give an algorithm that achieves $O(\log n)$ amortized relabelings per insertion, and $O(\log^2 n)$ in the worst-case, using an $O(n^2)$ tag range. Their solution was simplified by Bender *et al.* [2] with the same bounds. Recently, Kopelowitz [8] has given an algorithm that achieves $O(\log n)$ worst-case relabelings after each insertion, using a polynomial bound for N .

For streamed tree drawings, as we mentioned above, Binucci *et al.* [4,3] show that exponential area is required for planar drawings of trees, depending on the order in which vertices and edges are introduced (e.g, BFS, DFS, etc.).

Our Results. For the context of this paper, we focus on planar drawings of graphs, so we consider a drawing to consist essentially of a set of non-crossing line segments. For traditional drawings of trees and outerplanar graphs, the endpoints of the segments correspond to vertices and the segments represent edges. In tree-map drawings, each vertex v of a tree T is represented by a rectangle, R , such that the children of v are represented by rectangles inside R that share portions of at least two sides of R . Thus, in a tree-map drawing, the line segments correspond to the sides of rectangles.

We present new streamed graph drawing algorithms for general trees, tree-maps, and outerplanar graphs that keep the area of the drawing to be of polynomial size and allow new edges to arrive in any order, provided the graph is connected at all times. After each update to a graph is given, we allow a small number of, say, a polylogarithmic number of the endpoints of the segments in the drawing to move to accommodate the representation of the new edge. We alternatively consider these to be movements of either individual endpoints or sets of at most B endpoints, for a parameter B , provided that each set of such

¹ For instance, in the EDT text editor developed for the DEC PDP-11 series of mini-computers, each line was assigned a pseudo line number, 1.0000, 2.0000, and so on, and if a new line was to be introduced between two existing lines, x and y , it was given as a default label the average of the labels of x and y as its label.

endpoints is contained in a given convex region, R , and all the endpoints in this region are translated by the same vector. We call such operations the *bulk moves*.

All of our methods are based on our showing interesting connections between the streamed graph drawing problems we study and the file maintenance problem. In addition to utilizing existing algorithms for the file maintenance problem in our graph drawing schemes, we also give a new algorithm for this classic problem in a framework where bulk memory moves are allowed, and we show how this solution can also be applied to streamed graph drawing.

2 Building Blocks

The Ordered Streaming Model. We start with the description of the model under which we operate. At each time $t \geq 1$, a new edge, e , of a graph, G , arrives and has to be incorporated immediately into a drawing of G , using line segments whose endpoints are placed at grid points with integer coordinates. Since we are focused on planar drawings in this paper, together with the edge, e , we also get the information of its relative position among the neighbors of e 's endpoints (i.e., for every vertex we know the clockwise order of its neighbors and e 's placement in this order). At all times, the current graph, G , is connected, and the edges never disappear (infinite persistence).

Incidentally, the streaming model of Binucci *et al.* [4] is slightly different, in that edges arrive without the order information in their model. Under that model, they have given an $\Omega(2^{\frac{2}{3}})$ area lower bound for binary tree drawing and an $\Omega(n(d-1)^n)$ lower bound for drawing trees with maximum degree $d > 2$. These bounds stand when the algorithm is not allowed to move any vertex. However, they only apply to a very restricted class of algorithms, namely *predefined-location algorithms*, which are non-adaptive algorithms whose behavior does not depend on the order in which the edges arrive or the previously drawn edges. Also, as noted above, Binucci *et al.* do not allow for vertices to move once they are placed. As we show in the following theorem, even with the added information regarding the relative placement of an edge among its neighbors incident on the same vertex, if we don't allow for vertex moves, we must allow for exponential area. In the full version of this paper [7] we prove the following.

Theorem 1. *Under the ordered streaming model without vertex moves, any tree drawing algorithm requires $\Omega(2^{n/2})$ area in the worst case.*

File Maintenance with Bulk Moves. Here we consider the file maintenance problem and the online list labeling problem variants where we allow for bulk moves² of an interval of B labels, for some parameter B . We have already mentioned the known results for the file maintenance problem, where the only type of relabelings that are allowed are for individual elements, in which $O(\log^2 n)$

² Note that bulk moves are also motivated for the original file maintenance problem if we define the complexity of a solution in terms of the number of commands that are sent to a DMA controller for bulk memory-to-memory moves.

worst-case relabelings suffice for each update when N is $O(n)$ [2,12] and $O(\log n)$ suffice in the worst-case when N is a polynomial in n [8].

Bulk moves allow us to improve on these bounds. We have achieved several tradeoffs between the operation count and the size of B . Of course, if B is n , then achieving constant number of operations is easy, since we can maintain the n elements to have the indices 1 to n , and with each insertion, at some rank i , we simply move the elements currently from i to n up by one, as a single bulk move. Theorem 2 summarizes the rest of our results.

Theorem 2. *We can achieve the following bounds:*

1. $O(1)$ worst-case relabeling bulk moves suffice for the file maintenance problem if $B = n^{1/2}$.
2. $O(1)$ worst-case relabeling bulk moves suffice for the online list maintenance problem if $B = \log n$.
3. $O(\log n)$ worst-case relabeling bulk moves suffice for the file maintenance problem if $B = \log n$.

Proof.

1. This is accomplished in an amortized way by partitioning the array into $n^{1/2}$ chunks of size at most $2n^{1/2}$. Whenever a chunk, i , grows to have size $n^{1/2}$, we move all the chunks to the right of i by one chunk (using $O(n^{1/2})$ bulk moves). Then we split the chunk i in two, keeping half the items in chunk i and moving half to chunk $i + 1$. These moves are charged to the previous $n^{1/2}/2$ insertions in chunk i . Turning this bound into a worst-case bound is then done using standard de-amortization techniques.
2. This is accomplished by modifying a two-level structure of Kopelowitz [8]. Kopelowitz used the top level of this structure to maintain order of sublists of size $O(\log n)$ each. Order within each sublist was maintained using standard file maintenance problem solutions. Our modification is that each sublist is now represented as a small subarray of size $O(\log n)$ and operations on the top level of Kopelowitz's structure are simulated using bulk moves on a big array containing all concatenated subarrays.
3. This is accomplished by using the method of Bender *et al.* [2] and noting that each insertion in their scheme uses a process where each substep involves moving a contiguous subarray of size $O(\log n)$ using $O(\log n)$ individual moves. Each such move can alternatively be done using $O(1)$ bulk moves of subarrays of size $\log n$. \square

3 Streamed Graph Drawing of Trees

In this section, we present several algorithms for upward grid drawings of trees in the ordered streaming model. The algorithms are designed to accommodate different types of vertex moves allowed. For example, by a *bulk move*, we mean a move that translates all segment endpoints that belong to a given convex region,

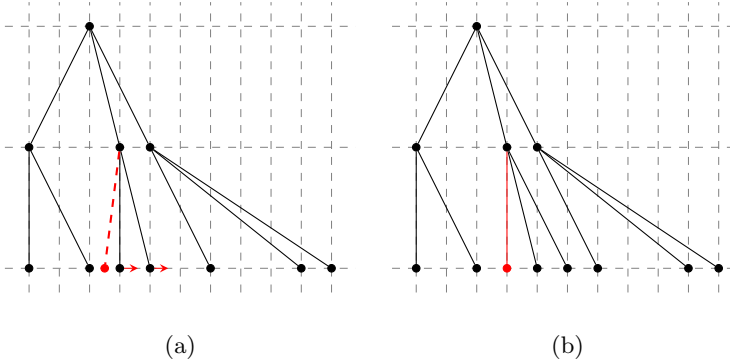


Fig. 1. Illustrating an insertion for our tree-drawing scheme: (a) determining relative position for the new (dashed, red) edge; (b) tree after edge insert and related vertex moves

R , by the same vector. This corresponds to the observation [13] that moving a small number of elements in the same direction is easy to follow and does not interfere with the ability to understand the structure of the drawing (as long as there are no intersections).

Let G be a tree that is revealed one edge at a time, keeping the graph connected. Algorithm 1 selects one endpoint of the first edge, r , puts it at position $(0, 0)$, and produces an upward straight-line grid drawing of G , level-by-level, with each edge from parent to child pointed downwards. (If a new edge is ever revealed for the current parent, we simply recalibrate what we are calling position $(0, 0)$ without changing the position of the vertices already drawn.) For the k th level, L_k , with n_k nodes, we place nodes in positions $(0, -k)$ through $(N, -k)$ in the order of their parents (to avoid intersections), where $N \geq n_k$. When a new edge is added, we locate the position (row and position in the row) of the new node and insert the new node after its predecessor (or before its successor), shifting other nodes on this level as needed to make room for the new node. (See Fig. 1.) The details are as shown in Algorithm 1.

Input: $e = (a, b)$, the edge to be added; b is the new vertex

- 1: $k \leftarrow b$'s distance from r
- 2: determine c , b 's predecessor (or successor) in level L_k
- 3: perform $L_k.\text{insertAfter}(c, b)$ (or $L_k.\text{insertBefore}(c, b)$), giving b integer label $L(b)$
- 4: move vertices whose labels have changed in the previous step
- 5: place b at $(L(b), -k)$ and draw e

Algorithm 1. Generic insertion algorithm for upward straight-line grid streamed tree drawing

Drawing the tree in this fashion ensures there are no intersections (edges connect only vertices in two neighboring levels), even as the vertices are shifted (relative order of vertices stays the same). In addition, there are at most $O(n)$ levels, and at most $O(n)$ nodes per level.

Theorem 3. *Depending on the implementation for the insertBefore and insertAfter methods, Algorithm 1 maintains a straight-line upward grid drawing of a tree in the ordered streaming model to have the following possible performance bounds:*

1. $O(n^2)$ area and $O(1)$ vertex moves per insertion if bulk moves of size $n^{1/2}$ are allowed.
2. $O(n^2)$ area and $O(\log n)$ vertex moves per insertion if bulk moves of size $\log n$ are allowed.
3. $O(n^2)$ area and $O(\log^2 n)$ vertex moves per insertion if bulk moves are not allowed.
4. polynomial area and $O(1)$ vertex moves per insertion if bulk moves of size $\log n$ are allowed.
5. polynomial area and $O(\log n)$ vertex moves per insertion if bulk moves are not allowed.

Proof. The claimed bounds follow immediately from Theorem 2. □

Note that $\Omega(n^2)$ area is necessary in the worst case for an upward straight-line grid drawing of a tree if siblings are always placed on the same level.

4 Streamed Graph Drawing of Tree-Maps

A tree-map, M , is a visualization technique introduced by Shneiderman [10], which draws a rooted tree, T , as a collection of nested rectangles. The root, r , of T is associated with a rectangle, R , and if r has k children, then R is partitioned into k sub-rectangles using line segments parallel to one of the coordinate axes (say, the x -axis), with each such rectangle associated with one of the children of r . Then, each child rectangle is recursively partitioned using line segments parallel to the other coordinate axis (say, the y -axis). (See Fig. 2.)

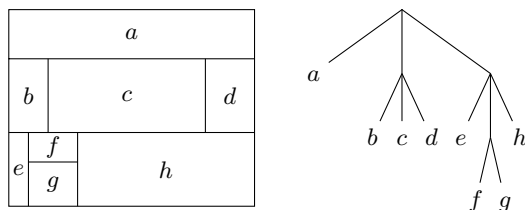


Fig. 2. A tree-map and its associated tree

We assume in this case that a tree, G , is released one edge at a time, as in the previous section. We assume inductively that we have a tree-map drawn for G , with a global set, X , of all x -coordinates maintained for the rectangle boundaries and a global set, Y , of all y -coordinates maintained for the rectangle boundaries. When an edge, e , of a rectangle has to be moved, the largest segment containing e is moved accordingly. Our insertion method is shown in Algorithm 2 (for brevity, the case when a vertex has no predecessors among its siblings is omitted).

Input: $e = (a, b)$, the edge to be added; b is a new child vertex

- 1: Let R be the rectangle for a , and let z be the primary axis for R (w.l.o.g., $z = x$)
- 2: **if** b has no siblings **then**
- 3: $R_b \leftarrow R$ (and give it primary axis orthogonal to z)
- 4: **else if then**
- 5: **else**
- 6: determine c , b 's predecessor sibling (w.l.o.g.), and let R_c be c 's rectangle
- 7: perform $X.\text{insertAfter}(R_c.x_{\max}, b)$, giving b integer label $L(b)$
- 8: move segment endpoints whose labels have changed in the previous step
- 9: $R_b \leftarrow$ the rectangle in R with left boundary $R_c.x_{\max}$ and right boundary $L(b)$
- 10: **end if**

Algorithm 2. Generic insertion algorithm for streamed tree-map drawing

Theorem 4. *Depending on the implementation for the `insertBefore` and `insertAfter` methods, Algorithm 2 maintains a tree-map drawing of a tree in the ordered streaming model to have the following possible performance bounds:*

1. $O(n^2)$ area and $O(1)$ x - and y -coordinate moves per insertion if bulk moves of size $n^{1/2}$ are allowed.
2. $O(n^2)$ area and $O(\log n)$ x - and y -coordinate moves per insertion if bulk moves of size $\log n$ are allowed.
3. $O(n^2)$ area and $O(\log^2 n)$ x - and y -coordinate moves per insertion if bulk moves are not allowed.
4. polynomial area and $O(1)$ x - and y -coordinate moves per insertion if bulk moves of size $\log n$ are allowed.
5. polynomial area and $O(\log n)$ x - and y -coordinate moves per insertion if bulk moves are not allowed.

Proof. The claimed bounds follow immediately from Theorem 2. □

5 Streamed Graph Drawing of Outerplanar Graphs

Our algorithm for drawing outerplanar graphs in the streaming model is based on a well-known fact about outerplanar graphs, namely that any outerplanar graph may be drawn with straight-line edges and without intersections in such a way that the vertices are placed on a circle [11].

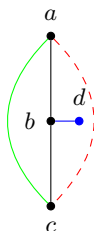


Fig. 3. Situation where information about order of edges around vertex is insufficient. Initially, there are vertices a, b, c and edges $(a, b), (b, c)$. When a new edge (a, c) is added, it can be drawn in two ways (solid green or dashed red) – ordering of edges does not specify which one to choose. When edge (b, d) arrives (with edge order (a, d, c) around b), if the dashed red edge location was selected, there is no way to move the vertices without intersections to produce an outerplanar drawing.

As previously, we assume that each new edge comes with the information about its relative placement among its endpoints’ incident edges. In other words, for each vertex, we know the clockwise order of its incident edges. Fig. 3 shows a situation when this information alone is not enough, however.

Nevertheless, this type of problem can only happen when the new edge connects two vertices of degree 1 as shown below.

Lemma 1. *If at least one of the newly connected vertices has degree > 1 , the information about relative order of incident edges suffice.*

Proof. Consider the situation shown in Fig. 4. (p, q) is the new edge. The graph is connected, and the path between p and q is shown. The initial direction of the edge (bold part) is determined by the ordering of edges around p . Then the edge can either go around r (shown in dashed red) or not (solid green). Obviously, the dashed red edge location is invalid, as it would surround r with a face, violating the requirement that each vertex belongs to the outer plane. Therefore, there is only one possibility for correctly drawing the edge. \square

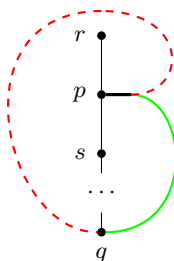


Fig. 4. Of the two possibilities of drawing new edge (p, q) , only solid green is valid

Invariant: vertices are placed on the circle in the same order as they appear on the outer face of the drawing

Input: outerplanar drawing of graph G on a circle; $e = (p, q)$ – edge to be added

- 1: **if** q is a new vertex **then**
- 2: place q on the circle according to ordering that includes e
- 3: **else**
- 4: add a *virtual* arc e' connecting p and q according to the specification of e s.t. e' does not intersect any existing edge
- 5: calculate order O of vertices on the outer plane (taking e' into account)
- 6: move vertices into place on the circle according to O
- 7: **end if**
- 8: draw e

Algorithm 3. Adding new edge to outerplanar drawing of graph G

It follows that when the new edge connects two vertices of degree 1, additional information (such as relative order of the vertices on the outer face) is necessary.

We present our streamed drawing algorithm for an outerplanar graph, G , in terms of placing vertices of G on a circle. We will later derive an algorithm for drawing G using grid points. We show in Algorithm 3 how to handle adding a new edge to the graph.

As mentioned previously, maintaining the invariant guarantees planarity of the drawing. We now show that vertices can move into place without causing any intersections in the process.

Lemma 2. *Moving a vertex v inside the circle along a trajectory that does not intersect any edge non-incident to v does not introduce any intersections and maintains the order of edges around v .*

Proof. Consider the drawing with edges incident to v removed (marked with dashed lines in Fig. 5). The face to which v belongs (limited by edges and circle boundary) is the area where v can move. Because it is convex, as v moves, its incident edges never intersect the boundaries of the area (other than at their incident vertices), and the relative order of the edges stays the same. □

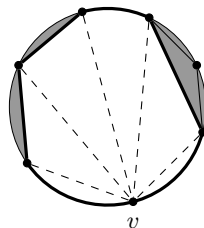


Fig. 5. Vertex v can move in the (convex) white area without causing intersections or edge order changes

Lemma 3. *A vertex, v , that moves into its new position can do so without crossing any edges.*

Proof. Consider the face (in the sense of Lemma 2 and Fig. 5) of the drawing that v belongs to. The drawing is still outerplanar after adding the *virtual* arc (which is not necessarily straight-line), and therefore at least part of the circle, C' , that forms this face's border still belongs to the outer face of the drawing (see vertex c in Fig. 6). By Lemma 2, v can move to C' without crossing any edges. When there are more vertices whose destination is the same part of the circle, C'' , (vertices d, e and f in Fig. 6), they must form a path with inner vertices all having degree 2. After adding the new edge, their order on the circle (and hence on C'') is the reverse of their current order, so their (straight-line) trajectories do not cross. Since they form a path, edges between them will not intersect as they move into place. \square

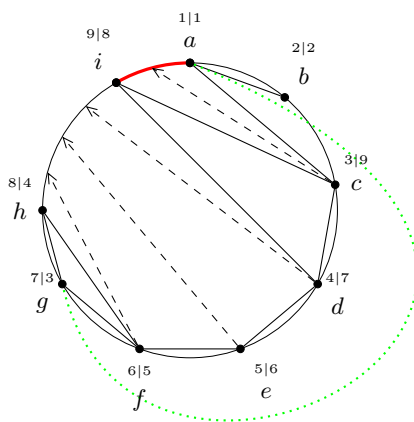


Fig. 6. Adding edge (a, g) . *Virtual* arc is drawn in green dots. Part of the circle that lies in the outer plane and is reachable from c is shown in bold red. Numbers above vertices denote the order of vertex before and after the edge is added. Dashed lines are the trajectories of the vertices that need to move to maintain the invariant.

Corollary 1. *Algorithm 3 maintains an outerplanar drawing of a graph G as new edges are added to it.*

Extending Algorithm 3 to placing nodes on a grid is straightforward. Instead of a circle, we operate on a set of grid points in convex position that are approximately circular. We apply one of the algorithms for the file maintenance problem or the online list labeling problem for maintaining order of vertices in this set. When such an algorithm would move vertex v , we first check if there is an unused grid point between new neighbors of v on the circle. If so, we simply move the vertex to that point. Otherwise, we “reserve” the destination for v by inserting a stub vertex in the correct place (between new neighbors of v) on the circle. The list labeling algorithm will move vertices around the circle (without changing order on the circle, so it will not cause any intersections) to make room for this stub. Afterwards, we move v into its reserved position.

Lemma 4. *Vertex v is moved in line 6 of Algorithm 3 at most $\deg(v) - 1$ times.*

Proof. A vertex v is moved when the new edge forms a *shortcut* that bypasses v on the outer face. v can appear at most $\deg(v)$ times on the outer face, so after $\deg(v) - 1$ moves, there will be only one valid position for v on the outer face, so it cannot be bypassed anymore. (See Fig. 7.) □

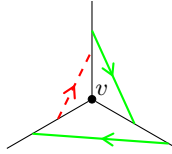


Fig. 7. Vertex v has degree 3. After two shortcuts (solid green lines) around v have been added, adding a third (dashed red line) would completely surround v , violating outerplanarity. Arrows show direction of edges on the outer plane.

Theorem 5. *The grid-based version of Algorithm 3 maintains an outerplanar drawing of a graph G and has the following update performances: uses $O(\log n)$ amortized moves per vertex, and*

1. $O(n^3)$ area and $O(\log^2 n)$ vertex moves per edge insertion.
2. polynomial area and $O(\log n)$ vertex moves per edge insertion.

In addition, each of the above complexity bounds applies in an amortized sense per vertex in the drawing.

Proof. By Corollary 1, we know that the algorithm maintains an outerplanar drawing of G . For the area bounds, the file maintenance algorithms requires $O(n)$ available integer tags (in this case, points in convex position) to handle n elements. Since m grid points in (strict) convex position require $O(m^3)$ area, the streamed drawing algorithm therefore uses $O(n^3)$ area in such cases. Likewise, it uses polynomial area when using a solution to the online list labeling problem.

By Lemma 4, each vertex v is moved by Algorithm 3 at most $\deg(v) - 1$ times. Each such move requires at most one insertion into the list for the file maintenance or list maintenance algorithm, so there are at most $O(1)$ such insertions per vertex in the amortized sense (sum of degrees of all vertices in an outerplanar graph is $O(n)$). □

Note that we cannot immediately apply our results for bulk moves, unless we restrict our attention to possible vertex points that are uniformly distributed on a circle and moves that involve rotations of intervals of points around this circle.

6 Conclusion

We provided a revised approach to streamed graph drawing, utilizing solutions to the file maintenance problem, either on a level-by-level basis (for level drawings of

trees), a cross-product basis (for tree-maps), or a circular/convex-position basis (for outerplanar graphs). For future work, it would be interesting to find other applications of this problem in streamed or dynamic graph drawing applications.

Acknowledgements. We would like to thank Alex Nicolau and Alex Veidenbaum for helpful discussions regarding the file maintenance problem. This work was supported in part by the NSF, under grants 1011840 and 1228639.

References

1. Archambault, D., Purchase, H.C.: Mental map preservation helps user orientation in dynamic graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 475–486. Springer, Heidelberg (2013)
2. Bender, M.A., Cole, R., Demaine, E.D., Farach-Colton, M., Zito, J.: Two simplified algorithms for maintaining order in a list. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 152–164. Springer, Heidelberg (2002)
3. Binucci, C., Brandes, U., Battista, G.D., Didimo, W., Gaertler, M., Palladino, P., Patrignani, M., Symvonis, A., Zweig, K.: Drawing trees in a streaming model. *Information Processing Letters* 112(11), 418–422 (2012)
4. Binucci, C., Brandes, U., Di Battista, G., Didimo, W., Gaertler, M., Palladino, P., Patrignani, M., Symvonis, A., Zweig, K.: Drawing trees in a streaming model. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 292–303. Springer, Heidelberg (2010)
5. Dietz, P., Sleator, D.: Two algorithms for maintaining order in a list. In: 19th ACM Symp. on Theory of Computing (STOC), pp. 365–372 (1987)
6. Eades, P., Lai, W., Misue, K., Sugiyama, K.: Preserving the mental map of a diagram. In: *Proceedings of Compugraphics*, pp. 24–33 (1991)
7. Goodrich, M.T., Pszona, P.: Cole’s parametric search technique made practical. *CoRR* (arXiv ePrint), abs/1308.6711 (2013)
8. Kopelowitz, T.: On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In: *FOCS*, pp. 283–292. IEEE Computer Society (2012)
9. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. *Journal of Visual Languages & Computing* 6(2), 183–210 (1995)
10. Shneiderman, B.: Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.* 11(1), 92–99 (1992)
11. Tamassia, R.: *Handbook of Graph Drawing and Visualization*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC (2007)
12. Willard, D.E.: Good worst-case algorithms for inserting and deleting records in dense sequential files. In: *ACM SIGMOD*, pp. 251–260 (1986)
13. Yantis, S.: Multielement visual tracking: Attention and perceptual organization. *Cognitive Psychology* 24(3), 295–340 (1992)

COAST: A Convex Optimization Approach to Stress-Based Embedding

Emden R. Gansner, Yifan Hu, and Shankar Krishnan

AT&T Labs - Research, Florham Park, NJ

Abstract. Visualizing graphs using virtual physical models is probably the most heavily used technique for drawing graphs in practice. There are many algorithms that are efficient and produce high-quality layouts. If one requires that the layout also respect a given set of non-uniform edge lengths, however, force-based approaches become problematic while energy-based layouts become intractable. In this paper, we propose a reformulation of the stress function into a two-part convex objective function to which we can apply semi-definite programming (SDP). We avoid the high computational cost associated with SDP by a novel, compact re-parameterization of the objective function using the eigenvectors of the graph Laplacian. This sparse representation makes our approach scalable. We provide experimental results to show that this method scales well and produces reasonable layouts while dealing with the edge length constraints.

1 Introduction

For visualizing general undirected graphs, algorithms based on virtual physical models are some of the most frequently used drawing methods. Among these, the spring-electrical model [7,8] treats edges as springs that pull nodes together, and nodes as electrically-charged entities that repel each other. Efficient and effective implementations [13,14,26] usually utilize a multilevel approach and fast force approximation with a suitable spatial data structure, and can scale to millions of vertices and edges while still producing high-quality layouts.

In certain instances, the graph may assign non-uniform lengths to its edges, and the layout problem will have the additional constraint of trying to match these lengths. A suitable formulation of the spring-electrical model that works well when edges have predefined target lengths is still an open problem.

In contrast, the (full) stress model assumes that there are springs connecting all vertex pairs of the graph. Assuming we have a graph $G = (V, E)$, with V the set of vertices and E the set of edges, the energy of this spring system is

$$\sum_{i,j \in V} w_{ij} (||x_i - x_j|| - d_{ij})^2, \quad (1)$$

where d_{ij} is the ideal distance between vertices i and j , and w_{ij} is a weight factor. The weight factor can modify the impact of an error. Weights can be arbitrary but are frequently taken as a negative power of d_{ij} , thus lessening the error for larger ideal distances. A layout that minimizes this stress energy is taken as an optimal layout of the

graph. The justification for this is clear: in most cases, it is not possible to find a drawing that respects all of the edge lengths, while expression (1) is basically the weighted mean square error of a drawing. (See also the work of Brandes and Pich [2].)

The stress model has its roots in multidimensional scaling (MDS) [19] which was eventually applied to graph drawing [16,20]. Note that typically we are given only the ideal distance between vertices that share an edge, which is taken to be unit length for graphs without predefined edge lengths. For other vertex pairs, a common practice is to define d_{ij} as the length of a shortest path between vertex i and j . Such a treatment, however, means that an all-pairs shortest path problem must be solved. Johnson’s algorithm [15] takes $O(|V|^2 \log |V| + |V||E|)$ time, and $O(|V|^2)$ memory. (A slightly faster, but still quadratic, algorithm is also known [23].) For large graphs, such complexities make solving the full stress model infeasible.

A number of techniques have been proposed for circumventing this problem, typically focused on approximate solutions, using only a few computed distances, or approximating the shortest path calculations. Gansner et al. [12] proposed another approach for solving the “stress model” efficiently, by redefining the problem. The key idea was to note that only the edge distances are given, while using shortest path lengths for the remainder is somewhat arbitrary, and could be replaced with some other constraint that is faster to compute but still works in terms of layout quality. This led them to propose a two-part modified stress function

$$\sum_{\{i,j\} \in E} w_{ij} (\|x_i - x_j\| - d_{ij})^2 - \alpha H(x), \quad (2)$$

where the first term encodes the stress associated with the given distances, and the second handles the remaining pairs.

In this paper, we also consider minimizing a two-part modified stress function. However, our formulation is such that the objective function is convex. More specifically, it is quartic in the positions of the nodes, and can be expressed as a quadratic function of auxiliary variables, where each of the auxiliary variables is a product of positions. We solve the problem by projecting the positions into a subspace spanned by the eigenvectors of the Laplacian, and transform the minimization problem into one of convex programming. We call our technique COAST (Convex Optimization Approach to STress-based embedding).

The rest of the paper is organized as follows. In Section 2, we discuss related work. Section 3 gives the COAST model, and discusses a way to solve the model by semi-definite programming. Section 4 evaluates our algorithm experimentally by comparing it with some of the existing fast approximate stress models. Section 5 presents a summary and topics for further study.

2 Related Work

Most of the earlier approaches [24,10,1,17,12] for efficiently handling graph drawings with edge lengths relied on approximately minimizing the stress model, typically using some sparse model [10]. One notable effort is that of PivotMDS of Brandes and Pich

[1]. This is an approximation algorithm which only requires distance calculations from all nodes to a few chosen nodes.

While PivotMDS is very efficient and works well for some graphs, for graphs such as trees, multiple nodes can share the same position. Khoury et al. [17] approximate the solution of the linear system in a stress majorization procedure [10] by a low-rank singular value decomposition (SVD). They used a result of Drineas et al. [6] which states that for a matrix with well-distributed SVD values, the SVD values and left SVD vectors of the submatrix consisting of randomly sampled columns of the original matrix are a good approximation to the corresponding SVD values and vectors of the original matrix. With this result, they were able to calculate only the shortest paths from a selected number of nodes, as in PivotMDS. The method avoided having nodes in a tree-like graph being embedded into the same position by using the exact (instead of approximate) right-hand-side of the stress majorization procedure, using an observation that this right-hand-side can be calculated efficiently for the special case of $w_{ij} = 1/d_{ij}$.

The work most akin to that presented here is the maxent-stress model [12]. That approach borrows from the principle of maximal entropy, which says that items should be placed uniformly in the absence of constraints. The model tries to minimize the local stresses, while selecting a layout that maximizes the dispersion of nodes. This leads to the function shown in expression (2), where typically $H(x) = \ln_{\{i,j\} \notin E} \|x_i - x_j\|$. The authors introduce an algorithm, called force-augmented stress majorization, to minimize this objective function.

Although it essentially ignores edge lengths, the binary stress model of Koren and Çivril [18] is stylistically related, in that the first term attempts to specify edge lengths (as uniformly 0) and the second term has the effect of uniformly spacing the nodes. Specifically, there is a distance of 0 between nodes sharing an edge, and a distance of 1 otherwise, giving the model

$$\sum_{\{i,j\} \in E} \|x_i - x_j\|^2 + \alpha \sum_{\{i,j\} \notin E} (\|x_i - x_j\| - 1)^2.$$

Similarly, Noack [21,22] has proposed the LinLog model and, more generally, the r -PolyLog model,

$$\sum_{\{i,j\} \in E} \|x_i - x_j\|^r - \sum_{i,j \in V} \ln \|x_i - x_j\|,$$

where, in particular, the second term is suggestive of the use of entropy in the maxent-stress model.

The most significant attempt to use a force-directed approach for encoding edge distances was the GRIP algorithm [9]. The multilevel coarsening uses maximal independent set based filtration, with the length of an edge at a coarse level computed from lengths of its composite edges. On coarse levels, the algorithm uses a version of the Kamada-Kawai algorithm [16] applied to each node within a local neighborhood of the original graph, thus handling the relevant edge lengths. On the finest level, however, a localized Fruchterman-Reingold algorithm [8] is used, with no modeling of edge lengths.

In the area of data clustering, Chen and Buja [3] present LMDS, a model based on localized versions of MDS. Algebraically, this reduces to

$$\sum_{\{i,j\} \in S} (\|x_i - x_j\| - d_{ij})^2 - t \sum_{(i,j) \notin S} \|x_i - x_j\|,$$

where S contains $\{i, j\}$ if node j is among the k nearest neighbors of i . It is difficult to determine how scalable this approach is but some tests indicate it is not appropriate for graph drawing.

3 The COAST Algorithm

Let $G = (V, E)$ denote an undirected graph, with the node set (vertices) V and edge set E . We use $n = |V|$ for the number of vertices in G . We assume that each edge (i, j) has a desired length d_{ij} with weight w_{ij} . Typically, one sets $w_{ij} = 1/d_{ij}^2$, but our analysis does not require that assumption. We wish to embed G into d -dimensional Euclidean space. Let x_i represent the coordinates of vertex i in \mathbb{R}^d , and let P be the $n \times d$ matrix whose rows are the x_i . We define the Gram matrix $X = (x_{ij})$ where $x_{ij} = x_i \cdot x_j$, the matrix of inner products. It is well known that X is a positive semi-definite matrix.

We consider minimizing a two-part modified stress function:

$$T(P) = \sum_{\{i,j\} \in E} (w_{ij}\|x_i - x_j\|^2 - w_{ij}d_{ij}^2)^2 - t\lambda \sum_{(i,j) \notin E} \|x_i - x_j\|^2, \tag{3}$$

where the first term attempts to assign edges their ideal edge lengths, and the second term separates unrelated nodes as much as possible. The parameter t can be used to balance the two terms, emphasizing either conformity to the specified edge lengths (small t) or uniform placement (large t). Without loss of generality, we can assume a zero mean for the x_i , i.e., $\sum_i x_i = 0$. We set $\lambda = |E| / \left(\binom{n}{2} - |E| + 1 \right)$ to balance the relative size of the two terms, as suggested by Chen and Buja [3]. To minimize $T(P)$, let T_1 and T_2 be the first and second terms of T , respectively, so that $T = T_1 - T_2$, and consider the first term. We have the following derivation:

$$\begin{aligned} T_1 &= \sum_{\{i,j\} \in E} \{w_{ij}(x_{ii} - x_{ij} - x_{ji} + x_{jj}) - w_{ij}d_{ij}^2\}^2 \\ &= \sum_{\{i,j\} \in E} \{w_{ij}Tr(E_{ij}X) - w_{ij}d_{ij}^2\}^2. \end{aligned} \tag{4}$$

where $Tr()$ is the trace function and $E_{ij} = (e_{kl})$ is the $n \times n$ matrix with

$$e_{kl} = \begin{cases} 1, & \text{if } k = l = i \text{ or } k = l = j \\ -1, & \text{if } k = i \text{ and } l = j \\ -1, & \text{if } k = j \text{ and } l = i \\ 0, & \text{otherwise} \end{cases}$$

Using standard properties of the trace, the expression (4) can be rewritten as

$$\sum_{\{i,j\} \in E} w_{ij}^2 \{ \text{vec}(E_{ij})^T \mathcal{X} - d_{ij}^2 \}^2, \tag{5}$$

where $\mathcal{X} = \text{vec}(X)$ and $\text{vec}()$ is the matrix vectorization operator.

Functions defined on nodes of a graph can be well approximated by the eigenvectors of the graph Laplacian [4], and the smoother the function is, fewer eigenvectors are required to approximate it well. It is reasonable to assume that the function that embeds the vertices in \mathbb{R}^d is smooth over the graph. Therefore, the bottom k eigenvectors of the graph's Laplacian provide a good sparse basis for the position vectors. Typical values of k range from 10-30 depending on the size of the graph. Let $Q \in \mathbb{R}^{n \times k}$ be the matrix composed of the eigenvectors of the Laplacian corresponding to the k smallest eigenvalues, ignoring the eigenvalue 0. It is well known that the eigenvector corresponding to eigenvalue 0 accounts for the center of mass of the function. Removing it from consideration automatically places the embedding at the origin. We can then find k vectors y_l in \mathbb{R}^k so that we can write each x_i as $\sum_l q_{il} y_l$ where $q_i = (q_{i1}, q_{i2}, \dots, q_{ik})$ is the i th row of Q . If we then define the $k \times k$ positive semi-definite matrix $Y = (y_{ij})$ where $y_{ij} = y_i \cdot y_j$, we have

$$X = PP^T = QYQ^T.$$

Using $\mathcal{X} = \text{vec}(X)$ and letting $\mathcal{Y} = \text{vec}(Y)$, we can rewrite the above as

$$\mathcal{X} = (Q \otimes Q) \mathcal{Y},$$

where \otimes is the Kronecker product. Using this in expression (5), we have

$$T_1 = \sum_{\{i,j\} \in E} w_{ij}^2 \{ \text{vec}(E_{ij})^T (Q \otimes Q) \mathcal{Y} - d_{ij}^2 \}^2. \tag{6}$$

Since $x_i - x_j = \sum_l (q_{il} - q_{jl}) y_l$, it is fairly straightforward to see that the following holds:

$$\text{vec}(E_{ij})^T (Q \otimes Q) = (q_i - q_j) \otimes (q_i - q_j).$$

Applying this to equation (6), we have

$$\begin{aligned} T_1 &= \sum_{\{i,j\} \in E} w_{ij}^2 \{ (q_i - q_j) \otimes (q_i - q_j) \mathcal{Y} - d_{ij}^2 \}^2 \\ &= \sum_{\{i,j\} \in E} w_{ij}^2 \mathcal{Y}^T [((q_i - q_j) \otimes (q_i - q_j))^T ((q_i - q_j) \otimes (q_j - q_i))] \mathcal{Y} - \\ &\quad 2 \sum_{\{i,j\} \in E} w_{ij}^2 d_{ij}^2 ((q_i - q_j) \otimes (q_i - q_j)) \mathcal{Y} + \sum_{\{i,j\} \in E} w_{ij}^2 d_{ij}^4. \end{aligned}$$

Now, turning to the second term of $T(P)$, we have

$$\begin{aligned} T_2 &= t\lambda \sum_{(i,j) \notin E} \|x_i - x_j\|^2 \\ &= t\lambda \left\{ \sum_{i>j} \|x_i - x_j\|^2 - \sum_{\{i,j\} \in E} \|x_i - x_j\|^2 \right\}. \tag{7} \end{aligned}$$

Lemma 1. $\sum_{i>j} \|x_i - x_j\|^2 = n\text{Tr}(Y)$ and $\sum_{\{i,j\} \in E} \|x_i - x_j\|^2 = ((q_i - q_j) \otimes (q_i - q_j))\mathcal{Y}$.

Proof. Because the x_i have zero mean, the first summation is equal to $n\sum_i \|x_i\|^2 = n\text{Tr}(X) = n\text{Tr}(Y)$. \square

Using lemma 1, we can rewrite equation (7) as

$$\begin{aligned} T_2 &= t\lambda \{n\text{Tr}(Y) - ((q_i - q_j) \otimes (q_i - q_j))\mathcal{Y}\} \\ &= t\lambda \{n\text{vec}(I)^T - \sum_{\{i,j\} \in E} ((q_i - q_j) \otimes (q_i - q_j))\}\mathcal{Y}. \end{aligned}$$

Combining our recastings of the two terms of equation (3), we have:

$$\begin{aligned} T(P) &= T_1 - T_2 \\ &= \mathcal{Y}^T \left[\sum_{\{i,j\} \in E} w_{ij}^2 \{((q_i - q_j) \otimes (q_i - q_j))^T ((q_i - q_j) \otimes (q_i - q_j))\} \right] \mathcal{Y} - \\ &\quad \left[\sum_{\{i,j\} \in E} (2w_{ij}^2 d_{ij}^2 - t\lambda) ((q_i - q_j) \otimes (q_i - q_j)) - nt\lambda \text{vec}(I)^T \right] \mathcal{Y} + \\ &\quad \sum_{\{i,j\} \in E} w_{ij}^2 d_{ij}^4. \end{aligned}$$

To simplify the exposition, we can write $T(P)$ as $\mathcal{Y}^T \mathbf{A} \mathcal{Y} + \mathbf{b}^T \mathcal{Y} + \text{constant}$. Since A and Y are symmetric positive semi-definite matrices, this is a convex function inside the semi-definite cone. It can be solved easily by any off-the-shelf semi-definite program (SDP). SDP is usually inefficient, taking cubic time in the size of the variables and constraints. A key novelty in our approach is the use of the approximation using the graph Laplacian. Instead of minimizing with n^2 variables, our re-parameterization with Y reduces the number of variables to k^2 . This is usually constant for most graphs and hence makes our approach scalable. Because of the special structure of our problem, we can further improve the running time by converting our quadratically-constrained SDP to a Semidefinite Quadratic Linear Program (SQLP) and use a specialized solver like SDPT3 [25]. Details of this conversion are given in the report [11].

4 Experimental Results

We implemented the COAST algorithm in a combination of Python, Matlab and C code. The main parts consist of forming the matrix A and vector b , calculating the eigenvectors of the Laplacian, and solving the optimization problem. Time for the last part is dependent only on the number of eigenvectors k , hence is constant for a fixed number of eigenvectors. For graphs of size up to 100,000, the minimization using SQLP takes less than 10 seconds inside Matlab.

We tested the COAST algorithm for solving the quartic stress model on a range of graphs. For comparison, we also tested PivotMDS; PivotMDS(1), which uses PivotMDS, followed by a sparse stress majorization; the maxent-stress model Maxent; and

Table 1. Algorithms tested

Algorithm	Model	Fits distances?
COAST	quartic stress model	Yes. Edges only
PivotMDS	approx. strain model	Yes/No
PivotMDS(1)	PivotMDS + sparse stress	Yes.
Maxent	PivotMDS + maxent-stress	Yes.
FSM	full stress model	Yes. All-pairs

the full stress model, using stress majorization. We summarize all the tested algorithms in Table 1.

With the exception of graph `gd`, which is an author collaboration graph of the International Symposium on Graph Drawing between 1994-2007, the graphs used are from the University of Florida Sparse Matrix Collection [5]. Our selection is exactly the same as that used by Gansner et al. [12]. Two of the graphs (`commanche` and `luxembourg`) have associated pre-defined non-unit edge lengths. In our study, a rectangular matrix, or one with an asymmetric pattern, is treated as a bipartite graph. Test graph sizes are given in Table 2.

Table 2. Test graphs. Graphs marked * have pre-specified non-unit edge lengths. Otherwise, unit edge length is assumed.

Graph	$ V $	$ E $	description
<code>gd</code>	464	1311	Collaboration graph
<code>btree</code>	1023	1022	Binary tree
<code>1138_bus</code>	1138	1358	Power system
<code>qh882</code>	1764	3354	Quebec hydro power
<code>lp_ship041</code>	2526	6380	Linear programming
<code>USpowerGrid</code>	4941	6594	US power grid
<code>commanche*</code>	7920	11880	Helicopter
<code>bcsstk31</code>	35586	572913	Automobile component
<code>luxembourg*</code>	114599	119666	Luxembourg street map

Tables 3 and 4 present the outcomes for two graphs. (The drawings for all of the graphs tested, with additional color detail, can be found in the companion report [11].) Following Brandes and Pich [2], each drawing has an associated error chart. In an error chart, the x -axis gives the graph distance bins, the y -axis is the difference between the actual geometric distance in the layout and the graph distance. The chart shows the median (black line), the 25 and 75 percentiles (gray band) and the min/max errors (gray lines) that fall within each bin. For ease of understanding, we plot graph distance against distance error, instead of graph distance vs. actual distance as suggested by Brandes and Pich [2]. Because generating the error chart requires an all-pairs shortest paths calculation, we provide this chart only for graphs with less than 10,000 nodes.

Table 3. Drawings and error charts of the tested algorithms for `btree`

PivotMDS	PivotMDS(1)	Maxent	COAST	FSM

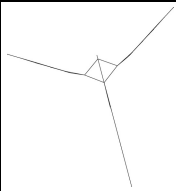
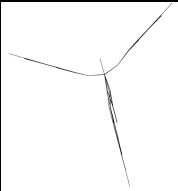


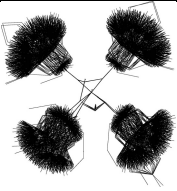
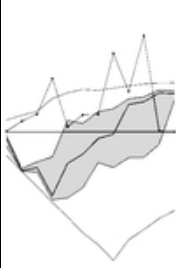
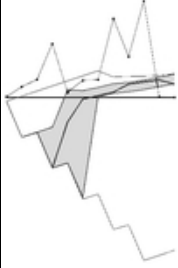


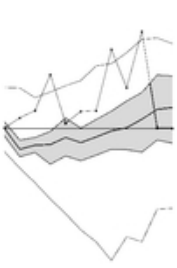
With the error chart, we also include a graph distance distribution curve (line with dots), representing the number of vertex pairs in each graph distance bin. This distribution depends on the graph, and is independent of the drawing. In making the error charts, the layout is scaled to minimize the full stress model (1), with $w_{ij} = 1/d_{ij}^2$.

As an example, the error chart for PivotMDS on `btree` (Table 3, column 1, bottom) shows that, on average, the median line is under the x -axis for small graph distances. This means that the PivotMDS layout under-represents the graph distance between vertex pairs that are a few hops away. This is because it collapses branches of tree-like structures. The leaves of such structures tend to be a few hops away, but are now positioned very near to each other. To some extent the same under-representation of graph distance for vertex pairs that are a few hops away is seen for PivotMDS and PivotMDS(1) on other non-rigid graphs, including `1138_bus`, `btree`, `lp_ship041` and `USpowerGrid`. Compared with PivotMDS and PivotMDS(1), the median line for Maxent (column 3) does not undershoot the x -axis as much.

Comparing the COAST layouts with the others, we note that it appears to track the x -axis more tightly and uniformly than the others, except for large lengths where, in certain cases, it dives significantly. In general, COAST has a more consistent bias for under-representation than the other layouts. The others tend to under-represent short lengths and over-represent long lengths. Visually, most of the COAST layouts are satisfactory, certainly avoiding the limitations of PivotMDS. For example, although it does not capture the symmetry of `btree` as well as Maxent, it does a better job of handling the details.

While visually comparing drawings made by different algorithms is informative, and may give an overall impression of the characteristics of each algorithm, such inspection is subjective. Ideally we would prefer to rely on a quantitative measure of performance. However such a measure is not easy to devise. For example, if we use sparse stress as our measure, PivotMDS, which minimizes sparse stress, is likely to come out best,

Table 4. Drawings and error charts of the tested algorithms for lp_ship041

PivotMDS	PivotMDS(1)	Maxent	COAST	FSM
				
				

despite its shortcomings. As a compromise, we propose to measure full stress, as defined by (1), with $w_{ij} = 1/d_{ij}^2$. Bear in mind that this measure naturally favors the full stress model.

Table 5 gives the full stress measure achieved by each algorithm, as well as the corresponding timings. Because it is expensive to calculate all-pairs shortest paths, we restrict experimental measurement to graphs with less than 10,000 nodes. From the table we can see that, as expected, FSM is the best, because it tries to optimize this measure. We note that COAST is mostly competitive with the other non-FSM layouts.

As for timings, COAST, although a hybrid implementation, is comparable with Maxent, and appears to work well on large graphs.

4.1 Measuring Precision of Neighborhood Preservation

Sometimes, in embedding high dimensional data into a lower dimension, one is interested in preserving the neighborhood structure. In such a situation, exact replication of distances between objects becomes a secondary concern.

For example, imagine a graph where each node is a movie. Based on some recommender algorithm, an edge is added between two movies if the algorithm predicts that a user who likes one movie would also like the other, with the length of the edge defined as the distance (dissimilarity) between the two movies. The graph is sparse because only movies that are strongly similar are connected by an edge. For a visualization of this data to be helpful, we need to embed this graph in 2D in such a way that, for each node (movie), nodes in its neighborhood in the layout are very likely to be similar to this node. This would allow the user to explore movies that are more likely of interest to her by examining, in the visualization, the neighborhoods of the movies she knew and liked.

Table 5. Full stress measure ($\times 1000$) and CPU time (in seconds) for PivotMDS, PivotMDS(1), Maxent, COAST and FSM. Smaller is better. We limit the measurements to graphs with less than 10,000 nodes and 10 hours of CPU time. A “-” is used to denote these missing data points.

Graph	PivotMDS		PivotMDS(1)		Maxent		COAST		FSM	
gd	19	0.3	15	0.3	12	0.8	13	4.6	10	2.3
btree	130	1.1	110	1.1	64	2.7	89	0.4	60	10.0
1138_bus	78	0.1	67	0.2	45	2.1	58	3.4	40	16.0
qh882	147	0.1	120	0.3	103	2.2	184	2.7	84	39.0
lp_ship041	667	0.1	769	0.1	363	2.2	368	5.0	251	58.0
USpowerGrid	1124	0.1	932	0.9	1018	6.5	1073	5.3	702	272.0
commanche	2305	0.2	1547	0.9	1545	9.0	2853	8.8	654	1025.0
bcsstk31	-	2.4	-	21.6	-	102.0	-	226.7	-	-
luxembourg	-	2.4	-	630.0	-	209.0	-	128.9	-	-

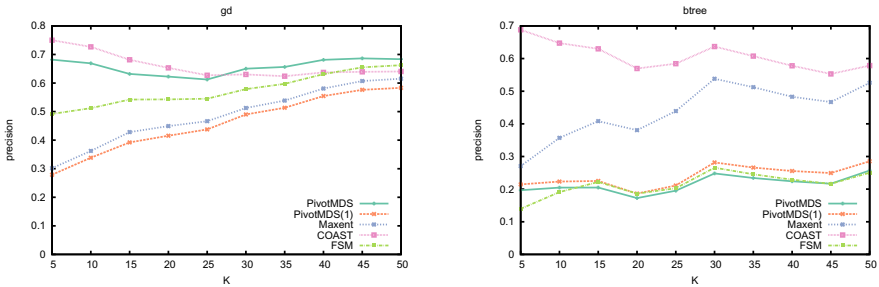


Fig. 1. Precision of neighborhood preservation of the algorithms, as a function of K . The higher the precision, the better.

Following Gansner et al. [12], we look at the *precision of neighborhood preservation*. We are interested in answering the question: if we see vertices nearby in the embedding, how many of these are actually also neighbors in the graph space? We define the precision of neighborhood preservation as follows. For each vertex i , K neighboring vertices of i in the layout are chosen. These K vertices are then checked to see if their graph distance is less than a threshold $d(K)$, where $d(K)$ is the distance of the K -th closest vertex to i in the graph space. The percentage of the K vertices that are within the threshold, averaged over all vertices i , is taken as the precision. Note that precision (the fraction of retrieved instances that are relevant) is a well-known concept in information retrieval. Chen and Buja [3] use a similar concept called *LC meta-criteria*.

Figure 1 gives the precision as a function of K for two representative graphs. (Figures for the remaining are available in the report [11].) From the figure, it is seen that, in general, COAST has the highest, or nearly the highest, precision. PivotMDS(1) tends to have low precision. The precision of other algorithms, including Maxent, tends to be between these two extremes.

Overall, precision of neighborhood preservation is a way to look at one aspect of embedding not well-captured by the full stress objective function, but is important to applications such as recommendations. COAST performs well in this respect.

5 Conclusion and Future Work

In this paper, we described a new technique for graph layout that attempts to satisfy edge length constraints. This technique uses a modified two-part stress function, one part for the edge lengths, the other to guide the relative placement of other node pairs. The stress is quartic in the positions of the nodes, and can be transformed to a form that is suitable for solution using convex programming. The results produced are good and the algorithm is scalable to large graphs.

Although the performance of the COAST algorithm is already competitive, we rely on an *ad hoc* implementation using a combination of Python, Matlab and C code. It would be very desirable to re-implement the algorithm entirely in C.

Our technique follows the general strategy of doing length-sensitive drawings for large graphs by reformulating the energy function, keeping the core length constraints, and then applying some appropriate mathematical machinery. Variations of this technique have been successfully used by others [17,12]. It would be interesting to explore additional adaptations of this approach.

Acknowledgements. We would like to thank the reviewers for their helpful comments.

References

1. Brandes, U., Pich, C.: Eigensolver methods for progressive multidimensional scaling of large data. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 42–53. Springer, Heidelberg (2007)
2. Brandes, U., Pich, C.: An experimental study on distance-based graph drawing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 218–229. Springer, Heidelberg (2009)
3. Chen, L., Buja, A.: Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *J. Amer. Statistical Assoc.* 104, 209–219 (2009)
4. Chung, F.R.K.: *Spectral Graph Theory* (CBMS Regional Conference Series in Mathematics, No. 92). American Mathematical Society, Providence (1996)
5. Davis, T.A., Hu, Y.: U. of Florida Sparse Matrix Collection. *ACM Transaction on Mathematical Software* 38, 1–18 (2011), <http://www.cise.ufl.edu/research/sparse/matrices/>
6. Drineas, P., Frieze, A.M., Kannan, R., Vempala, S., Vinay, V.: Clustering large graphs via the singular value decomposition. *Machine Learning* 56, 9–33 (2004)
7. Eades, P.: A heuristic for graph drawing. *Congressus Numerantium* 42, 149–160 (1984)
8. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force directed placement. *Software - Practice and Experience* 21, 1129–1164 (1991)
9. Gajer, P., Goodrich, M.T., Kobourov, S.G.: A multi-dimensional approach to force-directed layouts of large graphs. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 211–221. Springer, Heidelberg (2001)

10. Gansner, E.R., Koren, Y., North, S.C.: Graph drawing by stress majorization. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
11. Gansner, E.R., Hu, Y., Krishnan, S.: COAST: A convex optimization approach to stress-based embedding (2013), <http://arxiv.org/abs/1308.5218>
12. Gansner, E.R., Hu, Y., North, S.C.: A maxent-stress model for graph layout. *IEEE Trans. Vis. Comput. Graph.* 19(6), 927–940 (2013)
13. Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 285–295. Springer, Heidelberg (2005)
14. Hu, Y.: Efficient and high quality force-directed graph drawing. *Mathematica Journal* 10, 37–71 (2005)
15. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. *J. ACM* 24(1), 1–13 (1977)
16. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. *Information Processing Letters* 31, 7–15 (1989)
17. Houry, M., Hu, Y., Krishnan, S., Scheidegger, C.: Drawing large graphs by low-rank stress majorization. *Computer Graphics Forum* 31(3), 975–984 (2012)
18. Koren, Y., Çivril, A.: The binary stress model for graph drawing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 193–205. Springer, Heidelberg (2009)
19. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29, 1–27 (1964)
20. Kruskal, J.B., Seery, J.B.: Designing network diagrams. In: Proc. First General Conference on Social Graphics, pp. 22–50. U. S. Department of the Census, Washington, D.C. (July 1980), Bell Laboratories Technical Report No. 49
21. Noack, A.: Energy models for graph clustering. *J. Graph Algorithms and Applications* 11(2), 453–480 (2007)
22. Noack, A.: Modularity clustering is force-directed layout. *Physical Review E* 79, 026102 (2009)
23. Pettie, S.: A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science* 312(1), 47–74 (2004)
24. de Silva, V., Tenenbaum, J.B.: Global versus local methods in nonlinear dimensionality reduction. In: *Advances in Neural Information Processing Systems* 15, pp. 721–728. MIT Press, Cambridge (2003)
25. Tütüncü, R.H., Toh, K.C., Todd, M.J.: Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming* 95(2), 189–217 (2003)
26. Walshaw, C.: A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms and Applications* 7, 253–285 (2003)

Colored Spanning Graphs for Set Visualization*

Ferran Hurtado¹, Matias Korman¹, Marc van Kreveld², Maarten Löffler²,
Vera Sacristán¹, Rodrigo I. Silveira^{4,1}, and Bettina Speckmann³

¹ Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, Spain
{ferran.hurtado,matias.korman,vera.sacristan}@upc.edu

² Dept. of Computing and Information Sciences, Utrecht University, the Netherlands
{m.loffler,m.j.vankreveld}@uu.nl

³ Dept. of Mathematics and Computer Science,
Technical University Eindhoven, The Netherlands
speckman@win.tue.nl

⁴ Dept. de Matemática, Universidade de Aveiro, Portugal
rodrigo.silveira@ua.pt

Abstract. We study an algorithmic problem that is motivated by ink minimization for sparse set visualizations. Our input is a set of points in the plane which are either blue, red, or purple. Blue points belong exclusively to the blue set, red points belong exclusively to the red set, and purple points belong to both sets. A *red-blue-purple spanning graph* (RBP spanning graph) is a set of edges connecting the points such that the subgraph induced by the red and purple points is connected, and the subgraph induced by the blue and purple points is connected.

We study the geometric properties of minimum RBP spanning graphs and the algorithmic problems associated with computing them. Specifically, we show that the general problem is NP-hard. Hence we give an $(\frac{1}{2}\rho+1)$ -approximation, where ρ is the Steiner ratio. We also present efficient exact solutions if the points are located on a line or a circle. Finally we consider extensions to more than two sets.

1 Introduction

Visualizing sets and their elements is a recurring theme in information visualization. Sets arise in many application areas, as varied as social network analysis (grouping individuals into communities), linguistics (related words), or geography (related places). Among the oldest representations for sets are Venn diagrams [11] and their generalization, Euler diagrams. These representations are natural and effective for a small number of elements and sets. However, for larger numbers of sets and more complicated intersection patterns more intricate solutions are necessary. The last years have seen a steady

* M.L. was supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123. F. H., M. K., V. S. and R.I. S. were partially supported by ESF EURO-CORES programme EuroGIGA, CRP ComPoSe: grant EUI-EURC-2011-4306, and by project MINECO MTM2012-30951. F. H., V. S. and R.I. S. were supported by project Gen. Cat. DGR 2009SGR1040. M. K. was supported by the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union. R. I. S. was funded by the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235 and by FCT through grant SFRH/BPD/88455/2012.

stream of developments in this direction, both for the situation where the location of set elements can be freely chosen and for the important special case that elements have to be drawn at particular fixed positions (for example, restaurant locations on a city map).

Our paper is motivated by some recent approaches that use very sparse enclosing shapes when depicting sets. LineSets [3] are the most minimal of all, reducing the geometry to a single continuous line per set which connects all elements. Both Kelp Diagrams [10] and its successor KelpFusion [15] are based on sparse spanning graphs, essentially variations of minimal spanning trees for different distance measures. These methods attempt to reduce visual clutter by reducing the amount of “ink” (see Tufte’s rule [23]) necessary to connect all elements of a set. However, although the results are visually pleasing, neither method does use the optimal amount of ink. In this paper we explore the algorithmic questions that arise when computing spanning graphs for set visualization which are optimal with respect to ink usage.

Problem Statement. Our input is a set of n points in the plane. Each point is a member of one or more sets. We mostly consider the case where there are exactly two sets, namely a red and a blue set. A point is red if it is part only of the red set and it is blue if it is part only of the blue set. A point that is part of both the red and the blue set is purple.

A *red-blue-purple spanning graph* (RBP spanning graph) for a set of points that are red, blue and purple is a set of edges connecting the points such that the subgraph induced by the red and purple points is connected, and the subgraph induced by the blue and purple points is connected. A *minimum RBP spanning graph* for a set of points that are red, blue and purple is a red-blue-purple spanning graph that has minimum weight (total edge length) (see Figure 1). In this paper we consider the algorithmic problems associated with computing minimum RBP spanning graphs.

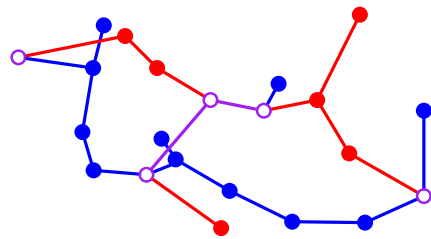


Fig. 1. A minimum RBP spanning graph

Results and Organization. We first review related work. In Section 2 we describe and prove various useful properties of (minimum) RBP spanning graphs. Then, in Section 3, we consider the two special cases where the points are located on a line or on a circle. This setting is meaningful if the elements of the sets are not associated with a specific location (for example, social networks or software systems). Here visualizations frequently choose to arrange elements in simple configurations such as lines or circles. We give an $O(n)$ time algorithm for points on a line, assuming that the input is already sorted. For points on a circle we exploit a structural result which allows us to use dynamic programming in $O(k^3 + n)$ time, where k is the number of purple points. In Section 4 we prove that computing a minimum RBP spanning graph is NP-hard in general. Hence, in Section 5 we turn to approximations. We describe an $O(n \log n)$ algorithm that computes a $(\frac{1}{2}\rho + 1)$ -approximation of the minimum RBP spanning graph, where ρ is the Steiner ratio. Finally, in Section 6 we discuss various extensions for situations with more than two sets. Due to space constraints some proofs have been deferred to the full version.

Related Work. In recent years a number of papers explored the problem of automatically drawing Euler diagrams, for example, Simonetto and Auber [20], Stapleton *et al.* [21], and Henry Riche and Dwyer [13]. These methods assume that the locations of the set elements can be freely chosen. An important variation is the case that elements have to be drawn at fixed positions. Collins *et al.* [9] presented *Bubble Sets*, a method based on isocontours. A similar approach was suggested by Byelas and Telea [7]. *LineSets* by Alper *et al.* [3] attempt to improve the overall readability by the minimalist approach of drawing a single line per set. Dinkla *et al.* [10] introduced *Kelp Diagrams* which use a sparse spanning graph, essentially a minimum spanning tree with some additional edges. Kelp Diagrams were extended by Meulemans *et al.* [15] to a hybrid technique named *KelpFusion* which uses a mix of hulls and lines to generate fitted boundaries.

Sets defined over points in the plane can be interpreted as an embedding of a *hypergraph* where the points are vertices and each set is a hyperedge connecting an arbitrary number of vertices. Drawings of hypergraphs have been discussed in several papers (e.g., see Brandes *et al.* [5] and references therein).

Also in the area of discrete and computational geometry many problems on colored point sets have been studied. Possibly the most famous example is the *Ham-Sandwich Theorem*: given a set of $2n$ red points and $2m$ blue points in general position in the plane, there is always a line ℓ such that each open halfplane bounded by ℓ contains exactly n red points and m blue points. There have been many variations on this theorem and also many other results on finding configurations or geometric graphs with constraints depending on colors. We refer the interested reader to the survey [14] and to Chapter 8 in the collection of research problems [6].

From an algorithmic point of view, many problems have been considered, here we mention only a few of them. The *bichromatic closest pair* (e.g., see Preparata and Shamos [19] Section 5.7), the *chromatic nearest neighbor search* (see Mount *et al.* [18]), the problems on finding *smallest color-spanning objects* (see Abellanas *et al.* [1]), the *colored range searching* problems (see Agarwal *et al.* [2]), and the *group Steiner tree* problem where, for a graph with colored vertices, the objective is to find a minimum weight subtree that covers all colors (see Mitchell [16], Section 7.1). Finally, Tokunaga [22] considers a set of red or blue points in the plane and computes two geometric spanning trees of the blue and the red points such that they intersect in as few points as possible.

2 Properties of RBP Spanning Graphs

We call an edge of a RBP spanning graph *red* if it connects two red points or a red and a purple point. We call an edge *blue* if it connects two blue points or a blue and a purple point. We call an edge *purple* if it connects two purple points. A minimum RBP spanning graph does not contain edges between a red and a blue point. The subgraph induced by the red and purple points in a minimum RBP spanning graph is a spanning tree (and the analogous statement holds for the blue and purple points). Figure 2 (a) illustrates the trade-off between red, blue, and purple edges in a minimum RBP spanning graph.

Every red edge in a minimum RBP spanning graph also occurs in a minimum spanning tree of only the red and purple points. The corresponding statement is true also

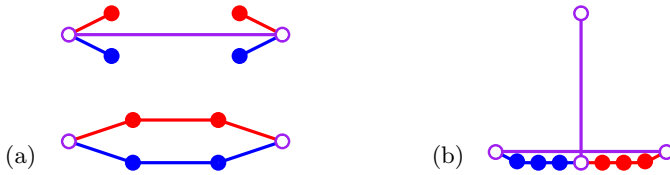


Fig. 2. (a) Two examples of minimum RBP spanning graphs of similar configurations of points. (b) A minimum RBP spanning graph with a purple edge crossing.

for the blue edges, but not for the purple edges. That is, there can be purple edges in a minimum RBP spanning graph which do not occur in a minimum spanning tree of only the purple points.

It is easy to see that a minimum RBP spanning graph is not necessarily planar. Red and blue edges are mostly independent and they can easily cross. Moreover, a red and a purple edge can cross, a blue and a purple edge can cross, and even two purple edges can cross in a minimum RBP spanning graph, as shown in Figure 2 (b). In fact, a single purple edge can cross arbitrarily many purple edges. The intricate construction and the additional observations necessary to prove this are relegated to the full version.

Lemma 1. *A purple edge in an optimal RBP spanning graph can cross $\Theta(n)$ other purple edges.*

Just as with standard minimum spanning trees the degree of the points in a minimum RBP spanning graph is bounded.

Lemma 2. *The maximum degree of a point in a minimum RBP spanning graph is at most 18.*

This bound can be attained by a purple point p : Let p be the center of a regular hexagon with radius 3 and two more regular hexagons with radius 1, one slightly rotated. Place a purple point on each corner of the large hexagon, place red points on the corners of one of the smaller hexagons, and blue points on the corners of the other one. Then the star graph with p at the center is a minimum RBP spanning graph. A similar construction shows that there is a point set such that the unique minimum RBP spanning graph requires a point of degree 15; a higher degree is never necessary. A red or blue point can have degree at most 6, degree 5 is the highest degree that can be enforced.

3 Points on a Line or on a Circle

Here we describe efficient algorithms to find a minimum RBP spanning graph if the points lie on a line or on a circle. In the circle case, we first present additional geometric observations which allow us to use dynamic programming.

3.1 Points on a Line

Given a problem instance S , we number the purple points p_1, \dots, p_k from left to right. For any $1 \leq i \leq k - 1$, let S_i be the set of points between p_i and p_{i+1} (including both

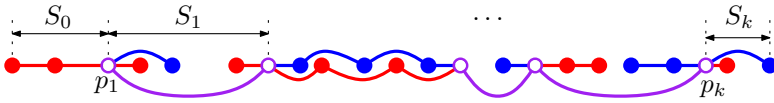


Fig. 3. A minimum RBP spanning graph of points on a line, and its partition into sets S_i (some edges are depicted by curved arcs for clarity).

p_i and p_{i+1}). We also define S_0 to contain p_1 and all red/blue points to its left, and S_k to contain p_k and all red/blue points to its right (see Figure 3). First, we show that each subset can be treated independently.

Lemma 3. *Let S be a set of red, blue and purple points located on a line, and let G^* be a minimum RBP spanning graph of S . Then for any edge $qq' \in G^*$, the points q and q' are contained in S_j , for some j .*

Using this lemma it is straightforward to obtain an efficient algorithm.

Theorem 1. *Let S be a set of n red, blue and purple points located on a line. We can compute a minimum RBP spanning graph of S in $O(n)$ time, provided that the points are sorted along the line.*

3.2 Points on a Circle

We proved in Lemma 1 that for points in general position a purple edge can cross many other purple edges. Even if the points lie in convex position, purple edges can cross each other (see Figure 2 (right)). But below we prove that for points on a circle the situation is structurally different and purple edges cannot cross any other edges.

Lemma 4. *Let S be a set of red, blue and purple points located on a circle. A minimum RBP spanning graph of S cannot have a purple edge crossing any other edge.*

Proof. Let G be a minimum RBP spanning graph in which two edges $e_1 = vv'$ and $e_2 = ww'$ cross. We will perform a local transformation that will reduce the weight of G , contradicting the minimality of G .

First assume that both e_1 and e_2 are purple, and consider the four red paths in G that start at either v or v' and end at w or w' . Without loss of generality, we can assume that the minimum-link path among the four (i.e., the path with smallest number of edges) connects v and w . Let π_R be such path; note that π_R cannot use edge vv' nor edge ww' . Likewise, let π_B be the minimum-link blue path among those that connect v or v' with w or w' . We now distinguish three cases depending on the number of shared endpoints between π_R and π_B (see Figure 4):

π_R and π_B share both endpoints. We replace edges vv' and ww' with edges vw' and $v'w$. The resulting graph G' clearly has smaller weight than G . We now prove that G' is indeed spanning. First consider the red tree in G' : the removal of edge ww' created two components. Moreover, points v and w' must belong to different components (otherwise, the edge ww' would create a cycle in G). Thus, by adding

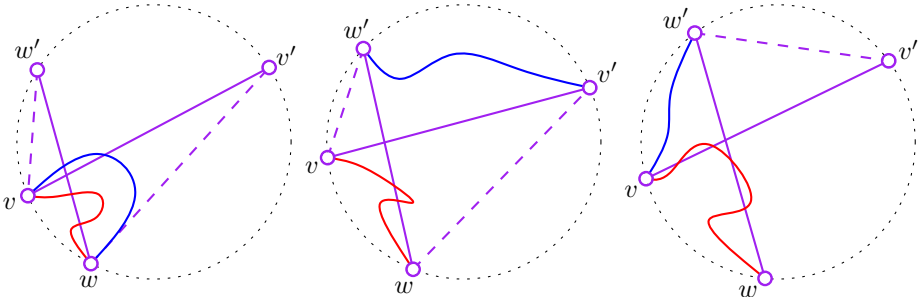


Fig. 4. The three cases in the proof of Lemma 4, local transformations are shown by dashed purple edges. No assumptions are made on the number of crossings between π_R , π_B , vv' , and ww' .

the edge vw' we reconnect the two components again. Likewise, the removal of edge vv' creates two red components that are reconnected with the edge ww' . That is, graph G' also spans red. We repeat the same reasoning for blue and obtain that G' is a RBP spanning graph with smaller weight than G , a contradiction.

π_R and π_B share no endpoints. We proceed as in the previous case, replacing edges vv' and ww' by vw and $v'w$. The argumentation is identical to the previous case.

π_R and π_B share one endpoint. We can assume that v is the shared endpoint, and that the other endpoint of π_B is w' (see Figure 4, right). In this case, both red and blue paths from v' to both w and w' in G use the edge vv' . Then we can replace this edge by either $v'w$ or $v'w'$ and maintain the spanning property. Using the fact that the four vertices are on the boundary of a circle, it is easy to see that either $\|v'w\| < \|v'v\|$ or $\|v'w'\| < \|v'v\|$ must hold, thus one of the two resulting graphs will have smaller weight.

If one of the edges is not purple the situation is easier, since we need to consider only one color. We assume that the edge e_2 is red, and that the path from v to w does not use e_1 nor e_2 . Then we can replace the edge ww' by either vw' or $v'w'$ to obtain a graph of smaller weight. Note that, since we are changing a red edge, the spanning property of blue cannot be altered, and the lemma is shown. \square

Next we present another crossing property that will be useful for our algorithm.

Corollary 1. *Let S be a set of red, blue and purple points located on the boundary of a circle. In a minimum RBP spanning graph G of S , no red or blue edge of G can cross a segment between two purple points.*

Proof. Let p, p' be two purple points, and assume that a red edge $rr' \in G$ crosses the segment pp' . As in the proof of Lemma 4, we can assume that the red path from p to r does not pass through neither p' nor r' . Then, we replace the edge rr' by either $r'p$ or $r'p'$ to obtain a RBP spanning graph of smaller weight. \square

Using these geometric observations and a few others proved in the full version we can now compute a minimum RBP spanning graph with dynamic programming.

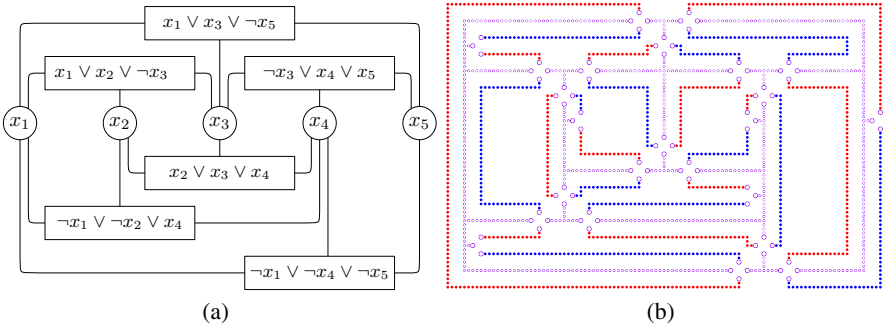


Fig. 5. (a) A planar 3-SAT formula. (b) The corresponding set of red, blue, and purple points.

Theorem 2. Let S be a set of n red, blue and purple points located on a circle. We can compute a minimum RBP spanning graph of S in $O(k^3 + n)$ time, where k is the number of purple points.

4 NP-hardness

Computing a minimum RBP spanning graph is NP-hard. We prove this by a reduction from planar 3-SAT. Figure 5 illustrates the global construction: an embedded 3-SAT formula, and the corresponding set of red, blue, and purple points that we construct. There is a value W such that a solution of weight less than W exists if and only if the 3-SAT formula is satisfiable.

The construction consists of two parts. First, we have a *variable gadget* for each variable x_i in the 3-SAT formula. Such a gadget consists of a purple *skeleton* and a red/blue *loop* around the skeleton. The loop consists of alternating densely-sampled blue and red chains, separated by a pair of purple points at distance $\sqrt{2}$ from each other. For each such pair of purple points there is also another purple point at distance 1 from both points, which is connected to the skeleton. We call such a group of 3 purple points a *switch*. Figure 6 (a) shows an example.

First, we observe that if the red, blue and purple chains are sampled sufficiently densely, then they will be connected in any optimal solution; hence, we ignore the costs of these connections from now on. There are exactly two ways to optimally connect the

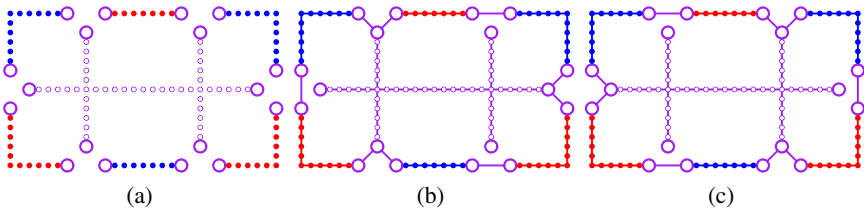


Fig. 6. (a) The input for a variable gadget. (b) One possible optimal solution of this construction, which represents the value *true*. (c) The other optimal solution represents the value *false*.

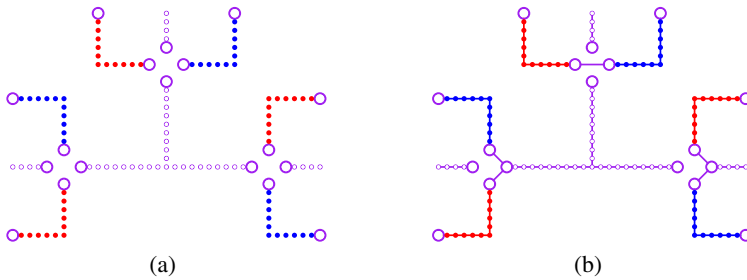


Fig. 7. (a) The input for a clause gadget. (b) One possible solution where the clause is satisfied. The left and top literals are in their *true* states; the right literal is in its *false* state.

remaining components: we alternately connect the purple points on opposite sides of a switch to each other, or both to the skeleton. Figures 6(b) and 6(c) illustrate the two solutions. These will correspond to the values *true* and *false* of the variable.

Next, we have *clause gadgets* for all clauses of the 3-SAT formula. These occur at places where the variable loops of the three involved variables get close to each other, and we make sure that there is a switch in each of them (from red to blue if the variable occurs positively in the clause, from blue to red otherwise). For these switches, we place a fourth point, also at distance 1 from both ends of the switch, and we connect these three extra points by a clause skeleton. Figure 7(a) shows an example.

Lemma 5. *There is a value W such that an RBP spanning graph of length less than W exists if and only if the 3-SAT formula is satisfiable.*

Proof. Suppose the planar 3-SAT formula has n variables and m clauses, and let k_i be the number of clauses that variable x_i appears in. There are $2k_i$ switches per variable. The total number of switches is $2 \sum_{i=1}^n k_i = 6m$. We ignore the red, blue and purple chains; we only argue about the total length of purple edges within the switches.

Within variable x_i , there is one skeleton, k_i blue pieces, and k_i red pieces, which means that there are $3k_i + 1$ components in either the red or the blue tree that need to be connected. For this, we need to add $3k_i$ edges. Within each switch, the purple points that are connected to the red or blue paths must certainly get a purple edge. Since we add only $3k_i$ edges, half of the switches get only one edge; these edges must then connect the blue-path purple point to the red-path purple point and have length $\sqrt{2}$. The other half of the switches are connected via the skeleton with two edges of length 1. So, the total length within a variable is at least $k_i\sqrt{2} + 2k_i$. This is also possible to achieve, as seen in Figure 6(b) and 6(c).

Now, the clause skeletons have to be connected to at least one of the variables. For each of them, we need one more edge in one of the switches. The cheapest possible way of doing this is by using the groups that had expensive edges (of length $\sqrt{2}$) and using two normal edges (total length 2) instead. So, globally, of the $6m$ switches, at least $4m$ need two edges and the ones with a single edge must have length at least $\sqrt{2}$. Thus, the total length must be at least $W = (8 + \sqrt{2})m$. We claim that a solution of this length exists if and only if the formula is satisfiable. \square

Theorem 3. *Computing a minimum RBP spanning graph is NP-hard.*

5 Approximation

A simple approximation algorithm determines the red-purple minimum spanning tree and the blue-purple minimum spanning tree, and takes the union of their edges. It is easy to see that this is a 2-approximation algorithm that requires $O(n \log n)$ time.

Another approximation algorithm, A , starts by computing the minimum spanning tree of the purple edges, and then adds the red and blue points in an optimal manner in the style of Kruskal’s algorithm for minimum spanning trees. Algorithm A can also be implemented to run in $O(n \log n)$ time by computing the Delaunay triangulation of the red and purple points and of the blue and purple points. It is easy to argue that A also is a 2-approximation algorithm but interestingly, we can prove a better bound (close to 1.6) by expressing the approximation factor in the Steiner ratio ρ . Gilbert and Pollak [12] conjectured that $\rho = \frac{2}{\sqrt{3}} \approx 1.15$, but this conjecture has not been proved yet.¹ Chung and Graham [8] showed a bound of ≈ 1.21 , which is the best-known upper bound on ρ .

Theorem 4. *Approximation algorithm A is a $(\frac{1}{2}\rho + 1)$ -approximation of the minimum RBP spanning graph, where ρ is the Steiner ratio. The approximation is not a c -approximation for any constant $c < 1 + \frac{1}{\sqrt{3}}$.*

Proof. Algorithm A is not a c -approximation for any $c < 1 + \frac{1}{\sqrt{3}}$ (see Figure 8). Hence our approximation analysis is tight if the Gilbert-Pollak conjecture is true.

Next we prove our claim on the approximation factor. Let R, B , and P be sets of red, blue, and purple points. Let G^* be their minimum RBP spanning graph. Let R^* be the red edges, B^* the blue edges, and P^* the purple edges in G^* . Let A be the algorithm that computes a spanning graph by taking the minimum spanning tree of the purple points, and then adding the red and blue points optimally. We denote the resulting graph on $R \cup B \cup P$ by G' , and its red, blue and purple edges by R', B' , and P' .

Suppose first that G^* has no purple edges. Then algorithm A gives extra length in terms of purple edges equal to the MST of the purple points, denoted $\|P'\|$. The optimal

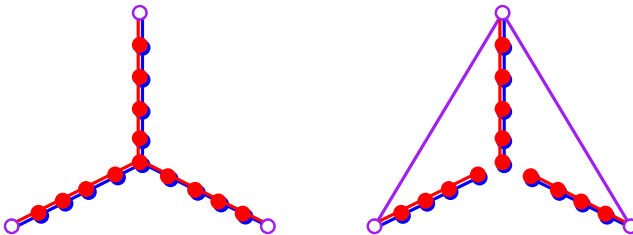


Fig. 8. A minimum RBP spanning graph and the RBP spanning graph on the same points obtained by approximation algorithm A

¹ A proof of the conjecture by Du and Hwang, “A proof of Gilbert-Pollak Conjecture on the Steiner ratio”, *Algorithmica* 7:121–135 (1992), turned out to be incorrect.

graph G^* must connect all purple points simultaneously through a red spanning tree and through a blue spanning tree whose lengths are $\|R^*\|$ and $\|B^*\|$. Algorithm A has a total length of red edges of $\|R'\| \leq \|R^*\|$ and a total length of blue edges of $\|B'\| \leq \|B^*\|$. Hence the approximation ratio of A in case of absence of purple edges in the optimal solution is

$$\frac{\|P'\| + \|R'\| + \|B'\|}{\|R^*\| + \|B^*\|} \leq \frac{\|P'\| + \|R^*\| + \|B^*\|}{\|R^*\| + \|B^*\|}.$$

This ratio is maximized when R lies very densely on the Steiner Minimum Tree of P , and the same is true for B . Due to the density, algorithm A will choose nearly the full length of the Steiner Minimum Tree of P as well, once for red and once for blue. The approximation ratio is then smaller than but arbitrarily close to

$$\frac{MST(P) + 2 \cdot SMT(P)}{2 \cdot SMT(P)} \leq \frac{\rho \cdot SMT(P) + 2 \cdot SMT(P)}{2 \cdot SMT(P)} = \frac{\rho + 2}{2} = \frac{1}{2}\rho + 1,$$

where $SMT(P)$ is the Steiner Minimum Tree of P (or its length) and $MST(P)$ is the Minimum Spanning tree of P (or its length).

Next, suppose that G^* has a set P^* of purple edges, and assume them fixed. We will reason about sets of red, blue and purple points for which the algorithm A performs as poorly as possible in terms of approximation ratio.

If G^* has any red point r that has a single red edge incident to it in R^* , then this edge will connect r to the closest red or purple point, otherwise G^* is not optimal. Algorithm A will choose exactly the same edge in its solution. Hence, the approximation ratio of A for the points $R \setminus \{r\}$, B , and P is higher than for the points R , B , and P . The same is true for a blue or purple point that has a single incident edge in G^* . So we can restrict ourselves to analyzing point sets whose optimal solution does not have any leafs in G^* .

Let $B@R$ be a set of blue points infinitesimally close to the locations of the red points, and let $R@B$ be a set of red points infinitesimally close to the locations of the blue points. Now we can compare the approximation ratio of A (i) on P , R , and B , (ii) on P , R , and $B@R$, and (iii) on P , $R@B$, and B , and notice that at least one of (ii) and (iii) gives an approximation ratio at least as high as for (i). Hence, we can restrict ourselves to analyzing point sets where the red and blue points lie at basically the same positions (but they are not purple points).

The edges of P^* partition the purple points of P into a number of purple components which are connected by a red spanning forest and a blue spanning forest. We have

$$\|P'\| \leq \|P^*\| + \rho \|R^*\|,$$

because in G^* the red (blue) connections between the purple components cannot be shorter than the Steiner Minimum Forest of the purple components. By the observations above we can assume that all red and blue points are used in the red and blue spanning forests that connect the purple components. Hence the approximation ratio is

$$\frac{\|P'\| + \|R'\| + \|B'\|}{\|P^*\| + \|R^*\| + \|B^*\|} \leq \frac{\|P'\| + 2\|R'\|}{\|P^*\| + 2\|R^*\|} \leq \frac{\|P^*\| + \rho\|R^*\| + 2\|R^*\|}{\|P^*\| + 2\|R^*\|}.$$

This ratio is maximized when $\|P^*\| = 0$, in which case we get exactly the same ratio as above, when no purple edges are present. □

It is possible that a PTAS exists for our problem, but it is not clear whether the techniques of Arora [4] or Mitchell [17] for the Euclidean traveling salesperson problem can be applied since RBP spanning graphs are not planar, and the number of crossings of a single edge can be large.

6 Extensions and Future Work

Beyond Purple. So far we considered the case where there are exactly two sets, the red set and the blue set, leading to an input with red, blue, and purple points. In general we might have k different sets, all denoted by primary colors. For instance, for $k = 3$ we could have red, blue and yellow sets, which leads to three secondary colors (purple, orange, green) and one tertiary color (black). The objective is again to minimize the total length of a multi-colored spanning graph which has the property that the subgraphs induced by the red, blue, and yellow sets are connected (see Figure 9 (a)).

This problem is clearly still NP-hard. The 2-approximation immediately generalizes to a 3-approximation (or a k -approximation for k primary colors). We can improve on this by incorporating our $(1 + \frac{1}{2}\rho)$ -approximation algorithm to obtain a $(2 + \frac{1}{2}\rho)$ -approximation for three sets, or more generally a $(\lceil \frac{1}{2}k \rceil + \lfloor \frac{1}{2}k \rfloor \frac{1}{2}\rho)$ -approximation for k sets. Interestingly, our algorithms for points on a line or on a circle are not straightforward to generalize; these problems remain open.

Line Drawings. Another extension is motivated by LineSets [3]. Returning for the moment to the setting with two sets (and red, blue, and purple points), we now wish to compute a minimum RBP spanning graph such that the subgraphs induced by the red and blue sets are paths. That is, the red and purple edges form a path connecting all red and purple points, and the blue and purple edges form a path connecting all blue and purple points (see Figure 9 (b)).

This problem is NP-hard since TSP is hard. Nonetheless, we can obtain a $(2 + \varepsilon)$ -approximation by independently computing an approximate TSP for the blue and purple points and for the red and purple points, and simply taking the union. An approach similar to the spanning tree case seems to fail and hence a better solution remains an open problem. The question also remains open for points on a line or on a circle.

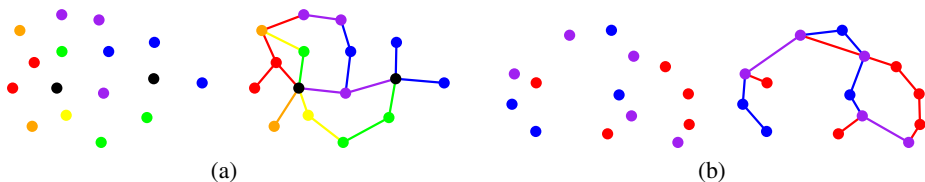


Fig. 9. (a) A set of multicolored points representing red, blue, and yellow sets and a corresponding spanning graph. (b) A set of red, blue and purple points, and a graph that connects all red points in a path and all blue points in a path.

References

1. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristán, V.: Smallest color-spanning objects. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 278–289. Springer, Heidelberg (2001)
2. Agarwal, P.K., Govindarajan, S., Muthukrishnan, S.M.: Range searching in categorical data: Colored range searching on grid. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 17–28. Springer, Heidelberg (2002)
3. Alper, B., Riche, N., Ramos, G., Czerwinski, M.: Design study of LineSets, a novel set visualization technique. *IEEE TVCG* 17(12), 2259–2267 (2011)
4. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM* 45(5), 753–782 (1998)
5. Brandes, U., Cornelsen, S., Pampel, B., Sallaberry, A.: Path-based supports for hypergraphs. *J. Discrete Algorithms* 14, 248–261 (2012)
6. Brass, P., Moser, W.O.J., Pach, J.: *Research Problems in Discrete Geometry*. Springer, New York (2005)
7. Byelas, H., Telea, A.: Towards realism in drawing areas of interest on architecture diagrams. *Visual Languages and Computing* 20(2), 110–128 (2009)
8. Chung, F., Graham, R.: A new bound for Euclidean Steiner minimum trees. *Ann. N.Y. Acad. Sci.* 440, 328–346 (1986)
9. Collins, C., Penn, G., Carpendale, S.: Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE TVCG* 15(6), 1009–1016 (2009)
10. Dinkla, K., van Kreveld, M., Speckmann, B., Westenberg, M.A.: Kelp Diagrams: Point set membership visualization. *Computer Graphics Forum* 31(3), 875–884 (2012)
11. Edwards, A.W.F.: *Cogwheels of the mind*. John Hopkins University Press (2004)
12. Gilbert, E., Pollak, H.: Steiner minimal trees. *SIAM J. Appl. Math.* 16, 1–29 (1968)
13. Henry Riche, N., Dwyer, T.: Untangling Euler diagrams. *IEEE TVCG* 16(6), 1090–1099 (2010)
14. Kaneko, A., Kano, M.: Discrete geometry on red and blue points in the plane – a survey. In: *Discrete and Comp. Geometry, The Goodman-Pollack Festschrift*, pp. 551–570 (2003)
15. Meulemans, W., Henry Riche, N., Speckmann, B., Alper, B., Dwyer, T.: KelpFusion: a hybrid set visualization technique. In: *IEEE TVCG* (to appear, 2013)
16. Mitchell, J.S.B.: Geometric shortest paths and network optimization. In: *Handbook of Computational Geometry*, pp. 633–701 (1998)
17. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.* 28(4), 1298–1309 (1999)
18. Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: Chromatic nearest neighbor searching: A query sensitive approach. *Computational Geometry: Theory and Applications* 17(3-4), 97–119 (2000)
19. Preparata, F.P., Shamos, M.I.: *Computational Geometry: An Introduction*. Springer, New York (1985)
20. Simonetto, P., Auber, D.: Visualise undrawable Euler diagrams. In: *Proc. 12th Conf. on Information Visualisation*, pp. 594–599 (2008)
21. Stapleton, G., Rodgers, P., Howse, J., Zhang, L.: Inductively generating Euler diagrams. *IEEE TVCG* 17(1), 88–100 (2011)
22. Tokunaga, S.: Intersection number of two connected geometric graphs. *Information Processing Letters* 59(6), 331–333 (1996)
23. Tufte, E.R.: *The Visual Display of Quantitative Information*. Graphics Press (1983)

Drawing Non-Planar Graphs with Crossing-Free Subgraphs*

Patrizio Angelini¹, Carla Binucci², Giordano Da Lozzo¹, Walter Didimo²,
Luca Grilli², Fabrizio Montecchiani², Maurizio Patrignani¹, and Ioannis G. Tollis³

¹ Università Roma Tre, Italy

{angelini,dalozzo,patrigna}@dia.uniroma3.it

² Università degli Studi di Perugia, Italy

{binucci,didimo,grilli,montecchiani}@diei.unipg.it

³ Univ. of Crete and Institute of Computer Science-FORTH, Greece

tollis@ics.forth.gr

Abstract. We initiate the study of the following problem: *Given a non-planar graph G and a planar subgraph S of G , does there exist a straight-line drawing Γ of G in the plane such that the edges of S are not crossed in Γ ?* We give positive and negative results for different kinds of spanning subgraphs S of G . Moreover, in order to enlarge the subset of instances that admit a solution, we consider the possibility of bending the edges of $G \setminus S$; in this setting different trade-offs between number of bends and drawing area are given.

1 Introduction

Lots of papers in graph drawing address the problem of computing drawings of non-planar graphs with the goal of mitigating the negative effect that edge crossings have on the drawing readability. Many of these papers describe crossing minimization methods, which are effective and computationally feasible for relatively small and sparse graphs (see [8] for a survey). Other papers study which non-planar graphs can be drawn such that the “crossing complexity” of the drawing is somewhat controlled, either in the number or in the type of crossings. They include the study of *k-planar drawings*, in which each edge is crossed at most k times (see, e.g., [7,11,12,15,16,20,24]), of *k-quasi planar drawings*, in which no k pairwise crossing edges exist (see, e.g., [1,2,10,23,26,28]), and of *large angle crossing drawings*, in which any two crossing edges form a sufficiently large angle (see [14] for a survey). Most of these drawings exist only for sparse graphs.

In this paper we initiate the study of a new graph drawing problem concerned with the drawing of non-planar graphs. Namely: *Given a non-planar graph G and a planar subgraph S of G , decide whether G admits a drawing Γ such that the edges of S are not crossed in Γ , and compute Γ if it exists.*

Besides its intrinsic theoretical interest, this problem is also of practical relevance in many application domains. Indeed, distinct groups of edges in a graph may have different semantics, and a group can be more important than another for some applications;

* Work on these results began at the 8th Bertinoro Workshop on Graph drawing. Discussion with other participants is gratefully acknowledged. Part of the research was conducted in the framework of ESF project 10-EuroGIGA-OP-003 GraDR “Graph Drawings and Representations”.

in this case a visual interface might attempt to display more important edges in a planar way. Again, the user could benefit from a layout in which a spanning connected subgraph is drawn crossing free, since it would support the user to quickly recognize paths between any two vertices, while keeping the other edges of the graph visible.

We remark that the problem of recognizing specific types of subgraphs that are not self-crossing (or that have few crossings) in a given drawing Γ , has been previously studied (see, e.g., [17,19,22,25]). This problem, which turns out to be NP-hard for most different kinds of instances, is also very different from our problem. Indeed, in our setting the drawing is not the input, but the output of the problem. Also, we require that the given subgraph S is not crossed by any edge of the graph, not only by its own edges.

In this paper we concentrate on the case in which S is a spanning subgraph of G and consider both straight-line and polyline drawings of G . Namely:

(i) In the straight-line drawing setting we prove that if S is any given spanning spider or caterpillar, then a drawing of G where S is crossing free always exists; such a drawing can be computed in linear time and requires polynomial area (Section 3.1). We also show that this positive result cannot be extended to any spanning tree, but we describe a large family of spanning trees that always admit a solution, and we show that any graph G contains such a spanning tree; unfortunately, our drawing technique for trees may require exponential area. Finally, we characterize the instances $\langle G, S \rangle$ that admit a solution when S is a spanning triconnected subgraph, and we provide a polynomial-time testing and drawing algorithm, whose layouts have polynomial area (Section 3.2).

(ii) We investigate polyline drawings where only the edges of $G \setminus S$ are allowed to bend. In this setting, we show that all spanning trees can be realized without crossings in a drawing of G of polynomial area, and we describe efficient algorithms that provide different trade-offs between number of bends per edge and drawing area (Section 4). Also, in Section 5 we briefly discuss a characterization of the instances $\langle G, S \rangle$ that admit a drawing when S is any given biconnected spanning subgraph.

Due to space restrictions, some proofs are omitted or only sketched in the text; full proofs for all results can be found in the [4].

2 Preliminaries and Definitions

We assume familiarity with basic concepts of graph drawing and planarity (see, e.g., [9]). Let $G(V, E)$ be a graph and let Γ be a drawing of G in the plane. If all vertices and edge bends of Γ have integer coordinates, then Γ is an *integer grid drawing* of G , and the *area* of Γ is the area of the minimum bounding box of Γ . Otherwise, suppose that Γ is not an integer grid drawing and let d_{min} be the minimum distance between two points of Γ on which either vertices or bends are drawn. In this case, the *area* of Γ is defined as the area of the minimum bounding box of a drawing obtained by scaling Γ by a constant c such that $c \times d_{min} = 1$; this corresponds to establish a certain resolution rule between vertices and bends of Γ , which is comparable to that of an integer grid drawing.

Let $G(V, E)$ be a graph and let $S(V, W)$, $W \subseteq E$, be a spanning subgraph of G . A straight-line drawing Γ of G such that S is crossing-free in Γ (i.e., such that crossings

occur only between edges of $E \setminus W$) is called a *straight-line compatible drawing* of $\langle G, S \rangle$. If each edge of $E \setminus W$ has at most k bends in Γ (but still the subdrawing of S is straight-line and crossing-free), Γ is called a *k -bend compatible drawing* of $\langle G, S \rangle$.

If S is a rooted spanning tree of G such that every edge of $G \setminus S$ connects either vertices at the same level of S or vertices that are on consecutive levels, then we say that S is a *BFS-tree* of G .

A *star* is a tree $T(V, E)$ such that all its vertices but one have degree one, that is, $V = \{u, v_1, v_2, \dots, v_k\}$ and $E = \{(u, v_1), (u, v_2), \dots, (u, v_k)\}$; any subdivision of T (including T), is a *spider*: vertex u is the *center* of the spider and each path from u to v_i is a *leg* of the spider. A *caterpillar* is a tree such that removing all its leaves (and their incident edges) results in a path, which is called the *spine* of the caterpillar. The one-degree vertices attached to a spine vertex v are called the *leaves* of v .

In the remainder of the paper we implicitly assume that G is always a connected graph (if the graph is not connected, our results apply for any connected component).

3 Straight-Line Drawings

We start studying straight-line compatible drawings of pairs $\langle G, S \rangle$: Section 3.1 concentrates on the case in which S is a spanning tree, while Section 3.2 investigates the case in which S is a spanning triconnected graph.

3.1 Spanning Trees

The simplest case is when S is a given Hamiltonian path of G ; in this case Γ can be easily computed by drawing all vertices of S in convex position, according to the ordering they occur in the path. In the following we prove that in fact a straight-line compatible drawing Γ of $\langle G, S \rangle$ can be always constructed in the more general cases in which S is a spanning spider (Theorem 1), or a spanning caterpillar (Theorem 2), or a BFS-tree (Theorem 3); our construction techniques guarantee polynomial-area drawings for spiders and caterpillars, while require exponential area for BFS-trees. On the negative side, we show that if S is an arbitrary spanning tree, a straight-line compatible drawing of $\langle G, S \rangle$ may not exist (Lemmas 1 and 2).

Theorem 1. *Let G be a graph with n vertices and m edges, and let S be a spanning spider of G . There exists an integer grid straight-line compatible drawing Γ of $\langle G, S \rangle$. Drawing Γ can be computed in $O(n + m)$ time and has $O(n^3)$ area.*

Proof. Let u be the center of S and let $\pi_1, \pi_2, \dots, \pi_k$ be the legs of S . Also, denote by v_i the vertex of degree one of leg π_i ($1 \leq i \leq k$). Order the vertices of S distinct from u such that: (i) the vertices of each π_i are ordered in the same way they appear in the simple path of S from u to v_i ; (ii) the vertices of π_i precede those of π_{i+1} ($1 \leq i \leq k - 1$). If v is the vertex at position j ($0 \leq j \leq n - 2$) in the ordering defined above, draw v at coordinates (j^2, j) . Finally, draw u at coordinates $(0, n - 2)$. With this strategy, all vertices of S are in convex position, and they are all visible from u in such a way that no edge incident to u can cross other edges of Γ . Hence, the edges of S do not cross other edges in Γ . The area of Γ is $(n - 2)^2 \times (n - 2) = O(n^3)$ and Γ is constructed in linear time. \square

The next algorithm computes a straight-line compatible drawing of $\langle G, S \rangle$ when S is a spanning caterpillar. Theorem 2 proves its correctness, time and area requirements. Although the drawing area is still polynomial, the layout is not an integer grid drawing.

Algorithm. STRAIGHT-LINE-CATERPILLAR. Denote by u_1, u_2, \dots, u_k the vertices of the spine of S . Also, for each spine vertex u_i ($1 \leq i \leq k$), let v_{i1}, \dots, v_{in_i} be its leaves in S (refer to the bottom image in Fig. 1(a)). The algorithm temporarily adds to S and G some dummy vertices, which will be removed in the final drawing. Namely, for each u_i , it attaches to u_i two dummy leaves, s_i and t_i . Also, it adds a dummy spine vertex u_{k+1} attached to u_k and a dummy leaf s_{k+1} of u_{k+1} (see the top image in Fig. 1(a)). Call G' and S' the new graph and the new caterpillar obtained by augmenting G and S with these dummy vertices.

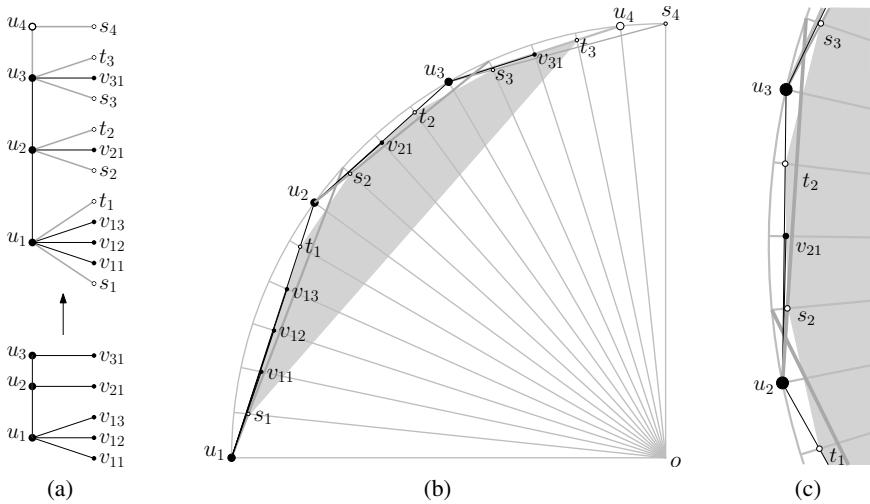


Fig. 1. Illustration of Algorithm STRAIGHT-LINE-CATERPILLAR: (a) a caterpillar S and its augmented version S' ; (b) a drawing of S' ; edges of the graph connecting leaves of S are drawn in the gray (convex) region; (c) enlarged detail of the picture (b)

The construction of a drawing Γ' of G' is illustrated in Fig. 1(b). Consider a quarter of circumference C with center o and radius r . Let N be the total number of vertices of G' . Let $\{p_1, p_2, \dots, p_N\}$ be N equally spaced points along C in clockwise order, where $\overline{op_1}$ and $\overline{op_N}$ are a horizontal and a vertical segment, respectively. For each $1 \leq i \leq k$, consider the ordered list of vertices $L_i = \{u_i, s_i, v_{i1}, \dots, v_{in_i}, t_i\}$, and let L be the concatenation of all L_i . Also, append to L the vertices u_{k+1} and s_{k+1} , in this order. Clearly the number of vertices in L equals N . For a vertex $v \in L$, denote by $j(v)$ the position of v in L . Vertex u_i is drawn at point $p_{j(u_i)}$ ($1 \leq i \leq k$); also, vertices u_{k+1} and s_{k+1} are drawn at points p_{N-1} and p_N , respectively. Each leaf v of S' will be suitably drawn along radius $\overline{op_{j(v)}}$ of C . More precisely, for any $i \in \{1, \dots, k\}$, let a_i be the intersection point between segments $\overline{p_{j(u_i)}p_{j(s_{i+1})}}$ and $\overline{op_{j(s_i)}}$, and let b_i be the intersection point between segments $\overline{p_{j(u_i)}p_{j(u_{i+1})}}$ and $\overline{op_{j(t_i)}}$. Vertices s_i and

t_i are drawn at points a_i and b_i , respectively. Also, let A_i be the circular arc that is tangent to $\overline{p_{j(u_i)}p_{j(u_{i+1})}}$ at point b_i , and that passes through a_i ; vertex v_{ih} is drawn at the intersection point between A_i and $\overline{op_{j(v_{ih})}}$ ($1 \leq h \leq n_i$).

Once all vertices of G' are drawn, each edge of G' is drawn in Γ' as a straight-line segment between its end-vertices. Drawing Γ is obtained from Γ' by deleting all dummy vertices and their incident edges.

Theorem 2. *Let G be graph with n vertices and m edges, and let S be a spanning caterpillar of G . There exists a straight-line compatible drawing Γ of $\langle G, S \rangle$. Drawing Γ can be computed in $O(n + m)$ time in the real RAM model¹ and has $O(n^2)$ area.*

Proof sketch: Let Γ be the output of Algorithm STRAIGHT-LINE-CATERPILLAR. We first prove that Γ is a straight-line compatible drawing of $\langle G, S \rangle$, and then we analyze time complexity and area requirement. We adopt the same notation used in the description of the algorithm.

CORRECTNESS. We have to prove that in Γ the edges of S are never crossed. Our construction places all spine vertices of S' (and hence of S) in convex position. It is also possible to see that the leaves of S' are all in convex position and form a convex polygon P . Since by construction the edges of S are all outside P in Γ , these edges cannot be crossed by edges of G connecting two leaves of S . Also, it is immediate to see that an edge of S cannot be crossed by another edge of S and it is not difficult to see that an edge of S cannot be crossed by an edge of G connecting either two non-consecutive spine vertices or a leaf of S to a spine vertex of S .

TIME AND AREA REQUIREMENT. Clearly, the construction of Γ' (and then of Γ) can be executed in linear time, in the real RAM model. About the area, let d_{\min} be the minimum distance between any two vertices of Γ . It can be proved that if we require $d_{\min} \geq 1$ then $r < \frac{\sqrt{2}}{\beta}$, for $\beta = \theta(\frac{1}{N})$. Thus, the area of Γ is $O(N^2) = O(n^2)$. \square

The next lemmas show that, unfortunately, Theorem 1 and Theorem 2 cannot be extended to any spanning tree S , that is, there are pairs $\langle G, S \rangle$ that do not admit a straight-line compatible drawing, even if S is a ternary or a binary tree.

Lemma 1. *Let G be the complete graph on 13 vertices and let S be a complete rooted ternary tree that spans G . There is no straight-line compatible drawing of $\langle G, S \rangle$.*

Proof sketch: Suppose, for a contradiction, that a straight-line compatible drawing Γ of $\langle G, S \rangle$ exists. Let r be the root of S (see Fig. 2(a)). Note that r is the only vertex of S with degree 3. Let u, v, w be the three neighbors of r in S . Two are the cases: either one of u, v, w (say u) lies inside triangle $\Delta(r, v, w)$ (Case 1, see Fig. 2(b)); or r lies inside triangle $\Delta(u, v, w)$ (Case 2).

In Case 1, consider a child u_1 of u . Vertex u_1 is placed in Γ in such a way that u lies inside either triangle $\Delta(u_1, r, w)$ or triangle $\Delta(u_1, r, v)$; assume the former (see Fig. 2(c)). Then, consider another child u_2 of u ; in order for edge (u, u_2) not to cross any edge, also u_2 has to lie inside $\Delta(u_1, r, w)$, in such a way that both u and u_1 lie inside triangle $\Delta(u_2, r, v)$. This implies that u lies inside $\Delta(u_1, r, u_2)$ (see Fig. 2(d)),

¹ We also assume that basic trigonometric functions are executed in constant time.

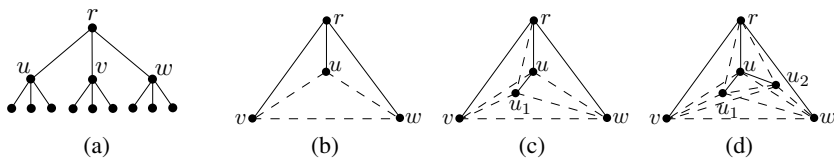


Fig. 2. Illustration for Lemma 1: (a) A complete rooted ternary tree with 13 vertices. (b) Case 1 in the proof; u lies inside $\triangle(r, v, w)$. (c) Placement of u_1 . (d) Placement of u_2 .

together with its last child u_3 . However, u_3 cannot be placed in any of the three triangles in which $\triangle(u_1, r, u_2)$ is partitioned by the edges (of S) connecting u to u_1 , to r , and to u_2 , respectively, without introducing any crossing involving edges of S , a contradiction. Case 2 can be analyzed with analogous considerations. \square

The proof strategy of Lemma 2 is similar to that of Lemma 1.

Lemma 2. *Let G be the complete graph on 22 vertices and let S be a complete unrooted binary tree that spans G . There is no straight-line compatible drawing of $\langle G, S \rangle$.*

In the light of Lemmas 1 and 2, it is natural to ask whether there are specific subfamilies of spanning trees S (other than paths, spiders, and caterpillars) such that a straight-line compatible drawing of $\langle G, S \rangle$ always exists. The next algorithm gives a positive answer to this question: it computes a straight-line compatible drawing when S is a BFS-tree of G . Theorem 3 proves the algorithm correctness, its time complexity, and its area requirement.

Algorithm. STRAIGHT-LINE-BFS-TREE. Let u be the root of S (which is at level 0) and let u_{l1}, \dots, u_{lk_l} be the vertices at level $l \in \{1, \dots, d\}$, where d is the depth of S . The algorithm temporarily adds to S and G some dummy vertices, which will be removed in the final drawing. Namely, for each $u_{li}, 1 \leq l \leq d - 1$ and $1 \leq i \leq k_l$, it attaches to u_{li} one more (leftmost) child s_{li} . Also, it attaches to root u a dummy (rightmost) child t . Denote by G' and S' the new graph and the new tree, respectively. Notice that S' is still a BFS-tree of G' . The algorithm iteratively computes a drawing Γ' of G' . For $l = 1, \dots, d$, the algorithm defines a circumference C_l with center $o = (0, 0)$ and radius $r_l < r_{l-1}$ (C_1, \dots, C_d are concentric). The vertices of level l are drawn on the quarter of C_l going from point $(-r_l, 0)$ to point $(0, r_l)$ clockwise.

Let $\{u_{11}, \dots, u_{1k_1}, t\}$ be the ordered list of the children of root u and let $\{p_{11}, \dots, p_{1k_1}, p_t\}$ be $k_1 + 1$ equally spaced points along C_1 in clockwise order, where $\overline{op_{11}}$ and $\overline{op_t}$ are a horizontal and a vertical segment, respectively. Vertex u_{1j} is drawn on p_{1j} ($1 \leq j \leq k_1$) and vertex t is drawn on p_t . Also, u is drawn on point $(-r_1, r_1)$.

Assume now that all vertices u_{l1}, \dots, u_{lk_l} of level l have been drawn ($1 \leq l \leq d - 1$) in this order on the sequence of points $\{q_1, \dots, q_{k_l}\}$, along C_l . The algorithm draws the vertices of level $l + 1$ as follows. Let $\overline{q_i q_{i+1}}$ be the chords of C_l , for $1 \leq i \leq k_l - 1$, and let c_l be the shortest of these chords. The radius r_{l+1} of C_{l+1} is chosen arbitrarily in such a way that C_{l+1} intersects c_l in two points and $r_{l+1} < r_l$. This implies that C_{l+1} also intersects every chord $\overline{q_i q_{i+1}}$ in two points. For $1 \leq i \leq k_l$, denote by $L(u_{li}) = \{v_1, \dots, v_{n_{li}}\}$ the ordered list of children of u_{li} in G' . Also, let a_i be the intersection

point between $\overline{q_i q_{i+1}}$ and C_{l+1} that is closest to q_i , and let ℓ_i be the line through q_i tangent to C_{l+1} ; denote by b_i the tangent point between ℓ_i and C_{l+1} . Let A_{l+1} be the arc of C_{l+1} between a_i and b_i , and let $\{p_0, p_1, \dots, p_{n_{l_i}}\}$ be $n_{l_i} + 1$ equally spaced points along A_{l+1} in clockwise order. For $v \in L(u_{l_i})$, denote by $j(v)$ the position of v in $L(u_{l_i})$. Vertex v_j is drawn on $p_{j(v)}$ ($1 \leq j \leq n_{l_i}$) and vertex s_{l_i} is drawn on p_0 .

Once all vertices of G' are drawn each edge of G' is drawn in Γ' as a straight-line segment between its end-vertices. Drawing Γ is obtained from Γ' by deleting all dummy vertices and their incident edges.

Theorem 3. *Let G be a graph with n vertices and m edges, and let S be a BFS-tree of G . There exists a straight-line compatible drawing Γ of $\langle G, S \rangle$. Drawing Γ can be computed in $O(n + m)$ time in the real RAM model.*

It is worth observing that any graph G admits a BFS-tree rooted at an arbitrarily chosen vertex r of G . Thus, each graph admits a straight-line drawing Γ such that one of its spanning trees S is never crossed in Γ . Unfortunately, the compatible drawing computed by Algorithm STRAIGHT-LINE-BFS-TREE may require area $\Omega(2^n)$.

3.2 Spanning Triconnected Subgraphs

Here we focus on triconnected spanning subgraph S of G . Clearly, since every tree can be augmented with edges to become a triconnected graph, Lemmas 1 and 2 imply that, if S is a triconnected graph, a straight-line compatible drawing of $\langle G, S \rangle$ may not exist. The next theorem characterizes those instances for which such a drawing exists.

Theorem 4. *Let $G(V, E)$ be a graph, $S(V, W)$ be a spanning planar triconnected subgraph of G , and \mathcal{E} be the unique planar (combinatorial) embedding of S (up to a flip). A straight-line compatible drawing Γ of $\langle G, S \rangle$ exists if and only if: (1) Each edge $e \in E \setminus W$ connects two vertices belonging to the same face of \mathcal{E} . (2) There exists a face f of \mathcal{E} containing three vertices such that any pair u, v of them does not separate in the circular order of f the end-vertices $x, y \in f$ of any other edge in $E \setminus W$.*

Proof sketch: Since \mathcal{E} is unique the necessity of Condition 1 is trivial. Its sufficiency would be also trivial if S admitted a convex drawing where the external face is a triangle. Otherwise, suppose that v_1, v_2 , and v_3 are three vertices of a face f satisfying Condition 2. Dummy vertices can be added to S among v_1, v_2 , and v_3 in order to have a triangular face to be used as external face when computing the convex drawing (for example, using the algorithm in [27]). The necessity of Condition 2 follows from considering any three vertices on the convex hull of a compatible drawing of $\langle G, S \rangle$. \square

The next algorithm exploits Theorem 4 in order to decide in polynomial time whether $\langle G, S \rangle$ admits a straight-line compatible drawing.

Algorithm. STRAIGHT-LINE-TRICONNECTED. Let \mathcal{E} be the unique planar embedding of S (up to a flip). The algorithm verifies that each edge of $E \setminus W$ satisfies Condition 1 of Theorem 4 and that there exists a face f of \mathcal{E} containing three vertices v_1, v_2 , and v_3 , that satisfy Condition 2 of Theorem 4. If both conditions hold, then v_1, v_2 , and v_3 can be used to find a straight-line compatible drawing Γ of $\langle G, S \rangle$ as described in the proof of Theorem 4.

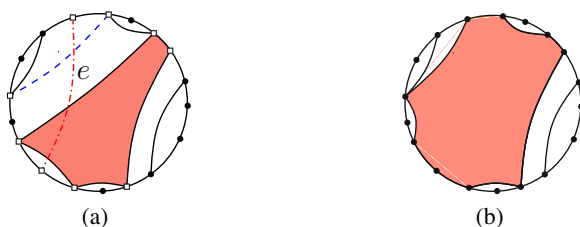


Fig. 3. Two consecutive steps of Algorithm STRAIGHT-LINE-TRICONNECTED. (a) The outerplane graph G_f ; the shaded face is `full` (the others are `empty`); the dash-dot edge e is the next edge of E_f to be considered; edges in E_χ are drawn as dashed lines; white squares are vertices of V_χ . (b) Graph G_f after the update due to edge e .

Condition 1 is verified as follows. Construct an auxiliary graph S' from S by subdividing each edge e of W with a dummy vertex v_e . Also, for each face f of \mathcal{E} add to S' a vertex v_f and connect v_f to all non-dummy vertices of f . We have that two vertices of V belong to the same face of \mathcal{E} if and only if their distance in S' is two.

To test Condition 2 of Theorem 4 we perform the following procedure on each face f of \mathcal{E} , restricting our attention to the set E_f of edges in $E \setminus W$ whose end-vertices belong to f . We maintain an auxiliary outerplane graph G_f whose vertices are the vertices V_f of f . Each internal face of G_f is either marked as `full` or as `empty`. Faces marked `full` are not adjacent to each other. Intuitively, we have that any three vertices of an `empty` face satisfy Condition 2, while all triples of vertices of a `full` face do not. We initialize G_f with the cycle composed of the vertices and the edges of f and mark its unique internal face as `empty`. At each step an edge e of E_f is considered and G_f is updated. If adding e to G_f splits a single face marked `empty`, we update G_f by splitting such a face into two `empty` faces. If the end-vertices of e belong to a single face marked `full`, we ignore e . Otherwise, adding e to G_f would cross several edges and faces (see Fig. 3(a)). Consider the set E_χ of internal edges of G_f crossed by e . Define a set of vertices V_χ of G_f with the end-vertices of e , the end-vertices of edges of E_χ that are incident to two `empty` faces, the vertices of the `full` faces traversed by e . Remove all edges in E_χ from G_f . Mark the face f' obtained by such a removal as `empty`. Form a new face f_χ inside f' with all vertices in V_χ by connecting them as they appear in the circular order of f , and mark f_χ as `full` (see Fig. 3(b)).

When all the edges of E_f have been considered, if G_f has an internal face marked as `empty`, any three vertices of this face satisfy Condition 2. Else, G_f has a single internal face marked `full` and all triples of vertices of f do not satisfy Condition 2.

Theorem 5. *Let $G(V, E)$ be a graph and let $S(V, W)$ be a spanning triconnected planar subgraph of G . There exists an $O(|V| \times |E \setminus W|)$ -time algorithm that decides whether $\langle G, S \rangle$ admits a straight-line compatible drawing Γ and, in the positive case, computes it on an $O(|V|^2) \times O(|V|^2)$ grid.*

Proof sketch: Algorithm STRAIGHT-LINE-TRICONNECTED constructs Γ . Its correctness trivially descends from Theorem 4. Regarding the time complexity, the unique planar embedding \mathcal{E} of S can be computed in $O(|V|)$ time. The auxiliary graph S' for

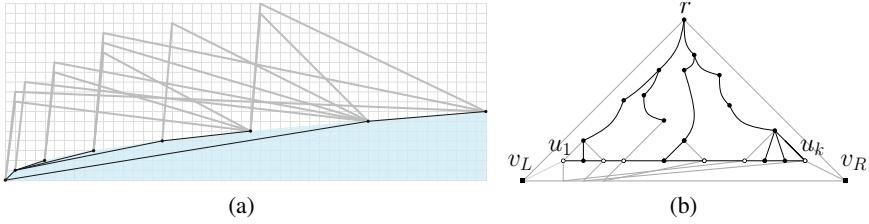


Fig. 4. Illustration of: (a) Algorithm ONE-BEND TREE and (b) Algorithm THREE-BEND TREE; a graph G with a given spanning tree S (black edges)

testing Condition 1 can be constructed in time linear in the size of S . Since S' is a planar graph, deciding if two vertices have distance two can be done in constant time [21]. Thus, testing Condition 1 for all edges in $E \setminus W$ can be done in $O(|V| + |E \setminus W|)$ time. While verifying Condition 1, E_f can be computed, for each face f of \mathcal{E} , in $O(|V| + |E \setminus W|)$ time. Since for each face f of \mathcal{E} , the size of G_f is $O(|V_f|)$, adding edges in E_f has time complexity $O(|E_f| \times |V_f|)$. Overall, we have that the time complexity of testing Condition 2 is $O(|E \setminus W| \times |V|)$, which gives the time complexity of the whole algorithm. Regarding the area, the algorithm in [5] can be used to obtain in linear time a straight-line grid drawing of S on an $O(|V|^2) \times O(|V|^2)$ grid; this drawing is strictly convex. \square

4 Polyline Drawings

We now prove that, using bends along the edges of $G \setminus S$ allows us to compute compatible drawings of pairs $\langle G, S \rangle$ for every spanning tree S of G ; such drawings are on a polynomial-area grid. In particular, since edge bends are negatively correlated to the drawing readability, we want to compute k -bend compatible drawings for small values of k . We provide algorithms that offer different trade-offs between number of bends and drawing area. In Section 5 we briefly discuss some preliminary results about 1-bend compatible drawings of $\langle G, S \rangle$ when S is a biconnected spanning subgraph.

Let $G(V, E)$ be a graph with n vertices and m edges, and let $S(V, W)$ be any spanning tree of G . We denote by $x(v)$ and $y(v)$ the x - and the y -coordinate of a vertex v , respectively. The next algorithm computes a 1-bend compatible drawing of $\langle G, S \rangle$.

Algorithm. ONE-BEND TREE. The algorithm works in two steps (refer to Fig. 4(a)).

STEP 1: Consider a point set of size n such that for each point p_i , the x - and y -coordinates of p_i are i^2 and i , respectively. Construct a straight-line drawing of S by placing the vertices on points p_i , $1 \leq i \leq n$, according to a DFS traversal.

STEP 2: Let v_i be the vertex placed on point p_i . For each $i \in \{1, \dots, n\}$, draw each edge $(v_i, v_j) \in E \setminus W$ such that $j > i$ as a polyline connecting p_i and p_j , and bending at point $(i^2 + 1, n + c)$ where c is a progressive counter, initially set to one.

Theorem 6. *Let $G(V, E)$ be a graph with n vertices and m edges, and let $S(V, W)$ be any spanning tree of G . There exists a 1-bend compatible drawing Γ of $\langle G, S \rangle$. Drawing Γ can be computed in $O(n + m)$ time and has $O(n^2(n + m))$ area.*

Proof sketch: The algorithm that computes Γ is Algorithm ONE-BEND TREE. Note that the drawing of S contained in Γ is planar, and that the edges in $E \setminus W$ are drawn outside the convex region containing the drawing of S . About area requirements, the width of Γ is $O(n^2)$, by construction, while the height of Γ is given by the y -coordinate of the topmost bend point, that is $n + m$. \square

Next, we describe an algorithm that constructs 3-bend compatible drawings of pairs $\langle G, S \rangle$ with better area bounds than Algorithm ONE-BEND TREE for sparse graphs.

Algorithm. THREE-BEND TREE. The algorithm works in four steps (see Fig. 4(b)).

STEP 1: Let G' be the graph obtained from G by subdividing each edge $(v_i, v_j) \in E \setminus W$ with two dummy vertices $d_{i,j}$ and $d_{j,i}$. Let S' be the spanning tree of G' , rooted at any non-dummy vertex r , obtained by deleting all edges connecting two dummy vertices. Clearly, every dummy vertex is a leaf of S' .

STEP 2: For each vertex of S' , order its children arbitrarily, thus inducing a left-to-right order of the leaves of S' . Rename the leaves of S' as u_1, \dots, u_k following this order. For each $i \in \{1, \dots, k - 1\}$, add an edge (u_i, u_{i+1}) to S' . Also, add to S' two dummy vertices v_L and v_R , and edges $(v_L, r), (v_R, r), (v_L, u_1), (u_k, v_R), (v_L, v_R)$.

STEP 3: Construct a straight-line grid drawing Γ' of S' , as described in [18], in which edge (v_L, v_R) is drawn as a horizontal segment on the outer face, vertices u_1, \dots, u_k all lie on points having the same y -coordinate Y , and the rest of S' is drawn above such points. Remove from Γ' the vertices and edges added in STEP 2.

STEP 4: Compute a drawing Γ of G such that each edge in W is drawn as in Γ' , while each edge $(v_i, v_j) \in E \setminus W$ is drawn as a polyline connecting v_i and v_j , bending at $d_{i,j}$, at $d_{j,i}$, and at a point $(c, Y - 1)$ where c is a progressive counter, initially set to $x(u_1)$.

Theorem 7. *Let $G(V, E)$ be a graph with n vertices and m edges, and let $S(V, W)$ be any spanning tree of G . There exists a 3-bend compatible drawing Γ of $\langle G, S \rangle$. Drawing Γ can be computed in $O(n + m)$ time and has $O((n + m)^2)$ area.*

Proof sketch: The algorithm that computes Γ is Algorithm THREE-BEND TREE. The drawing of S contained in Γ is planar ([18]) and lies above the horizontal line $y = Y$. The area bounds descend from the construction and from the area bounds of [18]. \square

We finally remark that there exists a drawing algorithm that computes 4-bend compatible drawings that are more readable than those computed by Algorithm THREE-BEND TREE. Although the area of these drawings is still $O((n + m)^2)$, they have optimal crossing angular resolution, i.e., edges cross only at right angles. Drawings of this type are called *RAC drawings* and are widely studied in the literature [13,14].

5 Discussion

We initiated the study of a new problem in graph drawing, i.e., computing a drawing Γ of a non-planar graph G such that a desired subgraph $S \subseteq G$ is crossing-free in Γ . In the setting where edges are straight-line segments and S is a spanning tree of G , we showed that Γ does not always exist; also, we provided existential and algorithmic results for meaningful subfamilies of spanning trees and we described a linear-time

testing and drawing algorithm when S is a spanning triconnected subgraph. One of the main problems still open in this setting is the following: *Given a graph G and a spanning tree S of G , what is the complexity of deciding whether $\langle G, S \rangle$ admits a straight-line compatible drawing?* This problem can be also studied when S is a biconnected spanning subgraph, trying to extend the characterization of Theorem 4. Another interesting problem is to extend the results of Lemmas 1 and 2 in order to give a characterization of what spanning trees S of a complete graph can be always realized.

Allowing bends on the edges of $G \setminus S$, a drawing Γ exists for any given spanning tree S ; we described several efficient algorithms that offer different compromises between drawing area and number of bends. Also, in this setting we have a characterization of which pairs $\langle G, S \rangle$ admit a 1-bend compatible drawing when S is a biconnected spanning subgraph. Namely, a necessary and sufficient condition is that S has a planar embedding such that for each edge e of $G \setminus S$ the end-vertices of e belong to the same face of S (as for Condition 1 of Theorem 4). Given such an embedding one can: (i) add a dummy vertex inside each face of S and connect it to all the vertices of the face; (ii) construct a planar straight-line drawing of the resulting graph, and (iii) construct a 1-bend compatible drawing where each edge (u, v) of $G \setminus S$ has a bend-point coinciding with the dummy vertex of the face containing u and v . A small perturbation of the bend-points will avoid that two of them coincide. An algorithm that tests the condition above can be derived as a simplification of the algorithm in [3], used to test the existence of a Simultaneous Embedding with Fixed Edges (SEFE) of two graphs [6]. Finally, we remark that Algorithm ONE-BEND TREE can be adapted to find a 1-bend compatible drawing when S is an outerplanar graph with the same bounds stated by Theorem 6. Many problems for k -compatible drawings are still open. Among them: trying to reduce the area bounds when S is a tree and devising algorithms for computing grid 1-bend compatible drawings of feasible $\langle G, S \rangle$ when S is biconnected.

References

1. Ackerman, E.: On the maximum number of edges in topological graphs with no four pairwise crossing edges. *Discrete & Computational Geometry* 41(3), 365–375 (2009)
2. Ackerman, E., Tardos, G.: On the maximum number of edges in quasi-planar graphs. *Journal of Combinatorial Theory, Ser. A* 114(3), 563–571 (2007)
3. Angelini, P., Di Battista, G., Frati, F., Patrignani, M., Rutter, I.: Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph. *Journal of Discrete Algorithms* 14, 150–172 (2012)
4. Angelini, P., Binucci, C., Da Lozzo, G., Didimo, W., Grilli, L., Montecchiani, F., Patrignani, M., Tollis, I.G.: Drawings of non-planar graphs with crossing-free subgraphs. *ArXiv e-prints* 1308.6706 (September 2013)
5. Bárány, I., Rote, G.: Strictly convex drawings of planar graphs. *Documenta. Math.* 11, 369–391 (2006)
6. Blasiüs, T., Kobourov, S.G., Rutter, I.: Simultaneous embedding of planar graphs. In: Tamassia, R. (ed.) *Handbook of Graph Drawing and Visualization*. CRC Press (2013)
7. Brandenburg, F.J., Eppstein, D., Gleißner, A., Goodrich, M.T., Hanauer, K., Reislhuber, J.: On the density of maximal 1-planar graphs. In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 327–338. Springer, Heidelberg (2013)

8. Buchheim, C., Chimani, M., Gutwenger, C., Jünger, M., Mutzel, P.: Crossings and planarization. In: Tamassia, R. (ed.) *Handbook of Graph Drawing and Visualization*. CRC Press (2013)
9. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Graph Drawing*. Prentice Hall, Upper Saddle River (1999)
10. Di Giacomo, E., Didimo, W., Liotta, G., Montecchiani, F.: h -quasi planar drawings of bounded treewidth graphs in linear area. In: Golumbic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) *WG 2012*. LNCS, vol. 7551, pp. 91–102. Springer, Heidelberg (2012)
11. Di Giacomo, E., Didimo, W., Liotta, G., Montecchiani, F.: Area requirement of graph drawings with few crossings per edge. *Computational Geometry* 46(8), 909–916 (2013)
12. Didimo, W.: Density of straight-line 1-planar graph drawings. *Information Processing Letters* 113(7), 236–240 (2013)
13. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. *Theoretical Computer Science* 412(39), 5156–5166 (2011)
14. Didimo, W., Liotta, G.: The crossing angle resolution in graph drawing. In: Pach, J. (ed.) *Thirty Essays on Geometric Graph Theory*. Springer (2013)
15. Eades, P., Hong, S.H., Katoh, N., Liotta, G., Schweitzer, P., Suzuki, Y.: Testing maximal 1-planarity of graphs with a rotation system in linear time - (extended abstract). In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 339–345. Springer, Heidelberg (2013)
16. Hong, S.-H., Eades, P., Liotta, G., Poon, S.-H.: Fáry's theorem for 1-planar graphs. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) *COCOON 2012*. LNCS, vol. 7434, pp. 335–346. Springer, Heidelberg (2012)
17. Jansen, K., Woeginger, G.J.: The complexity of detecting crossingfree configurations in the plane. *BIT Numerical Mathematics* 33(4), 580–595 (1993)
18. Kant, G.: Drawing planar graphs using the canonical ordering. *Algorithmica* 16(1), 4–32 (1996)
19. Knauer, C., Schramm, É., Spillner, A., Wolff, A.: Configurations with few crossings in topological graphs. *Computational Geometry* 37(2), 104–114 (2007)
20. Korzhik, V.P., Mohar, B.: Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory* 72(1), 30–71 (2013)
21. Kowalik, L., Kurowski, M.: Short path queries in planar graphs in constant time. In: Larmore, L.L., Goemans, M.X. (eds.) *STOC 2003*, pp. 143–148. ACM (2003)
22. Kratochvíl, J., Lubiv, A., Nešetřil, J.: Noncrossing subgraphs in topological layouts. *SIAM Journal on Discrete Mathematics* 4(2), 223–244 (1991)
23. Pach, J., Shahrokhi, F., Szegedy, M.: Applications of the crossing number. *Algorithmica* 16(1), 111–117 (1996)
24. Pach, J., Tóth, G.: Graphs drawn with few crossings per edge. *Combinatorica* 17(3), 427–439 (1997)
25. Rivera-Campo, E., Urrutia-Galicia, V.: A sufficient condition for the existence of plane spanning trees on geometric graphs. *Computational Geometry* 46(1), 1–6 (2013)
26. Suk, A.: k -quasi-planar graphs. In: Speckmann, B. (ed.) *GD 2011*. LNCS, vol. 7034, pp. 266–277. Springer, Heidelberg (2011)
27. Tutte, W.T.: How to draw a graph. *Proceedings of the London Mathematical Society* s3-13(1), 743–767 (1963)
28. Valtr, P.: On geometric graphs with no k pairwise parallel edges. *Discrete & Computational Geometry* 19(3), 461–469 (1998)

Exploring Complex Drawings via Edge Stratification

Emilio Di Giacomo¹, Walter Didimo¹, Giuseppe Liotta¹,
Fabrizio Montecchiani¹, and Ioannis G. Tollis²

¹ Dip. di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia
{digiacomo, didimo, liotta, montecchiani}@diei.unipg.it

² Univ. of Crete and Institute of Computer Science-FORTH, Greece
tollis@ics.forth.gr

Abstract. We propose an approach that allows a user to explore a layout produced by any graph drawing algorithm, in order to reduce the visual complexity and clarify its presentation. Our approach is based on *stratifying* the drawing into *layers* with desired properties; layers can be explored and combined by the user to gradually acquire details. We present stratification heuristics, a user study, and an experimental analysis that evaluates how our stratification heuristics behave on the drawings computed by a variety of popular force-directed algorithms.

1 Introduction

Graph drawing algorithms are used in many applications to visualize networked information. Among them, force-directed algorithms are the most popular and are widely adopted to compute drawings in which vertices are represented as small circles and edges are drawn as straight-line segments. Of course, the chosen algorithm is of great importance in creating a readable visualization. However, when the graph is complex (large or locally dense) a high number of edge crossings is typically unavoidable; this is the case, for example, of most small world and scale-free graphs (see, e.g., [13,30]). It is well known that a high number of edge crossings seriously affects the drawing readability [26,27], and makes it hard to perform detailed tasks based on visual inspection. These tasks include finding the shortest path between two given vertices, finding the vertices that are adjacent to both, or even determining the degree of a vertex.

In this paper we propose a new approach to support the user in the visual inspection of complex drawings. Namely, given a drawing Γ of a graph $G(V, E)$, we aim at partitioning the set of edges E into subsets E_1, E_2, \dots, E_h , such that the subdrawing $\Gamma_i \subseteq \Gamma$ of each subgraph $G_i(V, E_i)$ guarantees some desired readability property (in each subdrawing, the vertices remain fixed in their original positions as determined in Γ). For example, a user could prefer to see Γ_i without any edge crossing, i.e., as planar, or that any two crossing edges form a sufficiently large angle. We say that Γ is *stratified* into a set of *layers* Γ_i , each containing all the vertices of Γ (in their original positions) but only a portion of the edge set. The user can then interact with this edge stratification, by exploring one layer at a time, or by arbitrarily combining multiple layers into a single view. The edges of each layer are assigned the same color and different colors are used for the edges of different layers. The main advantage of this approach is that

users can get multiple readable views of different portions of the drawing, with the possibility of simplifying the total amount of information, thus allowing them to gradually acquire details by exploring or combining layers. On the negative side, from the cognitive point of view, the user has to face the difficulty of making sense of a distributed information. To deal with this difficulty in practical terms, it is crucial to minimize the number of layers required to stratify the drawing so that the desired readability property is guaranteed for each layer. The main contribution of this paper is as follows:

(i) We define an edge stratification model and the related optimization problems. Then, we give a general framework to solve these problems for several desired readability properties of the layers, and we describe heuristics within this framework (Section 3).

(ii) We present the results of a user study aimed at understanding the effectiveness of the proposed approach for executing tasks based on visual inspection (Section 4). These results highlight the usefulness of edge stratification, especially for some of these tasks and for some specific readability properties of the layers.

(iii) We present an experimental analysis that compares the number of layers required to stratify drawings computed by a variety of popular force-directed algorithms, using our heuristics (Section 5). On one side, these experiments suggest that for some of the computed drawings the number of layers required by some edge stratification is a more reliable measure of the drawing visual complexity with respect to the number of edge crossings. On the other side, the results show that most of the force-directed algorithms that we have considered guarantee a strong correlation between number of crossings and number of layers in the stratifications of their drawings. We interpret this behavior as a positive feature of the drawing algorithms, which witnesses a quite uniform distribution of the crossings in the drawing.

In Fig. 1 we give an example of how the number of layers produced by the stratification heuristics in this paper can be used to measure the readability of different drawings of a same graph. Fig. 1(a) shows a drawing T_1 of a graph with 50 vertices and 150 edges, computed by Fruchterman-Reingold's algorithm [17] and containing 1,395 edge crossings; Fig. 1(b) shows a drawing T_2 of the same graph, computed by Kamada-Kawai's algorithm [21] and having 1,437 edge crossings. Applying our stratification heuristic to compute planar layers, T_1 requires 8 layers while drawing T_2 requires 7 layers; even if we require crossing angles of at least $\frac{\pi}{4}$ in each layer, our stratification heuristic generates 4 layers for T_1 (Fig. 1(c)) and only 3 layers for T_2 (Fig. 1(d)). Indeed, despite its higher number of crossings, drawing T_2 appears more readable, due to a more uniform distribution of the crossings and a better area. Namely, in the figure the two drawings are scaled to fit in the same bounding box; if we scale the drawings such that they satisfy the same resolution rule, drawing T_1 has twice the area of drawing T_2 .

2 Related Work

To reduce the negative impact of edge crossings, different constraints on the type of crossings have been studied. Some of them require that edges cross only at large angles, or that each edge is crossed at most a limited number of times, or even that only few pairwise crossing edges are allowed. A very limited list of papers includes [10,11,12]. Unfortunately, only restricted sub-families of sparse graphs admit drawings that respect

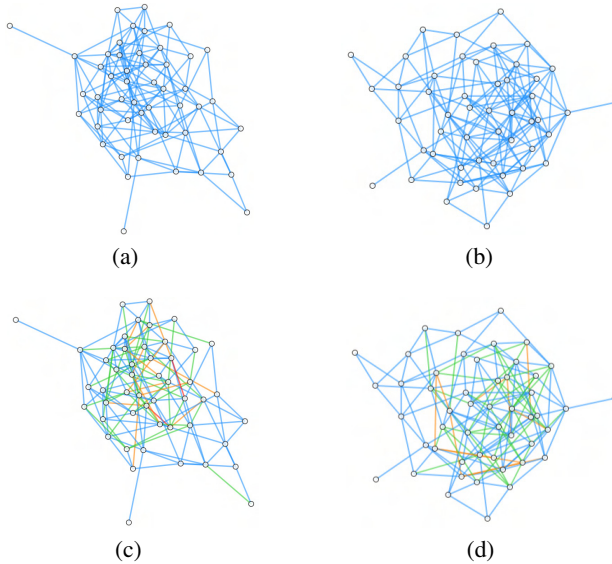


Fig. 1. (a-b) Drawings of the same graph computed by two different force-directed algorithms. (c-d) The same drawings in (a-b) stratified with layers having crossing angles of at least $\frac{\pi}{4}$; layers are conveyed with different edge colors.

these constraints. Also, an impressive set of crossing minimization methods are proposed in graph drawing (see [8] for a survey). However, these methods become computationally expensive (or even unfeasible) for large and dense graphs.

Many visualization techniques that compute a *hierarchical clustering* of the vertices and that allow users to interactively explore it are also known (see , e.g., [1,3,30]). The levels of a cluster hierarchy are a sort of vertex stratification, which allows users to control the amount of information displayed in the same view. Other well-studied approaches that facilitate in the visual exploration of networked data are based on node or edge *filtering*, *grouping*, and *motif simplification* (see, e.g., [1,29]).

Our edge stratification approach does not aim at computing drawings of graphs with controlled visual complexity, but rather it starts from a drawing of a graph and aims at supporting its exploration and analysis by distributing the whole drawing information into a set of logical layers with desired edge crossings properties. This idea is somewhat related to the notion of *geometric thickness* of a graph G [14,15], which is the minimum number of colors that can be assigned to the edges of G , such that there exists a straight-line drawing of G where no two edges of the same color cross. Similar to thickness, our stratification defines an edge coloring; the difference is that stratification is executed on a specific drawing, which cannot be changed. Hence, if the geometric thickness can be used as a measure of the graph complexity, the minimum size of a stratification of a straight-line drawing can be used as a measure of the drawing visual complexity in terms of number, types, and distribution of its edge crossings. Clearly, the size of a stratification into planar layers of a drawing Γ of G cannot be smaller than the geometric thickness of G .

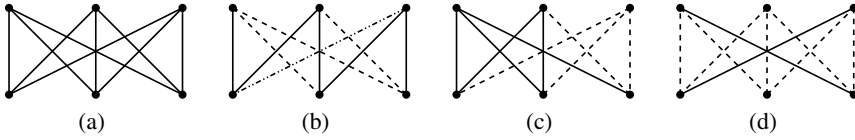


Fig. 2. (a) A straight-line drawing Γ of $K_{3,3}$. (b) A stratification $S(\Gamma, \text{PLANARITY})$. (c) A stratification $S(\Gamma, \text{LAC}(\frac{\pi}{3}))$. (d) A stratification $S(\Gamma, \text{1-PLANARITY})$. In each stratification different dash styles for the edges represent different layers.

Other approaches have been proposed for supporting the analysis of a given a drawing. One of the most popular is *edge bundling*, which deforms and groups together edges that are similar according to some metric (see [32] for a survey). Another approach, called *geometric graph generalization*, reduces vertex and/or edge clutter by collapsing groups of vertices that are geometrically close to one another into a single point [7]. Differently from our stratification, edge bundling and geometric graph generalization modify the input drawing, emphasizing its skeletal structure at the expenses of loss of details. We finally mention another recent technique, which aims to visually simplify or remove edge crossings in a given drawing by displaying only portions of the crossing edges; the modified drawings are called *partial edge drawings* [6].

3 Stratification: Model and Algorithms

Here we formally describe our edge stratification model and related algorithms.

Model. Let $G(V, E)$ be a graph and let Γ be a straight-line drawing of G . Given a subset $E' \subseteq E$, $G[E']$ denotes the subgraph $G'(V, E')$ of G , and $\Gamma[E']$ is the subdrawing of $G[E']$ in Γ . Also, let \mathcal{P} denote a desired geometric property of a drawing of a graph. An *edge stratification* (or simply a *stratification*) of Γ with respect to \mathcal{P} , also denoted as $S(\Gamma, \mathcal{P})$, is a partition of the edges of G into h subsets E_1, E_2, \dots, E_h such that, for every $i \in \{1, \dots, h\}$, property \mathcal{P} holds for $\Gamma[E_i]$. Each subdrawing $\Gamma[E_i]$ is called a *layer* of $S(\Gamma, \mathcal{P})$, and the *size* of $S(\Gamma, \mathcal{P})$ is the number h of its layers. We study the following general optimization problem.

Problem 1. – **MINGENERALSTRATIFICATION:** Given a straight-line drawing Γ and a geometric property \mathcal{P} , find a stratification $S(\Gamma, \mathcal{P})$ of minimum size.

In particular, we focus on the following geometric properties \mathcal{P} : (i) **PLANARITY:** the drawing is crossing free; (ii) **LAC(α):** any two crossing edges of the drawing form an angle of at least α radians (LAC stands for *large angle crossing*); (iii) **k -PLANARITY:** each edge of the drawing is crossed at most by k edges ($k \geq 1$). Each property gives rise to a specialized version of Problem 1 (see Fig. 2 for an example of the different types of stratification):

Problem 2. – **MINPLANARSTRATIFICATION:** Given a straight-line drawing Γ , find a stratification $S(\Gamma, \text{PLANARITY})$ of minimum size.

Problem 3. – MINLACSTRATIFICATION: Given a straight-line drawing Γ and a constant $\alpha \in (0, \frac{\pi}{2}]$, find a stratification $S(\Gamma, \text{LAC}(\alpha))$ of minimum size.

Problem 4. – MIN k -PLANARSTRATIFICATION: Given a straight-line drawing Γ and a constant $k > 0$, find a stratification $S(\Gamma, k\text{-PLANARITY})$ of minimum size.

It is natural to ask what is the complexity of the stratification problems defined above. We prove that they are difficult at least as the well-known problem called *classification*, restricted to planar graphs with maximum vertex degree 4 or 5; this problem is conjectured to be NP-hard [9]. Namely, an *edge coloring* of a graph $G(V, E)$ is an assignment of edge colors such that adjacent edges have different colors. The minimum number of colors of an edge coloring of G is called the *chromatic index* of G . It is known that the chromatic index of a graph is either $\Delta(G)$ or $\Delta(G) + 1$, where $\Delta(G)$ is the maximum vertex degree of G [31]. The classification problem is the problem of deciding whether a graph G has chromatic index $\Delta(G)$ or $\Delta(G) + 1$, and it is NP-complete in general [20]. Restricting the input graph to be planar, the classification problem can be reduced to our stratification problems (we omit details due to space limitations).

Algorithms. Let Γ be a drawing of $G(V, E)$. To solve our different stratification problems on Γ we provide heuristics based on a common unified framework. It exploits an enhanced version of the *crossing graph* of Γ , which is a graph $\chi_\Gamma(V_\chi, E_\chi)$ having a vertex for each edge of Γ , i.e., $V_\chi = E$, and an edge for each pair of crossing edges of Γ , i.e., $E_\chi = \{(e_1, e_2) | e_1, e_2 \in E \text{ and } e_1 \text{ and } e_2 \text{ cross in } \Gamma\}$. In our enhanced version of χ_Γ , we add a weight to each edge $(e_1, e_2) \in E_\chi$, equal to the minimum angle formed by e_1 and e_2 at their crossing point in Γ . Given the one-to-one correspondence between the edges of Γ and the vertices of χ_Γ , an edge stratification $S(\Gamma, \mathcal{P}) = \{E_1, \dots, E_h\}$ corresponds to coloring the vertices of χ_Γ such that the subgraph induced by all vertices with the same color satisfies a property \mathcal{P}' that is the “translation” of \mathcal{P} to the crossing graph. Namely, if $V_\chi^i \subseteq V_\chi$ is the color class associated with E_i ($1 \leq i \leq h$), we have that: (i) $\mathcal{P} = \text{PLANARITY}$ translates into $\mathcal{P}' = \text{INDEPENDENTSET}$: the vertices of V_χ^i form an independent set in χ_Γ ; (ii) $\mathcal{P} = k\text{-PLANARITY}$ translates into $\mathcal{P}' = \text{MAXDEGREE-}k$: the subgraph of χ_Γ induced by V_χ^i has vertex degree at most k ; (iii) $\mathcal{P} = \text{LAC}(\alpha)$ translates into $\mathcal{P}' = \text{EDGEWEIGHT}(\alpha)$: the subgraph of χ_Γ induced by V_χ^i has no edge weight less than α .

Hence, computing a stratification $S(\Gamma, \mathcal{P}) = \{E_1, \dots, E_h\}$ is equivalent to computing a coloring $C(\chi_\Gamma, \mathcal{P}') = \{V_\chi^1, \dots, V_\chi^h\}$ of the vertices of χ , such that the subgraph induced by each V_χ^i satisfies property \mathcal{P}' . In particular, Problem MINPLANARSTRATIFICATION equals to the classical *minimum vertex coloring* problem on χ_Γ , which consists of coloring the vertices of χ_Γ with the minimum number of colors, such that no vertices with the same color are adjacent. Problems MINLACSTRATIFICATION can be reduced to a minimum vertex coloring problem on χ_Γ by applying a pre-processing step that removes from χ_Γ all the edges whose weight is at least α . Problem MIN k -PLANARSTRATIFICATION corresponds to a generalization of the minimum vertex coloring on χ_Γ , which allows each vertex to have at most k adjacent vertices of its same color. Given this strong correlation among all problems, we solve them with a unified framework that is an adaptation of a heuristic for the minimum vertex coloring problem, called *sequential coloring* [5]. It has been shown to be more effective with respect to

other heuristics for the minimum vertex coloring, and can be easily adapted to all our variants of this problem. Our unified heuristic framework works as follows.

Let \mathcal{P}' be the desired property for the subgraph induced by each color class. The vertices $\{v_1, \dots, v_{|E|}\}$ of χ_Γ are processed one per time; the first vertex is assigned to color class V_χ^1 . If vertices v_1, v_2, \dots, v_{i-1} have been assigned to the color classes $V_\chi^1, V_\chi^2, \dots, V_\chi^k$, the next vertex v_i is assigned to the color class V_χ^j , where j is the minimum value for which $V_\chi^j \cup \{v_i\}$ satisfies property \mathcal{P}' ; if no such j exists, v_i is assigned to a new color class V_χ^{k+1} . Several different criteria can be used to choose the next vertex v_i to be processed. We choose vertex v_i with the highest *degree of saturation*, which is the number of different colors assigned to the neighbors of v_i . This strategy has been experimentally proven to give good performance in terms of number of colors used [5]. The time complexity of our heuristic can be evaluated as follows. By using a brute-force approach the crossing graph can be computed in $O(m^2)$, where m is the number of edges of Γ . Using the degree of saturation as the criterium for the vertex selection, the time complexity of the sequential coloring heuristic is $O(N^2 \log N)$, where N is the number of vertices of the graph to be colored. Since in our case $N = m$, the overall time complexity of our heuristic is $O(m^2 \log n)$.

4 User Study

To evaluate the usefulness of our approach, we performed a user study where different interfaces based on edge stratification are compared with an interface where drawings are not stratified. The stratifications were computed with the heuristic framework described in Section 3 and the geometric properties considered were PLANARITY and $\text{LAC}(\frac{\pi}{4})$. In particular, we chose $\frac{\pi}{4}$ as minimum crossing angular resolution, because we observed that this value gives rise to limited number of layers without affecting too much the readability of each layer (see also [12]). Also, in the experiment we decided not to evaluate stratifications obtained for k -PLANARITY for two reasons: (i) comparing too many interfaces would have taken to the users a very long time to complete their test; (ii) there were not significant differences between the sizes of the stratifications $S(\Gamma, \text{PLANARITY})$ and $S(\Gamma, k\text{-PLANARITY})$ (considering small values of k) for the drawings Γ of our benchmark, thus there was no clear advantage in using k -PLANARITY with respect to PLANARITY from the practical point of view. Clearly, this last observation motivates the study of more effective heuristics to compute a stratification $S(G, k\text{-PLANARITY})$, when $k > 0$. Alternatively, we could consider large values of k , however, this would strongly reduce the readability of the layers.

The goal of our study was to address the following two research questions: **(Q1)**. Given a straight-line drawing of a graph, does stratification assist in the reading of the relational information represented by the graph? **(Q2)**. If the first question is settled in the affirmative, is one of the two considered geometric properties (PLANARITY and $\text{LAC}(\frac{\pi}{4})$) more effective in assisting the reading of such relational information?

We performed a within-subjects experiment involving 40 participants. We used 5 different drawings; for each drawing, the participants had to solve 3 different tasks, using 3 different user interfaces. Thus, a trial was represented by the triple $\langle \text{drawing, task, interface} \rangle$ and the number of trials for each participant was $5 \times 3 \times 3 = 45$.

Table 1. The graphs of the user study

<i>graph</i>	<i>vertices</i>	<i>edges</i>	<i>density</i>	<i>stratification layers</i>	
				PLANARITY	LAC($\frac{\pi}{4}$)
lesmis	77	2,148	27.9	4	3
football	115	613	5.33	7	4
gd01	249	635	2.55	6	4
organization	165	726	4.4	7	4
scalefree	204	803	3.94	9	7

Drawings. We chose 5 different complex graphs modeling both real and artificial networks of different type, and we drew them using the OGDF¹ implementation of the multi-level force-directed algorithm FMMM [19]. The chosen graphs are: *lesmis*, a coappearance graph of characters in the novel “Les Miserables” [22]; *football*, a graph of American football games [18]; *gd01*, a graph drawing self-reference network used in the GD 2001 contest [25]; *organization*, a social network modeling the relationships among employees in a private company [24]; *scalefree*, a scale-free network generated using the Barabási-Albert model [2]. For each graph, Table 1 reports the number of vertices, the number of edges, the density (ratio between number of edges and number of vertices), and the number of stratification layers for properties PLANARITY and LAC($\frac{\pi}{4}$).

Tasks. We chose 3 tasks as representative of the possible tasks involving graph reading. We aimed at having tasks significantly different one to another, easy to understand by non-expert users, and requiring both local and global explorations of the drawing. We considered: the *Shortest Path (SP)* task, which asks “How long is the shortest path between the two highlighted vertices?”; the *Common Adjacent (CO)* task, which asks “What is the number of adjacent vertices shared by the two highlighted vertices?”; and the *Degree (DE)* task, which asks “What is the degree of the highlighted vertex?”. Similar tasks are used in other experiments on graph reading (see, e.g., [28]). The vertices highlighted for each task were chosen without looking at the stratification layers.

User interfaces. The three user interfaces differ from one another only by the geometric property used to compute a stratification. The *Planar Stratification (PS)* interface uses PLANARITY; the *LAC Stratification (LS)* interface uses LAC($\frac{\pi}{4}$); the *Overview (OV)* interface does not use any stratification (there is only one layer). To limit the number of layers in PS and LS, we halted the edge partitioning process when 80% of the edges of the drawing were placed in some layer. The remaining edges were all put in an additional layer, for which no geometric property is guaranteed. In every interface the edges of each layer were displayed with the same color and we used a different color for the edges of each layer. We chose the colors so to maximize the readability over a black background and so that different colors can be easily distinguished. Every interface allowed the user to select any combination of the available layers, so that only the edges in the selected layers were displayed on the screen; the edges in the non-selected

¹ <http://www.ogdf.net>

layers were sketched as transparent light gray segments ($\alpha=0.2$). In this way, the user could focus on the selected layers only, still keeping in mind that the remaining edges were part of the drawing and should possibly be considered to properly solve the task. Thus, the user's strategy to explore the drawing varied from looking at one single layer per time to looking at the drawing as a whole. Also, the vertices were always drawn as white circles except the highlighted vertices of each trial, which were drawn red and slightly larger. Such a consistent environment did not require to the user any cognitive shift to move from one interface to another. Finally, the participants were not aware of the criteria used to stratify the drawings.

Experimental Procedure. Before starting the experiments, the participants received a brief tutorial introducing the basic concepts on graphs. Also, an explanation of the tasks and of the user interfaces was given with practical examples. The 45 trials were preceded by 4 training trials whose results were not taken into account, although the participants were not aware of it. Regardless of the task, participants had to answer each question by entering a number in a text box, with no time limit. However, they were asked to spend at most 3 minutes for each question. In order to counter the learning effect, in each experiment the 45 stimuli appeared in a randomized order, and the system randomly flipped and rotated each drawing. Between a question and the next one, participants could take a short break, without the possibility of exchanging information.

Results. 40 volunteering students (with age from 19 to 25) in Computer Engineering took part in the experiments. We recorded their answers and the time spent for each question. We compared the performance of OV, PS, and LS in terms of absolute error rate (the absolute value of the difference between the user answer and the correct answer) and response time. First of all, we performed a Shapiro-Wilk test ($\alpha=0.05$) to determine whether the data was normally distributed or not. We found that none of the considered populations was normally distributed. Thus, we performed a non-parametric analysis exploiting repeated measures Friedman tests ($\alpha=0.05$), with post-hoc pairwise comparisons. We applied Bonferroni corrections on the pairwise comparisons setting $\alpha < 0.017$. We obtained these results (see Table 2). (i) Considering all the tasks, both PS and LS significantly outperformed OV in terms of absolute error rate. This improvement in the accuracy comes together with a slower response time: PS and LS show slower performance than OV, and LS is faster than PS. (ii) For task SP, the mean absolute error rate of LS is smaller than the one of OV and PS, although such a difference turns out to be not statistically significant. In terms of response time, again, OV is faster than PS and LS, while LS is faster than PS. (iii) For task CA, the three interfaces led to comparable performance in terms of absolute error rate. In terms of response time, the situation is similar to the previous cases, OV is faster than PS and LS, while LS is faster than PS. (iv) For task DE, both PS and LS outperformed OV in terms of absolute error rate, and the difference turns out to be statistically significant. Also, PS led to more accurate results than LS, still with statistical significance. About the response time, OV and LS behave similarly and slightly faster than PS.

Discussion. The results of the user study show an improvement in terms of accuracy in the reading of the displayed graphs when using stratification. The PS interface outperformed both LS and OV for most of the considered tasks. We conclude that stratifying

Table 2. Results of the user study. The mean values and the pairwise significance between each user interface are shown for absolute error rate and time

		Overall	SP	CA	DE			Overall	SP	CA	DE
Abs. Error Rate	mean OV	2.74	1.84	1.93	4.43	Time (sec.)	mean OV	54.36	60.47	61.23	41.36
	mean PS	1.87	1.79	1.96	1.86		mean PS	80.01	91.78	98.26	50.01
	mean LS	1.92	1.57	1.98	2.22		mean LS	68.73	77.46	85.71	43.00
	OV vs PS	< .001	n.s.	n.s.	< .001		OV vs PS	< .001	< .001	< .001	.001
	OV vs LS	< .001	n.s.	n.s.	< .001		OV vs LS	< .001	< .001	< .001	n.s.
	PS vs LS	n.s.	n.s.	n.s.	.010		PS vs LS	< .001	.002	.001	< .001

the drawing into planar layers gave a significant help to the participants. The task that received more significant advantage from both PS and LS is task DE. Indeed, counting the degree of a vertex can be quite hard when the drawing is cluttered around the highlighted vertex; on the other hand, by selecting a layer per time, one can effectively cope with such clutter and the partial degree of the vertex can be easily counted, at the expenses of a negligible increase of the response time (less than 10 seconds in the average). Moreover, according to this strategy, having planar layers guarantees that the region of the drawing around the highlighted vertex is crossing free, and hence clearer.

5 Comparison of Graph Drawing Algorithms

We describe a second experiment, which compares several force-directed algorithms to answer the following research questions: **(Q3)**. What is the force-directed drawing algorithm for which the computed layouts require the smallest number of layers with properties PLANARITY and $LAC(\frac{\pi}{4})$, when stratified with our heuristics? **(Q4)**. How are the number of crossings and the number of layers correlated?

We tested 5 different force-directed algorithms on a benchmark of 105 graphs, thus collecting 525 drawings; for each drawing Γ , we measured its number of crossings, the size of a stratification $S(\Gamma, \text{PLANARITY})$, and the size of a stratification $S(\Gamma, LAC(\frac{\pi}{4}))$ computed by our heuristics. Observe that, since the size of the crossing graph can be $\Omega(n^4)$, coloring this graph exactly is often prohibitive even for small graphs.

Algorithms. We tested the whole set of force-directed algorithms available in the OGDF library: Kamada-Kawai (KK) [21], Fruchterman-Reingold (FR) [17], GEM (GEM) [16], FM³ (FMMM) [19], and Stress majorization (SM) [4]. Since tuning is a critical issue for force-directed techniques, it is worth remarking that we initialized the algorithms by using the default parameters set by their implementations in the library.

Benchmark. We ran the drawing algorithms on a benchmark of 105 complex graphs, organized in three groups: `UniformRandGraphs`, containing 40 random graphs generated with a uniform probability distribution; for each $n \in \{100, 200, \dots, 400\}$ we generated 10 graphs with density in the interval $[2, 6]$. `ScaleFreeRandGraphs`, containing 60 small-world and scale-free graphs generated with the LFR algorithm [23], already used to generate graphs in previous extensive experimental works [13]; for each

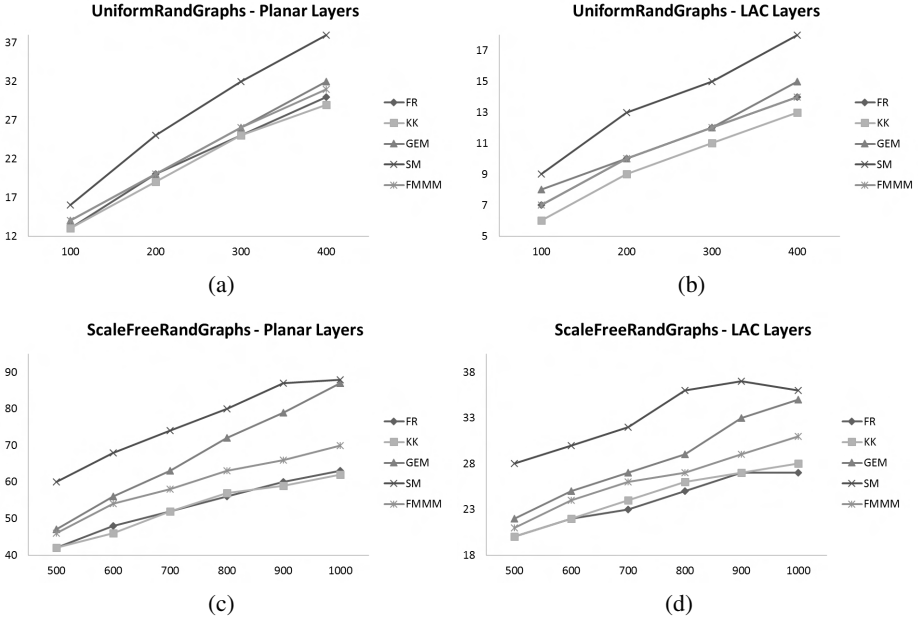


Fig. 3. Mean number of planar and LAC($\frac{\pi}{4}$) layers on UniformRandGraphs (a-b) and ScaleFreeRandGraphs (c-d). The x -axis reports the number of vertices.

$n \in \{500, 600, \dots, 1000\}$ we generated 10 graphs with density in the interval $[4, 8]$. UserStudyGraphs, which contains the 5 graphs used in the user study.

Results. (i) On UniformRandGraphs, the layouts of FR, GEM, and FMMM require a comparable number of layers regardless the geometric property. KK outperforms the other algorithms for LAC($\frac{\pi}{4}$) layers, while SM always led to the highest number of layers. See Figs. 3(a) and 3(b). (ii) On ScaleFreeRandGraphs, the layouts of FR and KK most frequently require the smallest number of layers, regardless the geometric property; FMMM led to slightly bigger numbers, while again SM led to the largest number of layers. Finally, GEM behaves similarly to FMMM for small values of n , and approaches SM as n grows. See Figs. 3(c) and 3(d). (iii) On UserStudyGraphs, the layouts computed by FR are those that more often require the smallest number of layers. KK and FMMM behave similarly, GEM is slightly worse, and SM most frequently led to the largest number of layers. We omit the charts due to space limitations.

To evaluate the correlation between layers and crossings, we executed a Kendall's τ test ($\alpha < 0.01$). The test showed a strong correlation ($0.9 < r < 0.96$) between number of crossings and number of layers (both planar and LAC($\frac{\pi}{4}$)), for all the algorithms.

Discussion. Concerning question Q3, the results show that FR and KK have the best performance in terms of number of layers required to stratify their layouts (both planar and LAC($\frac{\pi}{4}$)), while SM is always the worst for the graphs in our benchmark. About question Q4, we interpret the strong correlation between the number of crossings and

the number of layers as a positive feature of the drawing algorithms, which witnesses a quite uniform distribution of the crossings in the drawing. We also observe that, for some instances, there are pairs of drawings $\langle \Gamma_1, \Gamma_2 \rangle$ computed by different algorithms such that Γ_1 has more crossings but less layers than Γ_2 . By visually inspecting these instances, the crossings in the drawings with fewer layers appear to be more evenly distributed, resulting in a more readable drawing in spite of the greater number of crossings. This seems to confirm our intuition that the number of layers is also related to the distribution of the crossings in the drawing area, and suggests that the number of layers could be a more reliable measure of the drawing visual complexity with respect to the number of crossings. Note that just measuring the crossing spatial distribution in a drawing does not necessarily yield to similar conclusions; indeed, a drawing with low average crossing spatial distribution may still contain very cluttered spots surrounded by crossing-free regions, which might cause a high number of layers.

6 Conclusions and Future Research Directions

Our framework, based on the use of the crossing graph, is not suited for very large graphs. The number of crossings can be $\Omega(n^4)$ and hence, even for relatively small complex drawings, the crossing graph can be so large that the performance of our heuristic degrades both in terms of space and time. It would be useful to devise more efficient heuristics that do not make use of crossing graphs. Also, in our user interface we presented the different layers using colors and allowing users to select any subset of layers. It would be interesting to design new visualization paradigms to effectively present stratified drawings. A 2.5D visualization could be an interesting option to explore.

References

1. Auber, D., Chiricota, Y., Jourdan, F., Melançon, G.: Multiscale visualization of small world networks. In: *InfoVis 2003*, pp. 75–81. IEEE (2003)
2. Barabasi, A.-L., Albert, R.: Emergence of Scaling in Random Networks. *Science* 286(5439), 509–512 (1999)
3. Batagelj, V., Brandenburg, F., Didimo, W., Liotta, G., Palladino, P., Patrignani, M.: Visual analysis of large graphs using (X,Y)-clustering and hybrid visualizations. *IEEE TVCG* 17(11), 1587–1598 (2011)
4. Brandes, U., Pich, C.: More flexible radial layout. In: Eppstein, D., Gansner, E.R. (eds.) *GD 2009*. LNCS, vol. 5849, pp. 107–118. Springer, Heidelberg (2010)
5. Brélez, D.: New methods to color the vertices of a graph. *Comm. ACM* 22, 251–256 (1979)
6. Bruckdorfer, T., Cornelsen, S., Gutwenger, C., Kaufmann, M., Montecchiani, F., Nöllenburg, M., Wolff, A.: Progress on partial edge drawings. In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 67–78. Springer, Heidelberg (2013)
7. Brunel, E., Gamsa, A., Krug, M., Rutter, I., Wagner, D.: Generalizing geometric graphs. In: Speckmann, B. (ed.) *GD 2011*. LNCS, vol. 7034, pp. 179–190. Springer, Heidelberg (2011)
8. Buchheim, C., Chimani, M., Gutwenger, C., Jünger, M., Mutzel, P.: Crossings and planarization. In: Tamassia, R. (ed.) *Handbook of Graph Drawing and Visualization*. CRC Press (2013)
9. Chrobak, M., Nishizeki, T.: Improved edge-coloring algorithms for planar graphs. *J. Algo.* 11(1), 102–116 (1990)

10. Di Giacomo, E., Didimo, W., Liotta, G., Montecchiani, F.: h -quasi planar drawings of bounded treewidth graphs in linear area. In: Golubic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) WG 2012. LNCS, vol. 7551, pp. 91–102. Springer, Heidelberg (2012)
11. Di Giacomo, E., Didimo, W., Liotta, G., Montecchiani, F.: Area requirement of graph drawings with few crossings per edge. *Comp. Geom.* 46(8), 909–916 (2013)
12. Didimo, W., Liotta, G.: The crossing angle resolution in graph drawing. In: Pach, J. (ed.) *Thirty Essays on Geometric Graph Theory*. Springer (2012)
13. Didimo, W., Montecchiani, F.: Fast layout computation of hierarchically clustered networks: Algorithmic advances and experimental analysis. In: IV 2012, pp. 18–23 (2012)
14. Dillencourt, M.B., Eppstein, D., Hirschberg, D.S.: Geometric thickness of complete graphs. *Jour. Graph. Alg. and Appl.* 4(3), 5–17 (2000)
15. Duncan, C.A., Eppstein, D., Kobourov, S.G.: The geometric thickness of low degree graphs. In: SoCG 2004, pp. 340–346. ACM (2004)
16. Frick, A., Ludwig, A., Mehldau, H.: A fast adaptive layout algorithm for undirected graphs. In: Tamassia, R., Tollis, I.G. (eds.) GD 1994. LNCS, vol. 894, pp. 388–403. Springer, Heidelberg (1995)
17. Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. *Softw. Pract. Exper.* 21(11), 1129–1164 (1991)
18. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. *PNAS* 99(12), 7821–7826 (2002)
19. Hachul, S., Jünger, M.: Drawing large graphs with a potential-field-based multilevel algorithm. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 285–295. Springer, Heidelberg (2005)
20. Holyer, I.: The NP-completeness of edge-coloring. *SIAM J. Comput.* 10(4), 718–720 (1981)
21. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. *Inf. Process. Lett.* 31(1), 7–15 (1989)
22. Knuth, D.E.: *The Stanford Graphbase: A Platform for Combinatorial Computing*. Addison-Wesley Professional (1993)
23. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. *Phys. Rev. E* 78(4) (2008)
24. Michael Fire, Y.E., Puzis, R.: Organization mining using online social networks (2012), <http://proj.ise.bgu.ac.il/sns>
25. Mutzel, P., Jünger, M., Leipert, S. (eds.): GD 2001. LNCS, vol. 2265. Springer, Heidelberg (2002)
26. Purchase, H.C.: Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interact. Comput.* 13(2), 147–162 (2000)
27. Purchase, H.C., Carrington, D.A., Alder, J.-A.: Empirical evaluation of aesthetics-based graph layout. *Empir. Softw. Eng.* 7(3), 233–255 (2002)
28. Purchase, H.C., Hamer, J., Nöllenburg, M., Kobourov, S.G.: On the usability of lombardi graph drawings. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 451–462. Springer, Heidelberg (2013)
29. Shneiderman, B., Dunne, C.: Interactive network exploration to derive insights: Filtering, clustering, grouping, and simplification. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 2–18. Springer, Heidelberg (2013)
30. van Ham, F., van Wijk, J.J.: Interactive visualization of small world graphs. In: *InfoVis 2004*, pp. 199–206. IEEE (2004)
31. Vizing, V.G.: On an estimate of the chromatic class of a p -graph. *Diskret. Analiz* No. 3, 25–30 (1964)
32. Zhou, H., Xu, P., Yuan, X., Qu, H.: Edge bundling in information visualization. *Tsinghua Science and Technology* 18(2), 145–156 (2013)

Drawing Planar Graphs with a Prescribed Inner Face

Tamara Mchedlidze, Martin Nöllenburg, and Ignaz Rutter

Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Germany

Abstract. Given a plane graph G (i.e., a planar graph with a fixed planar embedding) and a simple cycle C in G whose vertices are mapped to a convex polygon, we consider the question whether this drawing can be extended to a planar straight-line drawing of G . We characterize when this is possible in terms of simple necessary conditions, which we prove to be sufficient. This also leads to a linear-time testing algorithm. If a drawing extension exists, it can be computed in the same running time.

1 Introduction

The problem of extending a partial drawing of a graph to a complete one is a fundamental problem in graph drawing that has many applications, e.g., in dynamic graph drawing and graph interaction. This problem has been studied most in the planar setting and often occurs as a subproblem in the construction of planar drawings.

The earliest example of such a drawing extension result are so-called Tutte embeddings. In his seminal paper “How to Draw a Graph” [10], Tutte showed that any triconnected planar graph admits a convex drawing with its outer vertices fixed to an arbitrary convex polygon. The strong impact of this fundamental result is illustrated by its, to date, more than 850 citations and the fact that it received the “Best Fundamental Paper Award” from GD’12. The work of Tutte has been extended in several ways. In particular, it has been strengthened to only require polynomial area [4], even in the presence of collinear points [3]. Hong and Nagamochi extended the result to show that triconnected graphs admit convex drawings when their outer vertices are fixed to a star-shaped polygon [5]. For general subdrawings, the decision problem of whether a planar straight-line drawing extension exists is NP-hard [9]. Pach and Wenger [8] showed that every subdrawing of a planar graph that fixes only the vertex positions can be extended to a planar drawing with $O(n)$ bends per edge and that this bound is tight. The drawing extension problem has also been studied in a topological setting, where edges are represented by non-crossing curves. In contrast to the straight-line variant, it can be tested in linear time whether a drawing extension of a given subdrawing exists [1]. Moreover, there is a characterization via forbidden substructures [6].

In this paper, we study the problem of finding a planar straight-line drawing extension of a plane graph for which an arbitrary cycle has been fixed to a convex polygon. It is easy to see that a drawing extension does not always exist in this case; see Fig. 1(a). Let G be a plane graph and let C be a simple cycle of G represented by a convex polygon Γ_C in the plane. The following two simple conditions are clearly necessary for the existence of a drawing extension: (i) C has no chords that must be embedded outside of C and (ii) for every vertex v with neighbors on C that must be embedded outside of

C there exists a placement of v outside Γ_C such that the drawing of the graph induced by C and v is plane and bounded by the same cycle as in G . We show in this paper that these two conditions are in fact sufficient. Both conditions can be tested in linear time, and if they are satisfied, a corresponding drawing extension can be constructed within the same time bound.

Our paper starts with some necessary definitions (Section 2) and useful combinatorial properties (Section 3). The idea of our main result has two steps. We first show in Section 4 that the conditions are sufficient if Γ_C is one-sided (i.e., it has an edge whose incident inner angles are both less than 90°). Afterward, we show in Section 5 that, for an arbitrary convex polygon Γ_C , we can place the neighborhood of C in such a way that the drawing is planar, and such that the boundary C' of its outer face is a one-sided polygon $\Gamma_{C'}$. Moreover, our construction ensures that the remaining graph satisfies the conditions for extendability of $\Gamma_{C'}$. The general result then follows directly from the one-sided case.

2 Definitions and a Necessary Condition

Plane Graphs and Subgraphs. A graph $G = (V, E)$ is *planar* if it has a drawing Γ in the plane \mathbb{R}^2 without edge crossings. Drawing Γ subdivides the plane into connected regions called *faces*; the unbounded region is the *outer* and the other regions are the *inner* faces. The boundary of a face is called *facial cycle*, and *outer cycle* for the outer face. The cyclic ordering of edges around each vertex of Γ together with the description of the external face of G is called an *embedding* of G . A graph G with a planar embedding is called *plane graph*. A *plane subgraph* H of G is a subgraph of G together with a planar embedding that is the restriction of the embedding of G to H .

Let G be a plane graph and let C be a simple cycle of G . Cycle C is called *strictly internal*, if it does not contain any vertex of the outer face of G . A chord of C is called *outer* if it lies outside C in G . A cycle without outer chords is called *outerchordless*. The *subgraph of G inside C* is the plane subgraph of G that is constituted by vertices and edges of C and all vertices and edges of G that lie inside C .

Connectivity. A graph G is *k-connected* if removal of any set of $k - 1$ vertices of G does not disconnect the graph. For $k = 2, 3$ a *k-connected* graph is also called *biconnected* and *triconnected*, respectively. An internally triangulated plane graph is triconnected if and only if there is no edge connecting two non-consecutive vertices of its outer cycle (see, for example, [2]).

Star-shaped and One-sided Polygons. Let Π be a polygon in the plane. Two points inside or on the boundary of Π are mutually *visible*, if the straight-line segment connecting them belongs to the interior of Π . The *kernel* $K(\Pi)$ of polygon Π is the set of all the points inside Π from which all vertices of Π are visible. We say that Π is *star-shaped* if $K(\Pi) \neq \emptyset$. We observe that the given definition of a star-shape ensures that its kernel has a positive area.

A convex polygon Π with k vertices is called *one-sided*, if there exists an edge e (i.e., a line segment) of Π such that the orthogonal projection to the line supporting e

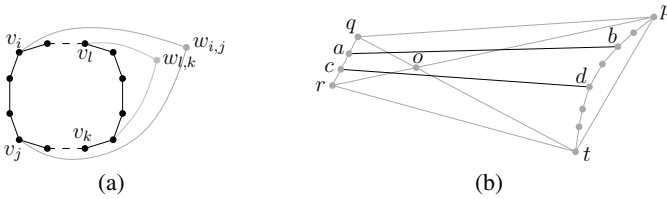


Fig. 1. Convex polygon of cycle C is denoted by black. Vertex $w_{i,j}$ cannot be placed on the plane without changing the embedding or intersecting C . Vertices $w_{i,j}$ and $w_{l,k}$ are petals of C , where $w_{l,k} \prec w_{i,j}$. Petal $w_{l,k}$ is realizable, while petal $w_{i,j}$ is not. (b) Illustration of Fact 1.

maps all polygon vertices actually onto segment e . Then e is called the *base edge* of Π . Without loss of generality let $e = (v_1, v_k)$ and v_1, \dots, v_k be the clockwise ordered sequence of vertices of Π .

Extension of a Drawing. Let G be a plane graph and let H be a plane subgraph of G . Let Γ_H be a planar straight-line drawing of H . We say that Γ_H is *extendable* if drawing Γ_H can be completed to a planar straight-line drawing Γ_G of the plane graph G . Then Γ_G is called an *extension* of Γ_H . A planar straight-line drawing of G is called *convex*, if every face of G (including the outer face) is represented as a convex polygon.

The following theorem by Hong and Nagamochi [5] shows the extendability of a prescribed star-shaped outer face of a plane graph.

Theorem 1 (Hong, Nagamochi [5]). *Every drawing of the outer face f of a 3-connected graph G as a star-shaped polygon can be extended to a planar drawing of G , where each internal face is represented by a convex polygon. Such a drawing can be computed in linear time.*

Petals and Stamens. Let G be a plane graph, and let P_{uv} be a path in G between vertices u and v . Its subpath from vertex a to vertex b is denoted by $P_{uv}[a, b]$. Let C be a simple cycle of G , and let v_1, \dots, v_k be the vertices of C in clockwise order. Given two vertices v_i and v_j of C , we denote by $C[v_i, v_j]$ the subpath of C encountered when we traverse C clockwise from v_i to v_j . Assume that C is represented by a convex polygon Γ_C in the plane. We say that a vertex $v_i, 1 \leq i \leq k$ of Γ_C is *flat*, if $\angle v_{i-1}v_i v_{i+1} = \pi$. Throughout this paper, we assume that convex polygons do not have flat vertices.

A vertex $w \in V(G) \setminus V(C)$ adjacent to at least two vertices of C and lying outside C in G , is called a *petal* of C (see Figure 1(a)). Consider the plane subgraph G' of G induced by the vertices $V(C) \cup \{w\}$. Vertex w appears on the boundary of G' between two vertices of C , i.e. after some $v_i \in V(C)$ and before some $v_j \in V(C)$ in clockwise order. To indicate this fact, we will denote petal w by $w_{i,j}$. Edges $(w_{i,j}, v_i)$ and $(w_{i,j}, v_j)$ are called the *outer edges* of petal $w_{i,j}$. The subpath $C[v_i, v_j]$ of C is called *base* of the petal $w_{i,j}$. A vertex v_f is called *internal*, if it appears on C after v_i and before v_j in clockwise order. A petal $w_{i,i+1}$ is called *trivial*. A vertex of $V(G) \setminus V(C)$ adjacent to exactly one vertex of C is called a *stamen* of C .

Let v be a petal of C and let u be either a petal or a stamen of C , we say that u is *nested* in v , and denote this fact by $u \prec v$, if u lies in the cycle delimited by the base

and the outer edges of petal v . For two stamens u and v , neither $u \prec v$ nor $v \prec u$. So for each pair of stamens or petals u and v we have either $u \prec v$, or $v \prec u$, or none of these. This relation \prec is a partial order. A petal or a stamen u of C is called *outermost* if it is maximal with respect to \prec .

Necessary Petal Condition. Let again G be a plane graph and let C be an outerchordless cycle of G represented by a convex polygon Γ_C in the plane. Let $w_{i,j}$ be a petal of C . Let G' be the plane subgraph of G , induced by the vertices $V(C) \cup \{w_{i,j}\}$. We say that $w_{i,j}$ is *realizable* if there exists a planar drawing of G' which is an extension of Γ_C . This gives us the necessary condition that Γ_C is extendable only if each petal of C is realizable. In the rest of the paper we prove that this condition is sufficient.

3 Combinatorial Properties of Graphs and Petals

In this section, we derive several properties of petals in graphs, which we use throughout the construction of the drawing extension in the remaining parts of this paper. Due to space constraints the proofs can be found in the full version of this paper [7]. Proposition 1 allows us to restrict our attention to maximal plane graphs for which the given cycle C is strictly internal. The remaining lemmas are concerned with the structure of the (outermost) petals of C in such a graph.

Proposition 1. *Let G be a plane graph on n vertices and let C be a simple outerchordless cycle of G . There exists a plane supergraph G' of G with $O(n)$ vertices such that (i) G' is maximal, (ii) there are no outer chords of C in G' , (iii) each petal of G' with respect to C is either trivial or has the same neighbors on C as in G , and (iv) C is strictly internal to G .*

In the following we assume that our given plane graph is maximal, and the given cycle is strictly internal, otherwise Proposition 1 is applied.

Lemma 1. *Let G be a triangulated planar graph with a strictly internal outerchordless cycle C . Then the following statements hold. (i) Each vertex of C that is not internal to an outermost petal is adjacent to two outermost petals. (ii) There is a simple cycle C' whose interior contains C and that contains exactly the outermost stamens and petals of C .*

Lemma 2. *Let G be a maximal planar graph with a strictly internal outerchordless cycle C . Let u and v be two adjacent vertices on C that are not internal to the same petal. Then there exists a third vertex w of C such that there exist three chordless disjoint paths from u, v and w to the vertices of the outer face of G such that none of them contains other vertices of C .*

4 Extension of a One-sided Polygon

Let G be a plane graph, and let C be a simple outerchordless cycle, represented by a one-sided polygon Γ_C . In this section, we show that if each petal of C is realizable,

then Γ_C is extendable to a straight-line drawing of G . This result serves as a tool for the general case, which is shown in Section 5.

The drawing extension we produce preserves the outer face, i.e., if the extension exists, then it has the same outer face as G . It is worth mentioning that, if we are allowed to change the outer face, the proof becomes rather simple, as the following claim shows.

Claim 1. *Let G be a maximal plane graph and let C be an outerchordless cycle of G , represented in the plane by a one-sided polygon Γ_C . Then drawing Γ_C is extendable.*

Proof. Let (v_1, v_k) be the base edge of Γ_C . Edge (v_1, v_k) is incident to two faces of G , to a face f_{in} inside C and to a face f_{out} outside C . We select f_{out} as the outer face of G . With this choice, edge (v_1, v_k) is on the outer face of G . Let v be the third vertex of this face. We place the vertex v far enough from Γ_C , so that all vertices of Γ_C are visible from v . Thus, we obtain a planar straight-line drawing of the subgraph G_v induced by the vertices $V(C) \cup \{v\}$, such that each face is represented by a star-shaped polygon. Each subgraph of G inside a face of G_v is triconnected, and therefore, we can complete the existing drawing to a straight-line planar drawing of G , by Theorem 1. \square

In the rest of the section we show that extendability of Γ_C can be efficiently tested, even if the outer face of G has to be preserved. The following simple geometric fact will be used in the proof of the result of this section (see Figure 1(b) for the illustration).

Fact 1. *Let $pqrt$ be a convex quadrilateral and let o be the intersection of its diagonals. Let S_{pt} be a one-sided convex polygon with base \overline{pt} , that lies inside triangle $\triangle opt$. Let \overline{ab} and \overline{cd} be such that $b, d \in S_{pt}$, ordered clockwise as t, d, b, p and $a, c \in \overline{qr}$, ordered as q, a, c, r . Then, neither \overline{ab} and \overline{cd} intersect each other, nor do they intersect a segment between two consecutive points of S_{pt} .*

We are now ready to prove the main result of this section.

Theorem 2. *Let G be a maximal plane graph and C be a strictly internal simple outerchordless cycle of G , represented in the plane by a one-sided polygon Γ_C . If every petal of C is realizable, then Γ_C is extendable.*

Proof. Let v_1, \dots, v_k be the clockwise ordering of the vertices of C , so that (v_1, v_k) is the base of Γ_C . We rotate Γ_C so that (v_1, v_k) is horizontal. Let a, b, c be the vertices of the external face of G , in clockwise order, see Fig. 2. By Lemma 2, there exists a vertex v_j , $1 < j < k$, such that there exist chordless disjoint paths between v_1, v_j, v_k , and the vertices a, b, c , respectively. Without loss of generality assume they are P_{v_1a} , P_{v_jb} and P_{v_kc} . Some vertices of P_{v_1a} and P_{v_kc} are possibly adjacent to each other, as well as to the boundary of C . Depending on these adjacencies, we show how to draw the paths P_{v_1a} , P_{v_kc} and how to place vertex b , so that the graph induced by these vertices and cycle C is drawn with star-shaped faces. Then, the drawing of G can be completed by applying Theorem 1. Let v_i be the topmost vertex of Γ_C . It can happen that there are two adjacent topmost vertices v_i and v_{i+1} . However, v_{i-1} and v_{i+2} are lower, since Γ_C does not contain flat vertices. In the following, we assume that v_i and v_{i+1} have the same y -coordinate. The case when v_i is unique can be seen as a special case where $v_i = v_{i+1}$. Without loss of generality assume that $i + 1 \leq j \leq k - 1$, the case where $2 \leq j \leq i$

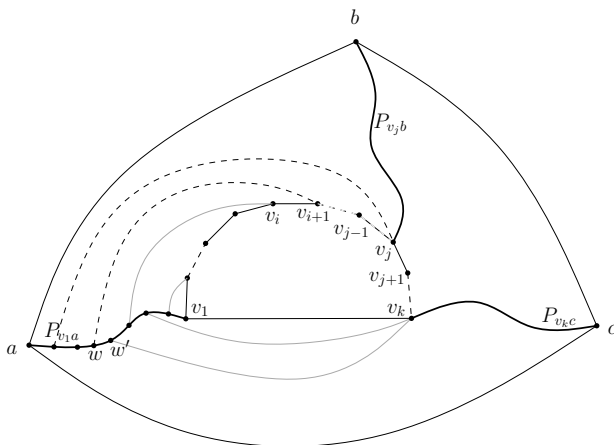


Fig. 2. Illustration for the proof of Theorem 2. Edges between $C[v_1, v_i]$ and $P_{v_1 a}[v_1, w'] \cup \{v_k\}$ are gray. Edges between $C[v_{i+1}, v_j]$ and $P_{v_1 a}[w, a]$ are dashed.

is treated symmetrically. Notice that the presence of the path $P_{v_j b}$ ensures that edges between vertices of $P_{v_1 a}$ and $P_{v_k c}$ can only lie in the interior of the cycle delimited by these paths and edges (v_1, v_k) and (a, c) (refer to Figure 2). Consider a vertex of $P_{v_1 a}$ which is a petal of C . The base of such a petal cannot contain edge (v_{k-1}, v_k) , since this would cause a crossing with $P_{v_k c}$. Moreover, if the base of this petal contains edge (v_1, v_k) , then it cannot contain any edge (v_f, v_{f+1}) for $i \leq f < j$, since otherwise this petal would not be realizable. Thus a vertex of $P_{v_1 a}$ is either adjacent to v_k or to a vertex v_f , where $i + 1 \leq f \leq j$, but not both. It is worth mentioning that a vertex of $P_{v_1 a}$ cannot be adjacent to any v_f , $j + 1 \leq f \leq k - 1$, since such an adjacency would cause a crossing either with $P_{v_j b}$ or with $P_{v_k c}$.

Let ℓ_a , ℓ and ℓ_c be three distinct lines through v_j that lie clockwise between the slopes of edges (v_{j-1}, v_j) and (v_j, v_{j+1}) (see Figure 3). Such lines exist since Γ_C does not contain flat vertices. Let ℓ_i be the line through v_i with the slope of (v_{i-1}, v_i) . Let ℓ_a^1 be the half-line originating at an internal point of (v_1, v_k) towards $-\infty$, slightly rotated counterclockwise from the horizontal position, so that it crosses ℓ_i . Let q denote the intersection point of ℓ_a^1 and ℓ_i . Let p be any point on ℓ_a^1 further away from v_1 than q . Let ℓ_a^2 be the line through p with the slope of ℓ . By construction of lines ℓ_a , ℓ and ℓ_c , line ℓ_a^2 crosses ℓ_a above the polygon Γ_C at point p_a and line ℓ_c below this polygon at point p_c .

Let G' be the plane subgraph of G induced by the vertices of C , $P_{v_1 a}$, and $P_{v_k c}$. The outer cycle of G' consists of edge (a, c) and a path P_{ac} between vertices a and c .

Claim 2. *The vertices of $P_{v_1 a}$ and $P_{v_k c}$ can be placed on lines ℓ_a^1 , ℓ_a^2 and ℓ_c such that in the resulting straight-line drawing of G' , path P_{ac} is represented by an x -monotone polygonal chain, and the inner faces of G' are star-shaped polygons.*

The vertices of $P_{v_1 a}$ will be placed on line ℓ_a^1 between points p and q and on line ℓ_a^2 above point p_a . The vertices of $P_{v_k c}$ will be placed on ℓ_c below p_c . In order to place

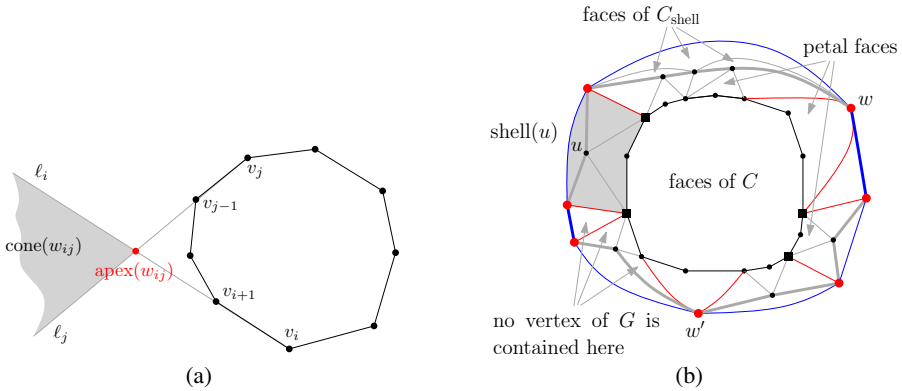


Fig. 4. (a) Vertex $w_{i,j}$ is the petal of C with base $C[v_i, v_j]$. Point $\text{apex}(w_{i,j})$ is red, region $\text{cone}(w_{i,j})$ is gray. (b) Graph G_{shell} . Polygon Γ_C is black. Cycle C_{shell} is bold gray. Cycle C'_{shell} is blue. Graph G'_{shell} is comprised by blue, red and black edges. Vertices of B are squares.

drawn without crossings. Edges between $P_{v_k c}$ and $P_{v_1 a}$ cross neither $P_{v_1 a}$, nor (v_1, v_k) by the choice of lines ℓ_c and ℓ_a^2 .

We have constructed a planar straight-line drawing of G' . We notice that path P_{ac} is drawn as an x -monotone polygonal chain. We also notice that the faces of G' , created when placing vertices of $P_{v_1 a}$ (resp. $P_{v_k c}$) are star-shaped and have their kernels arbitrarily close to the vertices of $P_{v_1 a}$ (resp. $P_{v_k c}$).

Notice that vertex b is possibly adjacent to some of the vertices of P_{ac} . Thus, placing b at an appropriate distance above P_{ac} , the edges between b and P_{ac} can be drawn straight-line without intersecting P_{ac} and therefore no other edge of G' . The faces created when placing b are star-shaped and have their kernels arbitrarily close to b . We finally apply Theorem 1. □

5 Main Theorem

Let G be a maximal plane graph and C be a strictly internal simple outerchordless cycle of G , represented by an arbitrary convex polygon Γ_C in the plane. In Theorem 3 we prove that it is still true that if each petal of C is realizable, then Γ_C is extendable. Before stating and proving Theorem 3, we introduce notation that will be used through this section.

Recall that v_1, \dots, v_k denote the vertices of C . Let $w_{i,j}$ be an outermost petal of C in G . Let ℓ_i (resp. ℓ_j) be a half-line with the slope of edge (v_i, v_{i+1}) (resp. (v_{j-1}, v_j)) originating at v_i (resp. v_j) (see Figure 4(a)). Since $w_{i,j}$ is realizable, lines ℓ_i and ℓ_j intersect. Denote by $\text{apex}(w_{i,j})$ their intersection point and by $\text{cone}(w_{i,j})$ the subset of \mathbb{R}^2 that is obtained by the intersection of the half-planes defined by ℓ_i and ℓ_j , not containing Γ_C . It is clear that any internal point of $\text{cone}(w_{i,j})$ is appropriate to draw $w_{i,j}$ so that the plane subgraph of G induced by $V(C) \cup \{w_{i,j}\}$ is crossing-free. For consistency, we also define $\text{cone}(w)$ and $\text{apex}(w)$ of an outer stamen w of C as follows.

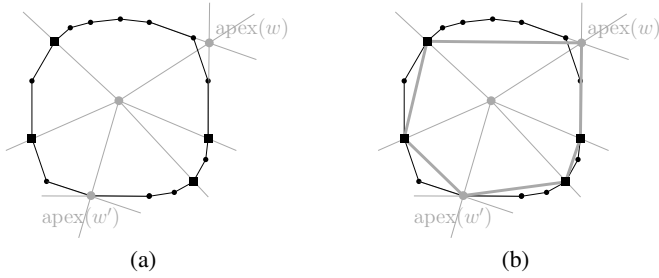


Fig. 5. Construction of drawing of graph G_{shell} shown in Figure 4(b). (a) Apex points are gray, points of B are black squares. (b) Polygon Π is gray, lines $\{\ell(w) \mid w \in S \cap C'_{\text{shell}}\}$ are dashed.

Assume that w is adjacent to $v_i \in C$. Then $\text{cone}(w) \subset \mathbb{R}^2$ is the union of the half-planes defined by lines of edges (v_{i-1}, v_i) and (v_i, v_{i+1}) , that do not contain Γ_C . We set $\text{apex}(w) = v_i$.

Let P (resp. S) denote the set of outermost petals (resp. stamens) of C in G . By Lemma 1, there exists a cycle C_{shell} in G that contains exactly $P \cup S$. Let G_{shell} denote the plane subgraph of G induced by the vertices of C and C_{shell} . (Figure 4(b)). Let C'_{shell} denote the outer cycle of G_{shell} . We denote the graph consisting of C , C'_{shell} and edges between them by G'_{shell} . Each petal or stamen of C , say w , that belongs to C_{shell} but not to C'_{shell} , belongs to a face of G'_{shell} . We denote this face by $\text{shell}(w)$. We categorize the faces of G_{shell} as follows. The faces that lie inside cycle C are called *faces of C* . The faces that are bounded only by C_{shell} and its chords, are called *faces of C_{shell}* . Notice that each face of C_{shell} is a triangle. Notice that a face of G_{shell} that is comprised by two consecutive edges adjacent to the same vertex of C (not belonging to C), is a triangle, and contains no vertex of $G \setminus G_{\text{shell}}$, since both facts would imply that the taken edges are not consecutive. Finally, faces bounded by a subpath of C and two edges adjacent to the same petal, are called *petal faces*. The plane subgraph of G inside a petal face is triangulated and does not have a chord connecting two vertices of its outer face, and therefore is triconnected. Thus we have the following

Observation 1. *Each vertex of $G \setminus G_{\text{shell}}$ either lies in a face of C , or in a face that is a triangle, or in a petal face, or outside C'_{shell} . Each subgraph of G inside a petal face is triconnected.*

Theorem 3. *Let G be a maximal plane graph and let C be a strictly internal simple outerchordless cycle of G , represented by a convex polygon Γ_C in the plane. Γ_C is extendable to a straight-line drawing of G if and only if each petal of C is realizable.*

Proof. The condition that each petal of C is realizable is clearly necessary. Next we show that it is also sufficient.

We first show how to draw the graph G'_{shell} . Afterward we complete it to a drawing of G_{shell} . Our target is to represent C'_{shell} as a one-sided polygon, so that Theorem 2 can be applied for the rest of G that lies outside C'_{shell} . We first decide which edge of C'_{shell} to “stretch”, i.e., which edge will serve as base edge of the one-sided polygon

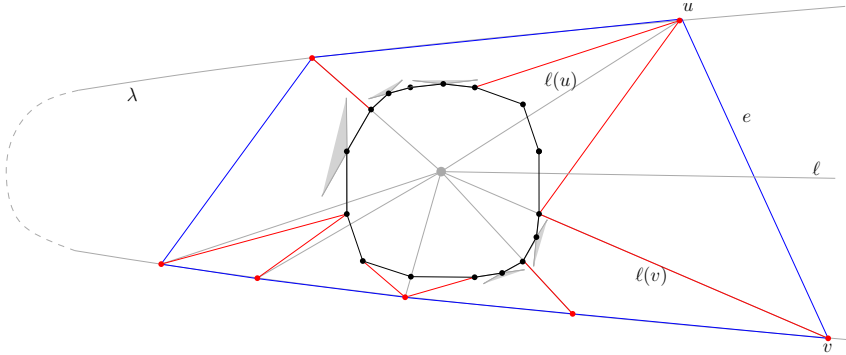


Fig. 6. Construction of Case 1. Corresponding G_{shell} is shown in Figure 4(b).

for representing C'_{shell} . In order to be able to apply Theorem 2, this one-sided polygon should be such that each petal of C'_{shell} is realizable. Thus we choose the base edge e of C'_{shell} as follows. If C'_{shell} contains an edge on the outer face of G , we choose e to be this edge. Otherwise, we choose an edge e , such that at least one of the end vertices of e is adjacent to an outermost petal of C'_{shell} in G . Such a choice of e ensures that each petal of C'_{shell} is realizable.

Claim 3. *Polygon Γ_C can be extended to a straight-line drawing of graph G'_{shell} , such that its outer face C'_{shell} is represented by a one-sided polygon with base edge e . Moreover, C'_{shell} contains in its interior all points of $\{\text{apex}(w) \mid w \in C'_{\text{shell}}\}$.*

Recall that P (resp. S) denotes the set of outermost petals (resp. stamens) of C in G . Let B denote the set of vertices of C , to which stamens $S \cap C'_{\text{shell}}$ are adjacent (refer to Figure 4(b)). By construction of the apex points, the set $\{\text{apex}(w) \mid w \in P \cap C'_{\text{shell}}\} \cup B$ is in convex position, and we denote by Π its convex hull. Polygon Π may be degenerate, and may contain only a single vertex or a single edge. We treat these cases separately to complete the construction of the drawing of the graph G'_{shell} . Next, we explain the construction in the non-degenerate case; the degenerate cases are covered in the full version of this paper [7].

Let p be a point inside Π . Let $\ell(w)$ denote a half-line from p through w , where w is a vertex of Π . If we order the constructed half-lines around p , any two consecutive lines have between them an angle less than π . If $w \in B$, we substitute $\ell(w)$ by the same number of slightly rotated lines as the number of stamens of C'_{shell} adjacent to w , without destroying the order (refer to Figure 5(b)). Thus, for each $w \in C'_{\text{shell}}$, a line $\ell(w)$ is defined. Notice that, for any $w \in P \cap C'_{\text{shell}}$, line $\ell(w)$ passes through $\text{apex}(w)$, and the infinite part of $\ell(w)$ lies in $\text{cone}(w)$. Thus, for any position of w on a point of $\ell(w) \cap \text{cone}(w)$, edges between C and w do not cross Γ_C . For a stamen $w \in S \cap C'_{\text{shell}}$, line $\ell(w)$ crosses $\text{cone}(w)$ very close to $\text{apex}(w)$, and its infinite part lies in $\text{cone}(w)$. Thus, similarly, for any position of w on a point of $\ell(w) \cap \text{cone}(w)$, edges between C and w do not cross Γ_C .

Recall that $e = (u, v)$ is the edge of C'_{shell} that we have decided to “stretch”. Recall also that $\ell(u)$ and $\ell(v)$ are consecutive in the sequence of half-lines we have

constructed. Let κ be a circle around Γ_C that contains in the interior the polygon Π and the set of points $\{\text{apex}(w) \mid w \in C_{\text{shell}}\}$. Let ℓ be a half-line bisecting the angle between $\ell(u)$ and $\ell(v)$ (refer to Figure 6). Let λ be a parabola with ℓ as axis of symmetry and the center of κ as focus. We position and parametrize λ such that it does not cross ℓ and κ .

With this placement of λ , each half-line $\ell(w)$, $w \in \Pi$, crosses λ , moreover, intersections with $\ell(u)$ and $\ell(v)$ are on different branches of λ and appear last on them as we walk on λ from its origin to infinity. Let Π' be the convex polygon comprised by the intersection points of lines $\{\ell(w) : w \in V(C'_{\text{shell}})\}$ with λ . We make λ large enough, so that the polygon Π' still contains the circle κ in the interior. As a results, for each $w \in C$, $\text{cone}(w) \cap \Pi'_{\text{in}} \neq \emptyset$, where Π'_{in} denotes the interior of Π' . This concludes the proof of the claim in the non-degenerate case.

Let Γ'_{shell} be the constructed drawing of G'_{shell} . Recall that each petal or stamen w of C , that does not belong to C'_{shell} , lies in a face of G'_{shell} , denoted by $\text{shell}(w)$. Let $\Gamma_{\text{shell}(w)}$ denote the polygon representing face $\text{shell}(w)$ in Γ'_{shell} . By construction, $\text{cone}(w) \cap \Gamma_{\text{shell}(w)} \neq \emptyset$. We next explain how to extend the drawing of G'_{shell} to the drawing of G_{shell} . For each edge (u, v) of C'_{shell} , we add a convex curve, lying close enough to this edge inside Γ'_{shell} . Let μ be the union of these curves for all edges of C'_{shell} . We notice that we can place them so close to C'_{shell} that all the points of $\{\text{apex}(w) \mid w \in C\}$ are still in the interior of μ . Thus μ is intersected by all the sets $\text{cone}(w)$, for each $w \in C$. We place each vertex w of $C_{\text{shell}} \setminus C'_{\text{shell}}$ on $\mu \cap \text{cone}(w)$ in the order they appear in C_{shell} . Since all edges induced by C_{shell} lie outside of C_{shell} , and both end points of such an edge are placed on a single convex curve, they can be drawn straight without intersecting each other, or other edges of G_{shell} . Thus, we have constructed a planar extension of Γ_C to a drawing of G_{shell} , call it Γ_{shell} .

Recall the definitions of faces of C , faces of C_{shell} and petal faces from the beginning of this section. The faces of C appear in Γ_{shell} as convex polygons. The faces of C_{shell} are triangles, and the petal faces of G_{shell} are star-shapes whose kernel is close to the corresponding petal. By Observation 1, each vertex of $G \setminus G_{\text{shell}}$ either lies in a face of C , or in a face that is a triangle, or in a petal face, or outside C'_{shell} . Moreover a subgraph of G inside a petal face is triconnected. Thus, by multiple applications of Theorem 1, we can extend the drawing of G_{shell} to a straight-line planar drawing of the subgraph of G inside C'_{shell} .

Finally, notice that in the constructed drawing of G_{shell} each petal of its outer cycle, i.e. C'_{shell} , is realizable. This is by the choice of edge e . Moreover, by construction of G_{shell} , C'_{shell} has no outer chords. In case C'_{shell} is not strictly internal, we apply Proposition 1, to construct a maximal plane graph G' , such that G is a plane subgraph of G' , C'_{shell} is a strictly internal outerchordless cycle of G' and each petal of C'_{shell} is realizable. Then, we apply Theorem 2, to complete the drawing of G' , lying outside C'_{shell} . Finally, we remove the edges of G' that do not belong to G . \square

We conclude with the following general statement, that follows from Proposition 1, Theorem 3 and one of the known algorithms that constructs drawing of a planar graph with a prescribed outer face (e.g. [4,10] or Theorem 1).

Corollary 1. *Let G be a plane graph and H be a biconnected plane subgraph of G . Let Γ_H be a straight-line convex drawing of Γ_H . Γ_H is extendable to a planar straight-line*

drawing of G if and only if the outer cycle of H is outerchordless and each petal of the outer cycle of H is realizable.

6 Conclusions

In this paper, we have studied the problem of extending a given convex drawing of a cycle of a plane graph G to a planar straight-line drawing of G . We characterized the cases when this is possible in terms of two simple necessary conditions, which we proved to also be sufficient. We note that it is easy to test whether the necessary conditions are satisfied in linear time. It is readily seen that our proof of existence of the extension is constructive and can be carried out in linear time. As an extension of our research it would be interesting to investigate whether more involved necessary conditions are sufficient for more general shape of a cycle, for instance a star-shaped polygon.

Acknowledgments. M.N. received financial support by the Concept for the Future of KIT. I.R. was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD). Part of this work was done within GRADR – EUROGIGA project no. 10-EuroGIGA-OP-003.

References

1. Angelini, P., Di Battista, G., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. In: 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 202–221. SIAM (2010)
2. Avis, D.: Generating rooted triangulations without repetitions. *Algorithmica* 16, 618–632 (1996)
3. Chambers, E.W., Eppstein, D., Goodrich, M.T., Löffler, M.: Drawing graphs in the plane with a prescribed outer face and polynomial area. *Journal of Graph Algorithms and Applications* 16(2), 243–259 (2012)
4. Duncan, C.A., Goodrich, M.T., Kobourov, S.G.: Planar drawings of higher-genus graphs. *Journal of Graph Algorithms and Applications* 15(1), 7–32 (2011)
5. Hong, S.-H., Nagamochi, H.: Convex drawings of graphs with non-convex boundary constraints. *Discrete Applied Mathematics* 156(12), 2368–2380 (2008)
6. Jelínek, V., Kratochvíl, J., Rutter, I.: A Kuratowski-type theorem for planarity of partially embedded graphs. *Computational Geometry Theory & Applications* 46(4), 466–492 (2013)
7. Mchedlidze, T., Nöllenburg, M., Rutter, I.: Drawing planar graphs with a prescribed inner face. CoRR, abs/1308.3370 (2013)
8. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 263–274. Springer, Heidelberg (1999)
9. Patrignani, M.: On extending a partial straight-line drawing. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 380–385. Springer, Heidelberg (2006)
10. Tutte, W.T.: How to draw a graph. *Proc. London Math. Soc.* 13(3), 743–768 (1963)

Metro-Line Crossing Minimization: Hardness, Approximations, and Tractable Cases

Martin Fink¹ and Sergey Pupyrev^{2,3,*}

¹ Lehrstuhl für Informatik I, Universität Würzburg, Germany

² Department of Computer Science, University of Arizona, USA

³ Institute of Mathematics and Computer Science, Ural Federal University, Russia

Abstract. Crossing minimization is one of the central problems in graph drawing. Recently, there has been an increased interest in the problem of minimizing crossings between paths in drawings of graphs. This is the *metro-line crossing minimization* problem (MLCM): Given an embedded graph and a set L of simple paths, called *lines*, order the lines on each edge so that the total number of crossings is minimized. So far, the complexity of MLCM has been an open problem. In contrast, the problem variant in which line ends must be placed in outermost position on their edges (MLCM-P) is known to be NP-hard.

Our main results answer two open questions: (i) We show that MLCM is NP-hard. (ii) We give an $O(\sqrt{\log |L|})$ -approximation algorithm for MLCM-P.

1 Introduction

In metro maps and transportation networks, some edges, that is, railway tracks or road segments, are used by several lines. Usually, lines that share an edge are drawn individually along the edge in distinct colors; see Fig. 1. Often, some lines must cross, and one normally wants to have as few crossings of metro lines as possible. In the *metro-line crossing minimization* problem (MLCM), the goal is to order different metro-lines along each edge of the underlying network so that the total number of crossings is minimized. Although the problem has been studied, many questions remain open.



Fig. 1. A part of the official metro map of Paris

* Research supported in part by NSF grant DEB 1053573.

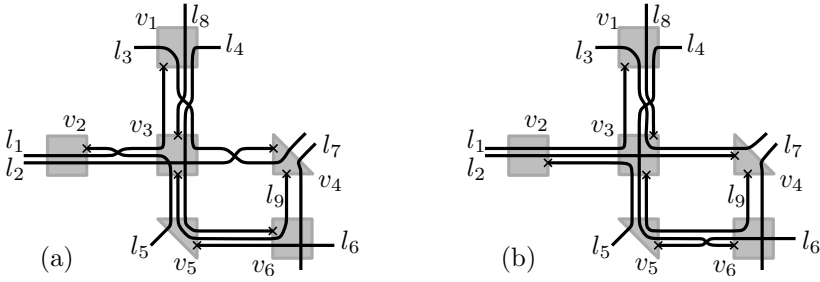


Fig. 2. Nine lines on a portion of an underlying network with 6 vertices and 8 edges. (a) $\pi_{v_3v_4} = (l_3, l_2)$ and $\pi_{v_3v_1} = (l_1, l_8, l_4, l_3)$. The lines l_3 and l_4 have an unavoidable edge crossing on $\{v_1, v_3\}$. In contrast, the crossing of l_2 and l_3 on $\{v_3, v_4\}$ is avoidable. In v_3 there is an unavoidable vertex crossing of the lines l_2 and l_8 . As the vertex crossing of l_2 and l_5 in v_3 is avoidable the solution is not feasible. (b) A feasible solution satisfying the periphery condition.

Apart from the visualization of metro maps, the problem has various applications including the visual representation of biochemical pathways. In very-large-scale integration (VLSI) design, there is the closely related problem of minimizing intersections between nets (physical wires) [8, 10]. Net patterns with fewer crossings have better electrical characteristics and require less area. In graph drawing, the number of edge crossings is one of the most important aesthetic criteria. In *edge bundling*, groups of edges are drawn close together—like metro lines—emphasizing the structure of the graph; minimizing crossings between parallel edges arises as a subproblem [14].

Problem Definitions. The input is a planarly embedded graph $G = (V, E)$ and a set L of simple paths in G . We call G the *underlying network*, the vertices *stations*, and the paths *lines*. The endpoints v_0, v_k of a line $(v_0, \dots, v_k) \in L$ are *terminals*, and the vertices v_1, \dots, v_{k-1} are *intermediate stations*. For each edge $e = (u, v) \in E$, let $L_e = L_{uv}$ be the set of lines passing through e .

Following previous work [2, 12], we use the *k-side* model; each station v is represented by a polygon with k sides, where k is the degree of v in G ; see Fig. 2. For $k \leq 2$ a rectangle is used. Each side of the polygon is called a *port* of v and corresponds to an incident edge $(v, u) \in E$. A line (v_0, \dots, v_k) is represented by a polyline starting at a port of v_0 (on the boundary of the polygon), passing through two ports of v_i for $1 \leq i < k$, and ending at a port of v_k . For each port of $u \in V$ corresponding to $(u, v) \in E$, we define the *line order* $\pi_{uv} = (l_1 \dots l_{|L_{uv}|})$ as an ordered sequence of the lines in L_{uv} , which specifies the clockwise order at which the lines L_{uv} are connected to the port of u with respect to the center of the polygon. Note that there are two different line orders π_{uv} and π_{vu} on any edge (u, v) of the network. A *solution*, or a *line layout*, specifies line orders π_{uv} and π_{vu} for each edge $(u, v) \in E$.

A line crossing is a crossing between polylines corresponding to a pair of lines. We distinguish two types of crossings; see Fig. 2(a). An *edge crossing* between lines l and l' occurs whenever $\pi_{uv} = (\dots l \dots l' \dots)$ and $\pi_{vu} = (\dots l \dots l' \dots)$ for some edge $(u, v) \in E$. We now consider the concatenated cyclic sequence π_u of the orders

$\pi_{uv_1}, \dots, \pi_{uv_k}$, where $(u, v_1), \dots, (u, v_k)$ are the edges incident to u in clockwise order. A *vertex crossing* between l and l' occurs in u if $\pi_u = (\dots l \dots l' \dots l \dots l' \dots)$. Intuitively, the lines change their relative order inside u . A crossing is called *unavoidable* if the lines cross in each line layout; otherwise it is *avoidable*. A crossing is unavoidable if neither l nor l' have a terminal on their common subpath and the lines split on both ends of this subpath in such a way that their relative order has to change; see Fig. 2. By checking all pairs of lines, we can determine all unavoidable crossings in $O(|L|^2|E|)$ time. Following previous work, we insist that (i) *avoidable vertex crossings are not allowed* in a solution, that is, these crossings are not hidden below a station symbol, and (ii) *unavoidable vertex crossings are not counted* since they occur in any solution.

A pair of lines may share several common subpaths, and the lines may cross multiple times on the subpaths. For simplicity of presentation, we assume that there is at most one common subpath of two lines. Our results do, however, also hold for the general case as every common subpath can be considered individually.

Problem variants. Several variants of the problem have been considered in the literature. The original metro-line crossing minimization problem is formulated as follows.

Problem 1 (MLCM). For a given instance (G, L) , find a line layout with the minimum number of crossings.

In practice, it is desirable to avoid gaps between adjacent lines; to this end, every line is drawn so that it starts and terminates at the topmost or bottommost end of a port; see Fig. 2(b). In fact, many manually created maps follow this *periphery condition* introduced by Bekos et al. [4]. Formally, we say that a line order π_{uv} at the port of u satisfies the periphery condition if $\pi_{uv} = (l_1 \dots l_p \dots l_q \dots l_{|L_{uv}|})$, where u is a terminal for the lines $l_1, \dots, l_p, l_q, \dots, l_{|L_{uv}|}$ and u is an intermediate station for the lines l_{p+1}, \dots, l_{q-1} . The problem is known as *MLCM with periphery condition*.

Problem 2 (MLCM-P). For a given instance (G, L) , find a line layout, subject to the periphery condition on every port, with the minimum number of crossings.

In the special case of MLCM-P with *side assignment* (MLCM-PA), the input additionally specifies for each line end on which side of its port it terminates; Nöllenburg [12] showed that MLCM-PA is computationally equivalent to the version of MLCM in which all lines terminate at vertices of degree one.

As MLCM and MLCM-P are NP-hard even for very simple networks, we introduce the additional constraint that no line is a subpath of another line. Indeed, this is often the case for bus and metro transportation networks; if, however, there is a line that is a subpath of a longer line then one can also visualize it as a part of the longer line. We call the problems with this new restriction *PROPER-MLCM* and *PROPER-MLCM-P*.

Previous Work. Benkert et al. [5] described a quadratic-time algorithm for MLCM when the underlying network consists of a single edge with attached leaves, leaving open the complexity status of MLCM.

Bekos et al. [4] studied MLCM-P and proved that the variant is NP-hard on paths. Motivated by the hardness, they introduced the variant MLCM-PA and studied the

Table 1. Overview of results for the metro-line crossing minimization problem

problem	graph class	result	reference
MLCM	caterpillar	NP-hard	Thm. 1
MLCM	single edge	$O(L ^2)$ -time algorithm	[5]
MLCM	general graph	crossing-free test	Thm. 2
MLCM-P	path	NP-hard	[2]
MLCM-P	general graph	ILP	[3]
MLCM-P	general graph	$O(\sqrt{\log L })$ -approximation	Thm. 5
MLCM-P	general graph	crossing-free test	Thm. 3
PROPER-MLCM-P	general graph with consistent lines	$O(L ^3)$ -time algorithm	Thm. 7
MLCM-PA	general graph	$O(V + E + V L)$ -time	[14]
MLCM-PA	general graph	crossing-free test	[12]

problem on simple networks. Later, polynomial-time algorithms for MLCM-PA were found with gradually improving running time by Asquith et al. [3], Argyriou et al. [2], and Nöllenburg [12], until Pupyrev et al. [14] presented a linear-time algorithm. Asquith et al. [3] formulated MLCM-P as an integer linear program that finds an optimal solution for the problem on general graphs. Note that in the worst case this approach requires exponential time. Fink and Pupyrev studied a variant of MLCM in which a crossing between two blocks of lines is counted as a single crossing [7]. Okamoto et al. [13] presented exact and fixed-parameter tractable algorithms for MLCM-P on paths.

In the circuit design community (VLSI), Groeneveld [8] considered the problem of adjusting the routing so as to minimize crossings between the pairs of nets, which is equivalent to MLCM-PA, and suggested an algorithm for general graphs. Marek-Sadowska et al. [10] considered a related problem of distributing the line crossings among edges of the underlying graph in order to simplify the net routing.

Our Results. Table 1 summarizes our contributions and previous results. We first prove that the unconstrained variant MLCM is NP-hard even on caterpillars (paths with attached leaves), thus, answering an open question of Benkert et al. [5] and Nöllenburg [11]. As crossing minimization is hard, it is natural to ask whether there exists a *crossing-free* solution. We show that there is a crossing-free solution if and only if there is no pair of lines forming an unavoidable crossing.

We then study MLCM-P. Argyriou et al. [2] and Nöllenburg [11] asked for an approximation algorithm. To this end, we develop a 2SAT model for the problem. Using the model, we get an $O(\sqrt{\log |L|})$ -approximation algorithm for MLCM-P. This is the first approximation algorithm in the context of metro-line crossing minimization. We also show how to find a crossing-free solution (if it exists) in polynomial time. Moreover, we prove that MLCM-P is fixed-parameter tractable with respect to the maximum number of allowed crossings by using the fixed-parameter tractability of 2SAT.

We then study the new variant PROPER-MLCM-P and show how to solve it on caterpillars, *left-to-right trees* (considered in [4,2]), and other instances described in Section 4. An optimal solution can be found by applying a maximum flow algorithm on a certain graph. This is the first polynomial-time exact algorithm for the variant in which avoidable crossings may be present in an optimal solution.

2 The MLCM Problem

We begin with MLCM, the most general formulation of the problem, and show that it is hard to decide whether there is a solution with at most $k > 0$ crossings, even if the underlying network is a caterpillar. In contrast, we give a polynomial-time algorithm for deciding whether there exists a crossing-free solution.

Theorem 1. *MLCM is NP-hard on caterpillars.*

Proof. We prove hardness by reduction from MLCM-P which is known to be NP-hard on paths [2]. Suppose we have an instance of MLCM-P consisting of a path $G = (V, E)$ and lines L on the path. We want to decide whether it is possible to order the lines with periphery condition and at most k crossings.

We create a new underlying network $G' = (V', E')$ by adding vertices and edges to G . We assume that G is embedded along a horizontal line and specify positions relative to this line. For each edge $e = (u, v) \in E$, we add vertices u_1, u_2, v_1 , and v_2 and edges $(u, u_1), (u, u_2), (v, v_1)$, and (v, v_2) such that v_1 and u_1 are above the path and v_2 and u_2 are below. Next, we add $\ell = |L|^2$ lines from u_1 to v_2 , and ℓ lines from u_2 to v_1 to $L' \supseteq L$; see Fig. 3. We call the added structure the *red cross* of e and the added lines *red lines*. We claim that there is a number K such that there is a solution of MLCM-P on (G, L) with at most k crossings if and only if there is solution of MLCM on (G', L') with at most $k + K$ crossings.

Let $e = (u, v) \in E$ be an edge of the path, and let $l \in L_e$. If l has its terminals on u and v , that is, completely lies on e , it never has to cross in G or G' ; hence, we assume such lines do not exist. Assume l has none of its terminals on u or v . It has to cross all 2ℓ lines of the red cross of e . Finally, suppose l has just one terminal at a vertex of e , say on u . If the line end of l at u is above the edge (u, u_1) in the order π_{uv} , then it has to cross all red lines from u_2 to v_1 but can avoid the red lines from u_1 to v_2 , that is, ℓ crossings with red lines are necessary. Symmetrically, if the line end is below (u, u_2) then only the ℓ crossings with the red lines from u_1 to v_2 are necessary. If the terminal is between the edges (u, u_1) and (u, u_2) then all 2ℓ red edges must be crossed. There are, of course, always ℓ^2 unavoidable internal crossings of the red cross of e .

Let $\ell_e = \ell_e^t + \ell_e^m$ be the number of lines on e , where ℓ_e^t and ℓ_e^m are the numbers of lines on e that do or do not have a terminal at u or v , respectively. In any solution there are at least $\ell_e^t \cdot \ell + 2 \cdot \ell_e^m \cdot \ell + \ell^2$ crossings on e in which at least one red line is involved. It is easy to see that placing a terminal between red lines leaving towards a leaf never brings an advantage. On the other hand, if just a single line has an avoidable crossing with a block of red lines, the number of crossings increases by $\ell = |L|^2$, which is more than the number of crossings in any solution for (G, L) without double crossings. Hence, any optimal solution of the lines in G' has no avoidable crossings with red blocks and,

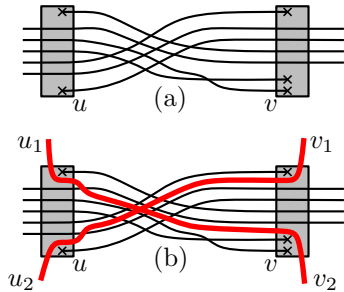


Fig. 3. (a) MLCM-P-solution on (u, v) . (b) Insertion of a red cross

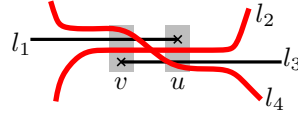
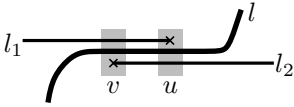


Fig. 4. A separator l of lines l_1 and l_2 **Fig. 5.** Unavoidable crossing of 2 separators of l_1 and l_3

therefore, satisfies the periphery condition; thus, after deleting the added edges and red lines, we get a feasible solution for MLCM-P on G .

Let $K := |E| \cdot \ell^2 + \sum_{e \in E} (\ell_e^t + 2\ell_e^m) \cdot \ell$ be the minimum number of crossings with red lines involved on G' . Suppose we have an MLCM-solution on G' with at most $K+k$ crossings. Then, after deleting the red lines, we get a feasible solution for MLCM-P on G with at most k crossings. On the other hand, if we have an MLCM-P-solution on G with k crossings, then we can insert the red lines with just K new crossings: Suppose we want to insert the block of red lines from u_1 to v_2 on an edge $e = (u, v) \in E$. We start by putting them immediately below the lines with a terminal on the top of u . Then we cross all lines below until we see the first line that ends on the bottom of v and, hence, must not be crossed by this red block. We go to the right and just keep always directly above the block of lines that end at the bottom side of v ; see Fig. 3. Finally, we reach v and have not created any avoidable crossing. Once we have inserted all blocks of red lines, we get a solution for the lines on G with exactly $K + k$ crossings. \square

Crossing-Free Instances. Given an instance of MLCM, we want to check whether there exists a solution without any crossings. If there exists such a crossing-free solution then there cannot be a pair of lines with an unavoidable crossing. We show that this condition is already sufficient. We sketch the proofs; see full version for the complete proof [6].

We assume that no line is a subpath of another line as a subpath can be reinserted parallel to the longer line in a crossing-free solution. Consider a pair of lines l_1, l_2 whose common subpath P starts in u and ends in v . If u (similarly, v) is not a terminal for both l_1 and l_2 then there is a unique relative order of the lines along P in any crossing-free solution. Hence, we assume u is a terminal for l_1 , v is a terminal for l_2 , and we call such a pair *overlapping*. Suppose there is a *separator* for l_1 and l_2 , that is, a line l on the subpath of l_1 and l_2 that has to be below l_1 and above l_2 (or the other way round) as shown in Fig. 4. Then, l_1 has to be above l_2 in a crossing-free solution. The only remaining case is a pair of lines l_1, l_2 without a separator. Suppose l_1, l_2 is chosen such that the number of edges of the common subpath is minimum. If there exists a crossing-free solution then there is also a crossing-free solution in which l_1 and l_2 are immediate neighbors in the orders on their common subpath; see full version [6].

Theorem 2. *Any instance of MLCM without unavoidable crossings has a crossing-free solution.*

Proof (sketch). We can merge a pair of overlapping lines without a separator into a new line. This merging step does not introduce an unavoidable crossing. We iteratively perform merging steps until any overlapping pair has a separator. There might be multiple separators for a pair, but all of them separate the pair in the same relative order;

otherwise, there would be a pair of separators with an unavoidable crossing; see Fig. 5. After the merging steps, we get a relative order for every pair of lines sharing an edge. One can show that the relative orders are acyclic. We build a crossing-free solution by putting all lines in the (only) right order on the edge. As the relative order of any pair of lines is the same on any edge, there cannot be a crossing. \square

The proof yields an $O(|L|^2|E|)$ -time algorithm for finding crossing-free solutions.

3 The MLCM-P Problem

Let $(G = (V, E), L)$ be an instance of MLCM-P. Our goal is to decide for each line end on which side of its terminal port it should lie. We arbitrarily choose one side of each port and call it “top”, the opposite side is called “bottom”. For each line l starting at vertex u and ending at vertex v , we create binary variables l_u and l_v , which are true if and only if l terminates at the top side of the respective port. We formulate the problem of finding a truth assignment that minimizes the number of crossings as a 2SAT instance. Note that Asquith et al. [3] already used 2SAT clauses as a tool for developing their ILP for MLCM, where the variables represent above/below relations between line ends. In contrast, in our model a variable directly represents the position of a line on the top or bottom side of a port. We first prove a simple property of lines.

Lemma 1. *Let l, l' be a pair of lines sharing a terminal. We can transform any solution in which l and l' cross to a solution with fewer crossings in which the lines do not cross.*

Proof. Assume l and l' cross in a solution. We switch the positions of line ends at the common terminal v between l and l' and reroute the two lines between the crossing’s position and v . By reusing the route of l for l' and vice versa, the number of crossings does not increase. On the other hand, the crossing between l and l' is eliminated. \square

Let l, l' be two lines whose common subpath P starts at vertex u and ends at v . Terminals of l and l' that lie on P can only be at u or v . If neither l nor l' has a terminal on P , then a crossing of the lines does not depend on the positions of the terminals; hence, we assume that there is a terminal at u or v . A possible crossing between l and l' is modeled by a 2SAT formula, the *crossing formula*, consisting of at most two clauses. This formula evaluates to true if and only if l and l' do not cross. For simplicity, we assume that the top sides of the terminal ports of u and v are located on the same side of P ; see Fig. 6. If it is not the case, a variable l_u should be substituted with its inverse $\neg l_u$ in the formula. Note that generating all crossing formulas needs $O(|E||L|^2)$ time. We consider several cases; see also the illustrations in the full version [6].

- (f1) Suppose u and v are terminals for l and intermediate stations for l' , that is, l is a subpath of l' . Then, l does not cross l' if and only if both terminals of l lie on the same side of P . This is expressed by the crossing formula $(l_u \wedge l_v) \vee (\neg l_u \wedge \neg l_v) \equiv (\neg l_u \vee l_v) \wedge (l_u \vee \neg l_v)$, which may occur multiple times, caused by a different l' .

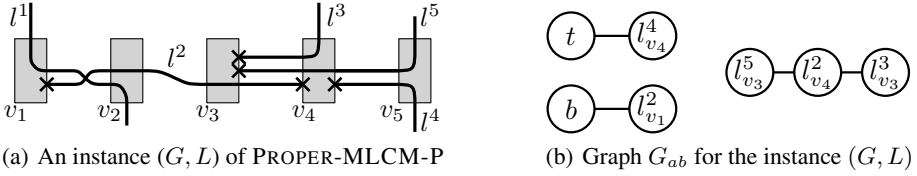


Fig. 6. A small instance of MLCM-P. The generated 2SAT formulas are: $(l^2_{v_1})$ for the crossing of l^1 and l^2 ; $(\neg l^4_{v_4})$ for the crossing of l^5 and l^4 ; $(l^2_{v_4} \vee l^3_{v_3}) \wedge (\neg l^2_{v_4} \vee \neg l^3_{v_3})$ for the crossing of l^2 and l^3 ; $(l^2_{v_4} \vee l^5_{v_3}) \wedge (\neg l^2_{v_4} \vee \neg l^5_{v_3})$ for the crossing of l^2 and l^5 .

- (f2) Suppose u is a terminal for l and intermediate for l' , and v is a terminal for l' and intermediate for l . Then there is no crossing if and only if both terminals lie on opposite sides of P . This is described by the formula $(l_u \wedge \neg l'_v) \vee (\neg l_u \wedge l'_v) \equiv (l_u \vee l'_v) \wedge (\neg l_u \vee \neg l'_v)$.
- (f3) Suppose both l and l' terminate at the same vertex u or v . By Lemma 1, a solution of MLCM-P with a crossing of l and l' can be transformed into a solution in which l and l' do not cross. Hence, we do not introduce formulas in this case.
- (f4) In the remaining case, there is only one terminal of l and l' on P . Without loss of generality, let l terminate at u . A crossing is triggered by a single variable. Depending on the fixed terminals or leaving edges at v and u , we get the single clause (l_u) or $(\neg l_u)$. The same clause can occur multiple times, caused by different lines l' .

Crossing-free solutions. Note that the 2SAT formulation of the problem yields an algorithm for deciding whether there exists a crossing-free solution of an MLCM-P instance. First, we check for unavoidable crossings by analyzing every pair of lines individually. Second, the 2SAT model is satisfiable if and only if there is a solution of the MLCM-P instance without unavoidable crossing. Since 2SAT can be solved in linear time and there are at most $|L|^2$ crossing formulas, we conclude as follows.

Theorem 3. *Deciding whether there exists a crossing-free solution for MLCM-P can be accomplished in $O(|E||L|^2)$ time.*

For MLCM the existence of a crossing-free solution is equivalent to the absence of unavoidable crossings. In contrast, there is no such simple criterion for MLCM-P.

Fixed-parameter tractability. We can use the 2SAT model for obtaining a fixed-parameter tractable algorithm on the number k of allowed crossings. We must show that we can check in $f(k) \cdot \text{poly}(\mathcal{I})$ time whether there is a solution with at most k avoidable crossings, where f must be a computable function and \mathcal{I} is the input size.

First, note that minimizing the number of crossings is the same as maximizing the number of satisfied clauses in the corresponding 2SAT instance. Maximizing the number of satisfied clauses, or solving the MAX-2SAT problem, is NP-hard.

However, the problem of deciding whether it is possible to remove a given number k of m 2SAT clauses so that the formula becomes satisfiable is fixed-parameter tractable with respect to the parameter k [15]. This yields the following theorem.

Theorem 4. *MLCM-P is fixed-parameter tractable with respect to the maximum allowed number of avoidable crossings.*

Proof. We show that the SAT formula can be made satisfiable by removing at most k clauses if and only if there is a solution with at most k crossings.

First, suppose it is possible to remove at most k clauses from the 2SAT model so that there is a truth assignment satisfying all remaining clauses. Fix such a truth assignment, and consider the corresponding assignment of sides to the terminals. Any crossing leads to an unsatisfied clause in the SAT formula, and no two crossings share an unsatisfied clause. Hence, we have a side assignment that causes at most k crossings.

Now, we assume that there is an assignment of sides for all terminals that causes at most k crossings. We know that in the corresponding truth assignment for all pairs of clauses of types (f1)–(f2) of the SAT model at most one is unsatisfied. Hence, there are at most k unsatisfied clauses since any crossing just leads to a single unsatisfied clause. The removal of these clauses creates a new, satisfiable formula. \square

Using the $O(15^k km^3)$ -time algorithm for 2SAT [15] our algorithm has a running time of $O(15^k \cdot k \cdot |L|^6 + |L|^2|E|)$.

Approximating MLCM-P. The proof of Theorem 4 yields that the number of crossings in a crossing-minimal solution of MLCM-P equals the minimum number of clauses that we need to remove from the 2SAT formula in order to make it satisfiable. Furthermore, a set of k clauses, whose removal makes the 2SAT formula satisfiable, corresponds to an MLCM-P solution with at most k crossings. Hence, an approximation algorithm for the problem of making a 2SAT formula satisfiable by removing the minimum number of clauses (also called MIN 2CNF DELETION) yields an approximation for MLCM-P of the same quality. As there is an $O(\sqrt{\log m})$ -approximation algorithm for MIN 2CNF DELETION [1], we have the following result.

Theorem 5. *There is an $O(\sqrt{\log |L|})$ -approximation algorithm for MLCM-P.*

4 The PROPER-MLCM-P Problem

In this section we consider the PROPER-MLCM-P problem, where no line in L is a subpath of another line. First we focus on graphs whose underlying network is a caterpillar. There, the top and bottom sides of ports are given naturally; see Fig. 6.

Based on the 2SAT model described in the previous section, we construct a graph G_{ab} , which has a vertex l_u for each variable of the model and two additional vertices b and t . Since no line is a subpath of another line, our 2SAT model has only the two types of crossing formulas (f2) and (f4); compare Section 3. For case (f2), we create an edge (l_u, l'_v) . The edge models a possible crossing between lines l and l' ; that is, the lines cross if and only if l terminates on top (bottom) of u and l' terminates on top (bottom) of v . For a crossing formula of type (l_u) (case (f4)), we add an edge (b, l_u) to G_{ab} ; similarly, we add an edge (t, l_u) for a formula $(\neg l_u)$; see Fig. 6(b) for an example.

Any truth assignment to the variables is equivalent to a b - t cut in G_{ab} , that is, a cut separating b and t . Indeed, any edge in the graph models the fact that two lines should

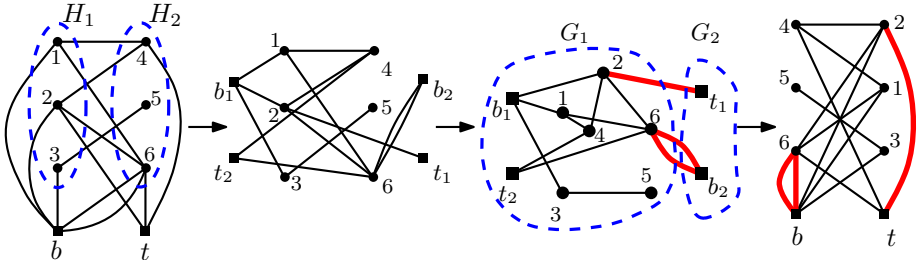


Fig. 7. Solving MIN-UNCUT on an almost bipartite graph. The maximum flow (minimum cut) with value 3 results in vertex partitions $V_b^1 = \{b_1, 4, 5, 6\}$, $V_t^1 = \{t_2, 1, 2, 3\}$, $V_b^2 = \{b_2\}$, and $V_t^2 = \{t_1\}$. The optimal partition $S_b = \{b, 4, 5, 6\}$, $S_t = \{t, 1, 2, 3\}$ induces 3 uncut edges $(b, 6), (b, 6), (t, 2)$.

not be assigned to the same side as they would cause a crossing otherwise. Hence, any line crossing corresponds to an *uncut* edge. Therefore, for minimizing the number of crossings, we need to solve the known MIN-UNCUT problem, which asks for a partitioning of the vertices of a graph into sets S_t, S_b so that the number of uncut edges $((v, u)$ with $v, u \in S_t$ or $v, u \in S_b$) is minimized. Although MIN-UNCUT is NP-hard, it turns out that the graph G_{ab} has a special structure, which we call *almost bipartite*.

Definition 1. A graph $G = (V, E)$ is called *almost bipartite* if it is a union of a bipartite graph $H = (V_H, E_H)$ and two additional vertices b, t whose edges may be incident to vertices of both partitions of H , that is, $V = V_H \cup \{b\} \cup \{t\}$ and $E = E_H \cup E'$, where $E' \subseteq \{(b, v) \mid v \in V\} \cup \{(t, v) \mid v \in V\}$.

The bipartition is given by the fact that “left” (similarly, “right”) terminals of two lines can never be connected by an edge in G_{ab} . We show that MIN-UNCUT can be solved optimally for almost bipartite graphs.

Theorem 6. MIN-UNCUT can be solved in $O(|V|^3)$ on almost bipartite graphs.

Proof. Almost bipartite graphs are a subclass of *weakly bipartite graphs*. It is known that MIN-UNCUT can be solved in polynomial time on weakly bipartite graphs using the ellipsoid method [9]. We present a simple and efficient combinatorial algorithm.

The special vertices b and t have to belong to different partitions of G_{ab} . We create a new graph G' from G_{ab} . We split vertex b into b_1, b_2 and t into t_1, t_2 such that b_1 and t_1 are connected to the vertices of the first partition H_1 of H , and b_2 and t_2 are connected to the second partition H_2 . Formally, for each edge $(b, v) \in E, v \in H_1$, we create an edge (b_1, v) ; for each edge $(b, v) \in E, v \in H_2$, we create an edge (v, b_2) . Similarly, edges (v, t_1) are created for all $(t, v) \in E, v \in H_1$, and edges (t_2, v) are created for all $(t, v) \in E, v \in H_2$. The construction is illustrated in Fig. 7.

Now, for each edge (u, v) of G' we assign capacity 1, and compute a maximum flow between the pair of sources b_1, t_2 to the pair of sinks b_2, t_1 . This can be done in $O(|V|^3)$ time using a maximum flow algorithm with a supersource (connected to b_1 and t_2) and a supersink (connected to b_2 and t_1). Indeed, there is an integral maximum flow in G' .

A maximum flow corresponds to a maximum set of edge-disjoint paths starting at b_1 or t_2 and ending at b_2 or t_1 . Such a path corresponds to one of the following structures in the original graph G : (i) an odd cycle containing vertex b (a cycle with an odd number of edges); (ii) an odd cycle containing vertex t ; (iii) an even path between b and t .

Note that if a graph has an odd cycle then at least one of the edges of the cycle belongs to the same partition in any solution of MIN-UNCUT. The same holds for an even path connecting b and t in G since b and t have to belong to different partitions. Since the maximum flow corresponds to the edge-disjoint odd cycles and even paths in G , the value of the flow is a lower bound for a solution of MIN-UNCUT.

Let us prove that the value of the maximum flow in G' is also an upper bound. By Menger's theorem, this value is the cardinality of a minimum edge cut separating sources and sinks. Let E^* be the minimum edge cut and let G_1 and G_2 be the correspondent disconnected subgraphs of G' ; see Fig. 7. Note that G_1 is bipartite since $H \cap G_1$ is bipartite; vertex b_1 is only connected to vertices of H_1 and vertex t_2 is only connected to vertices of H_2 . Therefore, there is a 2-partition of vertices of G_1 such that b_1 and t_2 belong to different partitions; let us denote the partitions V_b^1 and V_t^1 . Similarly, there is a 2-partition of G_2 into V_b^2 and V_t^2 with $b_2 \in V_b^2$ and $t_1 \in V_t^2$. We combine these partitions so that $S_b = \{b\} \cup (V_b^1 \cup V_b^2) \setminus \{b_1, b_2\}$ and $S_t = \{t\} \cup (V_t^1 \cup V_t^2) \setminus \{t_1, t_2\}$, which yield the required partition of vertices of G for MIN-UNCUT. The set of uncut edges is E^* , which completes the proof of the theorem. \square

As a direct corollary, we get a $O(|L|^3)$ -time algorithm for PROPER-MLCM-P on caterpillars. It can be applied for some other underlying networks. Let $(G = (V, E), L)$ be an instance of PROPER-MLCM-P. The lines L have *consistent* directions on G if the lines can be directed so that for each edge $e \in E$ all lines L_e have the same direction. If the underlying graph is a path then we can consistently direct the lines from left to right. Similarly, consistent line directions exist for "left-to-right" [4,2] and "upward" [7] trees, that is, trees for which there is an embedding with all lines being monotone in some direction. It is easy to test whether there are consistent line directions by giving an arbitrary direction to some first line, and then applying the same direction on all lines sharing edges with the first line until all lines have directions or an inconsistency is found. Hence, we get the following result; see full version for the proof [6].

Theorem 7. PROPER-MLCM-P can be solved in $O(|L|^3)$ time for instances (G, L) admitting consistent line directions.

5 Conclusion and Open Problems

We proved that MLCM is NP-hard and presented an $O(\sqrt{\log |L|})$ -approximation algorithm for MLCM-P, as well as an exact $O(|L|^3)$ -time algorithm for PROPER-MLCM-P on instances with consistent line directions. We also suggested polynomial-time algorithms for crossing-free solutions for MLCM and MLCM-P. From a theoretical point of view, there are still interesting open problems: 1. Is there an approximation algorithm for MLCM? 2. Is there a constant-factor approximation algorithm for MLCM-P? 3. What is the complexity status of PROPER-MLCM/PROPER-MLCM-P in general?

On the practical side, the visualization of the computed line crossings is a possible future direction. So far, the focus has been on the number of crossings, although two line orders with the same crossing number may look quite differently [7]. For example, a metro line is easy to follow if it has few bends. Hence, an interesting question is how to visualize metro lines using the minimum total number of bends.

Acknowledgments. We thank Martin Nöllenburg, Jan-Henrik Haunert, Joachim Spoerhase, Lukas Barth, Stephen Kobourov, and Sankar Veeramoni for discussions about problem variants. We are especially grateful to Alexander Wolff for help with the paper.

References

1. Agarwal, A., Charikar, M., Makarychev, K., Makarychev, Y.: $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In: STOC 2005, pp. 573–581. ACM, New York (2005)
2. Argyriou, E.N., Bekos, M.A., Kaufmann, M., Symvonis, A.: On metro-line crossing minimization. *Journal of Graph Algorithms and Applications* 14(1), 75–96 (2010)
3. Asquith, M., Gudmundsson, J., Merrick, D.: An ILP for the metro-line crossing problem. In: Harland, J., Manyem, P. (eds.) CATS 2008. CRPIT, vol. 77, pp. 49–56. Australian Computer Society (2008)
4. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Line crossing minimization on metro maps. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 231–242. Springer, Heidelberg (2008)
5. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
6. Fink, M., Pupyrev, S.: Metro-line crossing minimization: Hardness, approximations, and tractable cases. ArXiv e-print abs/1306.2079 (2013), <http://arxiv.org/abs/1306.2079>
7. Fink, M., Pupyrev, S.: Ordering metro lines by block crossings. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 397–408. Springer, Heidelberg (2013)
8. Groeneveld, P.: Wire ordering for detailed routing. *IEEE Des. Test* 6(6), 6–17 (1989)
9. Grötschel, M., Pulleyblank, W.: Weakly bipartite graphs and the Max-Cut problem. *Operations Research Letters* 1(1), 23–27 (1981)
10. Marek-Sadowska, M., Sarrafzadeh, M.: The crossing distribution problem. *IEEE Transactions on CAD of Integrated Circuits and Systems* 14(4), 423–433 (1995)
11. Nöllenburg, M.: *Network Visualization: Algorithms, Applications, and Complexity*. Ph.D. thesis, Fakultät für Informatik, Universität Karlsruhe (TH) (2009)
12. Nöllenburg, M.: An improved algorithm for the metro-line crossing minimization problem. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 381–392. Springer, Heidelberg (2010)
13. Okamoto, Y., Tatsu, Y., Uno, Y.: Exact and fixed-parameter algorithms for metro-line crossing minimization problems. ArXiv e-print abs/1306.3538 (2013)
14. Pupyrev, S., Nachmanson, L., Bereg, S., Holroyd, A.E.: Edge routing with ordered bundles. In: van Kreveld, M.J., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 136–147. Springer, Heidelberg (2012)
15. Razgon, I., O’Sullivan, B.: Almost 2-SAT is fixed-parameter tractable. *Journal of Computer and System Sciences* 75(8), 435–450 (2009)

Fixed Parameter Tractability of Crossing Minimization of Almost-Trees

Michael J. Bannister, David Eppstein, and Joseph A. Simons

Department of Computer Science, University of California, Irvine

Abstract. We investigate exact crossing minimization for graphs that differ from trees by a small number of additional edges, for several variants of the crossing minimization problem. In particular, we provide fixed parameter tractable algorithms for the 1-page book crossing number, the 2-page book crossing number, and the minimum number of crossed edges in 1-page and 2-page book drawings.

1 Introduction

Graphs that differ from a tree by the inclusion of a small number of edges arise in many applications; such graphs are called *almost-trees*. Almost-trees can be found in the areas of biology, medicine, operations research, sociology, genealogy, distributed systems, and telecommunications, and in each of these applications it is important to find effective visualizations¹. One of the most important criteria for the aesthetics and readability of a graph drawing is its number of its crossings. Although crossing minimization problems tend to be NP-complete, we may hope that the graphs arising in applications are not hard instances for these problems, allowing us to find optimal drawings for them efficiently. In this paper we prove that almost-trees are indeed not hard instances by designing algorithms for crossing minimization of almost-trees that are fixed-parameter tractable when parameterized by the number of extra non-tree edges in these graphs.

Many different variants of the crossing number have been studied, depending on what types of drawing are allowed and what we count as a crossing [1]. The most frequently studied is the topological crossing number, $cr(G)$, which counts the number of crossings in a unrestricted placement of vertices and edges in the plane. In this paper we consider also the 1-page and 2-page crossing numbers, denoted $cr_1(G)$ and $cr_2(G)$ respectively. The 1-page crossing number counts the minimal number of crossings in a drawing where all the vertices of G are placed on a straight line, and all edges must be placed to one side of the line. The 2-page crossing number is defined similarly: all vertices of G are placed on a straight line and edges may be assigned to either side of the line, but are not allowed to cross the line. In both 1-page and 2-page drawings it is not uncommon to place the vertices on a circle instead of a straight line; this does not change the crossing structure of the drawing. In addition to the number of crossings we consider the number of crossed edges for these drawing styles, denoted $cre_1(G)$ and $cre_2(G)$.

¹ We provide more details about these applications in Section 2.

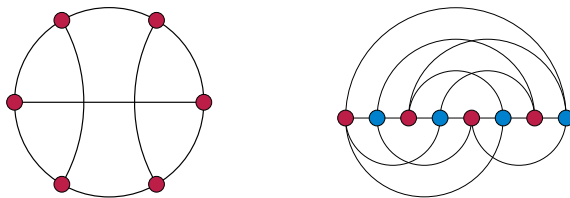


Fig. 1. Left: 1-page circular embedding with two crossings and three crossed edges. Right: 2-page linear embedding of $K_{4,4}$ with four crossings and eight crossed edges.

Following Gurevich, Stockmeyer and Vishkin [2] we define a k -almost-tree to be a graph such that every biconnected component of the graph has cyclomatic number at most k , where the *cyclomatic number* is the difference between the number of edges in a graph and in one of its maximal spanning forests. The k -almost-tree parameter has been used in past fixed-parameter algorithms [3–9], and will play the same role in our algorithms for crossing minimization.

Grohe and later Kawarabayashi and Reed showed the topological crossing number to be fixed parameter tractable for its natural parameter [10, 11]; the same is true for odd crossing number [12]. Because the topological crossing number is at most quadratic in the k -almost-tree parameter, $cr(G)$ is also fixed parameter tractable for k -almost-trees. However, to our knowledge no fixed parameter tractable algorithms were known for computing 1-page or 2-page crossing numbers. Indeed, for the 2-page problem, determining whether a graph can be drawn with zero crossings is already NP-complete [13], so to achieve fixed parameter tractability we must use some other parameter such as the k -almost-tree parameter rather than using the crossing number itself as a parameter.

Our main results are that $cr_1(G)$, $cr_2(G)$, $cre_1(G)$, and $cre_2(G)$ are all fixed-parameter tractable for almost-trees. As with previous work on parameterized algorithms for crossing numbers [10–12], our algorithms have a high dependence on their parameters. Making our algorithms more practical by reducing this dependence remains an open problem.

2 Application Domains

Examples of k -almost-trees can be found in biological gene expression networks, where vertices represent genes and edges represent correlations between pairs of genes. The k -almost-tree structure of such graphs has been exploited in parameterized algorithms for finding sequences of valid labelings of genes as active or inactive [5]. The parameter k has also been used in algorithms for continuous facility location where weighted edges represent a road network on which to efficiently place facilities serving clients [2]. Intraprogram communication networks whose vertices represent modules of a distributed system and whose edges represent communicating pairs of modules also have an almost-tree structure that has been exploited for parameterized algorithms [3].

A typical example of almost-trees arises when studying the spread of sexually transmitted infections, where sexual networks are constructed by voluntary survey. In these graphs vertices represent people who have received treatment, and edges represent their reported sexual partners. Analysis of these networks allows researchers to identify the growth and decline phases of an outbreak, and the general spread of the disease [14–16].

Another type of social network represents the business dealings of individuals and business entities. Examples of these networks can be found in the art of Mark Lombardi, an artist famous for his drawings of networks connecting the key players of conspiracy theories [17]. Many of Lombardi’s networks show an almost-tree structure; the Lippo Group Shipping network listed below is one.

The directed acyclic graphs originating from genealogical data where edges represent parental relationships on the vertices are another example of k -almost-trees, when viewed as undirected graphs, since in modern societies marriage between close relatives is rare. Similar types of graphs also come from animal pedigrees, academic family trees, and organizational lineages [18].

Utility networks such as telecommunication networks and power grids also form an almost-tree structure, where additional edges beyond those of a spanning tree provide load balancing and redundancy. Since such links are expensive they are placed in the network sparingly.

In order to visually distinguish the tree-like parts of these graphs from the parts with nontrivial connectivity, we may use a *sunburst* style (Figure 2) in which the 2-*core* of the graph (the part of a graph which is left after repeatedly removing all degree one vertices [19]) is drawn with a one-page circular layout and the rest of the graph extends outwards using a radial layout on concentric circles. In this style, crossings occur only within the inner one-page layout, motivating our interest in crossing minimization for one-page drawings of almost-trees.

We collect statistics for several real world graphs in Table 1. The table shows vertex and edge counts (n and m), the cyclomatic number $a = m - n + 1$, the

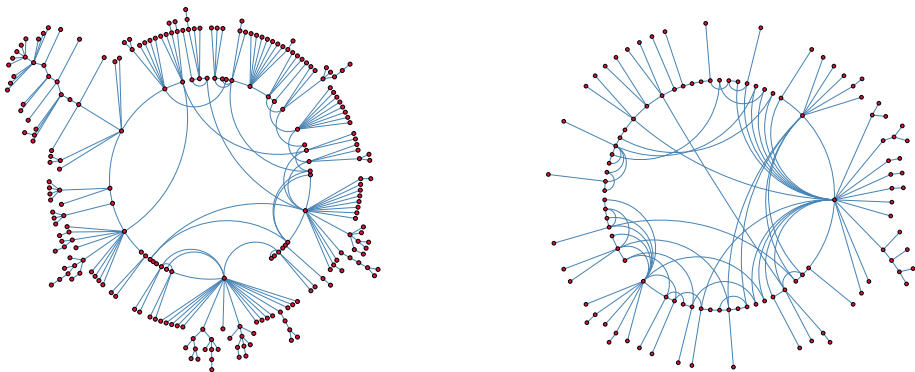


Fig. 2. Two sunburst drawings. Left: An HIV infection graph. Right: Lombardi’s World Finance Miami graph.

k -almost-tree parameter k , and the vertex and edge counts for the 2-core (n_2 and m_2). For most of these graphs the parameters a and k are low.

Table 1. Statistics for real-world almost-trees

Name	n	m	a	k	n_2	m_2
Gonorrhoea sexual network 1 [15]	38	39	2	2	9	10
Gonorrhoea sexual network 2 [15]	84	90	7	4	22	28
Lippo Group Shipping [17]	96	112	17	16	45	61
Global International Airways and Indian Springs State Bank [17]	82	99	18	15	33	50
Gondola Genealogy [20]	242	255	14	14	50	63
HIV [14]	243	257	15	12	39	53
Power Grid [21]	4941	6594	1654	1516	3353	5006

3 The Kernel

Our fixed-parameter algorithms use the *kernelization method*. In this method we find a polynomial time transformation from an arbitrary input instance to a *kernel*, an instance whose size is bounded by a fixed function $f(k)$ of the parameter value, and then apply a non-polynomial algorithm to the kernel. In this section we outline the general method for kernelization that we use in our fixed parameter algorithms, based on a similar kernelization by Bannister, Cabello and Eppstein [22] for a different problem, 1-planarity testing.

We first describe our kernelization for cyclomatic number, which starts by reducing the graph to its 2-core. The 2-core of a graph can be found in linear time by initializing a queue of degree one vertices, repeatedly finding and removing vertices from the queue and the graph, and updating the degree and queue membership of the neighbor of each removed vertex. The 2-core consists of vertices of degree at least three connected to each other by paths of degree two vertices. The following lemma bounds the numbers of high degree vertices and maximal paths of degree two vertices.

Lemma 1. *If G is a graph with cyclomatic number k and minimum degree three then G has at most $2k - 2$ vertices and at most $3k - 3$ edges. Furthermore, this bound is tight. As a consequence, the 2-core of a graph with cyclomatic number k has at most $2k - 2$ vertices of degree at least three, and at most $3k - 3$ maximal paths of degree two vertices.*

Proof. Double counting yields $2(n - 1 + k) \geq 3n$, simplifying to $n \leq 2k - 2$. A spanning tree of G has at most $2k - 3$ edges, and there are k edges outside the tree, from which the bound on edges follows. For a graph realizing the upper bound consider any cubic graph with $2k - 2$ vertices. □

The final step in this kernelization is to reduce the length of the maximal degree two paths. Depending on the specific problem, we will determine a maximal allowed path length $\ell(k)$, and if any paths exceed this length we will shorten them to length exactly $\ell(k)$. After this step the kernel will have $O(k\ell(k))$ edges and vertices, bounded by a function of k .

To change the parameter of our algorithms from the cyclomatic number to the k -almost-tree parameter, we first decompose the graph into its biconnected components. These components have a tree structure and in most drawing styles they can be embedded separately without introducing crossings. We then kernelize and optimally embed each biconnected component individually.

4 1-page Crossing Minimization

Minimizing crossings in 1-page drawings is important for several drawing styles, but is NP-hard [23], leading Baur and Brandes to develop fast practical heuristics for reducing but not optimizing the number of crossings [24]. As we now show, crossing minimization and crossed edge minimization in 1-page drawings of k -almost-trees is fixed-parameter tractable in the parameter k . We use the kernelization of Section 3, keeping one vertex per maximal degree two path.

Lemma 2. *Let G have cyclomatic number k and let K be the kernel constructed from G with $\ell(k) = 2$. Then*

1. K has at most $5k$ vertices and $6k$ edges;
2. $\text{cr}_1(G) = \text{cr}_1(K)$;
3. $\text{cre}_1(G) = \text{cre}_1(K)$.

Proof. (1) After reducing a graph with cyclomatic number k to its 2-core and reducing all maximal degree two paths to single edges we have a graph with $2k - 2$ vertices and $3k - 3$ edges, by Lemma 1. Since we are then adding one vertex back to every path that was not a single edge in the original graph, K has at most $5k - 5 \leq 5k$ vertices and $6k - 6 \leq 6k$ edges.

(2) First we show that $\text{cr}_1(G) \leq \text{cr}_1(K)$. Suppose that K has been embedded in one page with the minimum number of crossings. Every degree two vertex v in K corresponds to a path of degree two vertices in G . We can expand this path in a small neighborhood of v without introducing any new crossings. After expanding all degree two paths we have an embedding of the 2-core of G . Now each of the remaining vertices corresponds to a tree in G . Since trees can be embedded in one page without crossings, we can expand each tree in a small neighborhood of its corresponding vertex without introducing further crossings.

Now we show that $\text{cr}_1(K) \leq \text{cr}_1(G)$. Suppose that G is embedded on one page with minimum crossings. Reduce G to its 2-core; this does not increase crossings. Let u and v be two adjacent degree-two vertices of G , let e be the edge between u and v and let f be the edge from v not equal to e . Now, change the embedding of G by keeping u fixed and moving v next to u , rerouting f along the former path used by both e and f . This change moves all crossings from e to f but does

not create new crossings, so it produces another minimum-crossing embedding. After this change, edge e may be contracted, again without changing the crossing number. Repeatedly moving one of two adjacent degree-two vertices and then contracting their connecting edge eventually produces an embedding of K whose crossing number equals that of G .

(3) Follows from the proof of (2) with minor modification. \square

Lemma 3. *If G is a graph with n vertices and m edges, then $\text{cr}_1(G)$ and $\text{cre}_1(G)$ can be computed in $O(n!)$ time.*

Proof. We place the vertices in an arbitrary order on a circle, and compute the number of crossings or crossed edges for this layout. Then we use the Steinhaus–Johnson–Trotter algorithm [25] to list the $(n - 1)!$ permutations of all but one vertex efficiently, with consecutive permutations differing by a transposition. When a transposition swaps u and v , the number of crossings (or crossed edges) in the new layout can be updated from its previous value in $O(\deg(u) + \deg(v)) = O(n)$ time, as in [24]. This yields a total run time of $O(n!)$. \square

Combining the above lemmas, we apply the non-polynomial time algorithm only on the kernel of the graph to achieve the following fixed parameter result.

Theorem 1. *If G is a graph with cyclomatic number k , then $\text{cr}_1(G)$ and $\text{cre}_1(G)$ can be computed in $O((5k)! + n)$ time. If G is a k -almost-tree, then $\text{cr}_1(G)$ and $\text{cre}_1(G)$ can be computed in $O((5k)!n)$ time.*

In Section 6 we show how to improve the base of the factorial in this bound by applying fast matrix multiplication algorithms.

5 2-page Crossing Minimization

In this section we consider the problem of 2-page crossing minimization. I.e., we seek a circular arrangement of the vertices of a graph G , and an assignment of the edges to either the interior or exterior of the circle, such that the total number of crossings is minimized. As in the 1-page case, we consider minimizing both the number of crossings $\text{cr}_2(G)$ and the number of crossed edges $\text{cre}_2(G)$.

There are two sources of combinatorial complexity for this problem, the vertex ordering and the edge assignment. However, even when the vertex ordering is fixed, choosing an edge assignment to minimize crossings is NP-hard [26]. The hard instances of this problem can be chosen to be perfect matchings (with k -almost-tree parameter zero), so unless $\text{P} = \text{NP}$ there can be no FPT algorithm for the version of the problem with a fixed vertex ordering. Paradoxically, we show that requiring the algorithm to choose the ordering as well as the edge assignment makes the problem easier. A straightforward exact algorithm considers all $2^m(n - 1)!$ possible configurations and chooses the one minimizing the total number of crossings, running in $O(2^m n!)$ time. We will combine this fact with our kernelization to produce an FPT algorithm.

We will give a sequence of reduction rules that transforms any drawing of G into a drawing with the same number of crossings and crossed edges, in which

the lengths of all paths are bounded by a function $f(k)$ of the parameter k . These reductions will justify the correctness of our kernelization using the same function. Our reductions are based on the observation that, if uv is an uncrossed edge, and u and v are consecutive vertices on the spine, then edge uv can be contracted without changing the crossing number or number of uncrossed edges. A given layout may not have any uncrossed edges connecting consecutive vertices, but we will show that, for a graph with a long degree two path, the layout can be modified to produce edges of this type without changing its crossings.

Lemma 4. *Let G be a graph with cyclomatic number k . Then there exists a 2-page drawing with at most k crossed edges, and at most $\binom{k}{2}$ crossings.*

Proof. Remove k edges from G to produce a forest, F . Draw F without crossings on one page, and draw the remaining k edges on the other page. Only the k edges in the second page may participate in a crossing. □

We classify the possible configurations of pairs of consecutive edges of a degree two path, up to horizontal and vertical symmetries, into four possible types: m , s , $rainbow$, and $spiral$, as depicted in Figure 3.

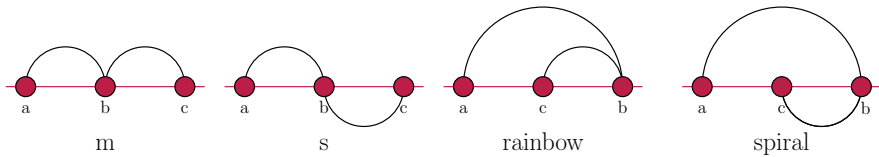


Fig. 3. Up to horizontal and vertical symmetry, the only possible arrangements of two consecutive edges are m , s , $rainbow$, and $spiral$

Lemma 5. *If a layout contains a pair of edges ab and bc of m or $rainbow$ type with edge bc uncrossed and with b and c both having degree two, then it can be reduced without changing its crossings by a rearrangement followed by a contraction of the edge bc .*

Proof. In either configuration we move vertex b adjacent to vertex c , on the opposite side of c from its other neighbor, as demonstrated in Figure 4. Since the edge bc is uncrossed this transformation does not change the crossing structure of the drawing. Now that b and c are placed next to each other the edge bc may be contracted. □

Lemma 6. *If a layout contains a pair of uncrossed edges ab and bc of s or $spiral$ type, with a , b , and c all having degree two, then the layout can be reduced without changing its crossings by a rearrangement and contraction.*

Proof. We assume by symmetry that a is the leftmost of the three vertices, edge ab is in the upper page, and edge bc is in the lower page. Let x be the neighbor of

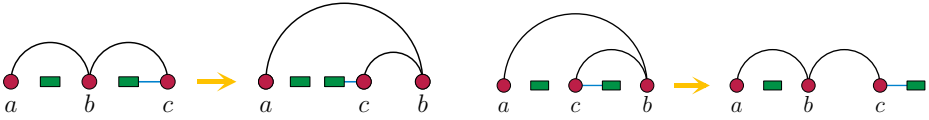


Fig. 4. The m reduction (left) and the rainbow reduction (right) shown with an edge into the β region

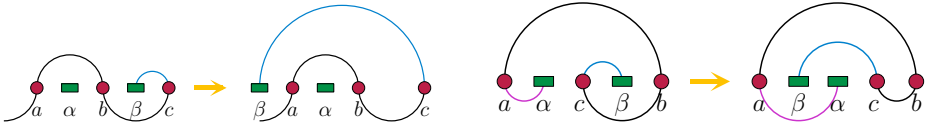


Fig. 5. The s reduction (left) shown with an edge into β and the spiral reduction (right) shown with edges into α and β

a that is not b and let y be the neighbor of c that is not b . We may assume that edge xa is in the lower page, for if it were in the upper page then edge ab would be part of an m or rainbow configuration and could be reduced by Lemma 5. By the same reasoning we may assume that cy is in the upper page.

First we consider s configurations. Let α be the set of vertices between a and b , and let β be the set of vertices between b and c . Then β can have no incoming edges in the lower page, because cy is upper and bc blocks all other edges. Therefore, we may move β directly to the left of a , as in Figure 5. Since edges ab and bc are uncrossed this transformation does not change the crossing structure of the drawing. We can then contract edge ab .

For the spiral, assume by symmetry that c is between a and b . Let α be the set of vertices between a and c , and let β be the set of vertices between c and b . Because cy is assumed to be in the upper page, and bc blocks all other lower edges, β can have no incoming lower edges; however, it might have edges in the upper page connecting it to α , so we must be careful to avoid twisting those connections and introducing new crossings. In this case, we move β between a and α and contract edge bc . \square

As shown above, if any degree two path has at least four edges and two consecutive uncrossed edges, then we can apply one of the reduction rules and reduce the number of edges. For this reason we define the kernel K for computing $cr_2(G)$ using the method in Section 3, with the bound $\ell(k) = 2k^2$ on the length of the maximal degree two paths. Similarly, we define the the kernel L for computing $cre_2(G)$ by setting $\ell(k) = 2k$.

Lemma 7. *Let G be a graph with cyclomatic number k . Then,*

1. K has at most $6k^3$ vertices and $6k^3$ edges;
2. $cr_2(G) = cr_2(K)$;
3. L has at most $6k^2$ vertices and $6k^2$ edges;
4. $cre_2(G) = cre_2(L)$

Proof. (1) Since we have at most $2k^2$ vertices per maximal degree two path, the total number of vertices is at most $2k^2(3k - 3) + 2k - 2 \leq 6k^3$. The number of edges is at most $2k^2(3k - 3) + (2k - 2) + (k - 1) \leq 6k^3$.

(2) The proof that $cr_2(G) \leq cr_2(K)$ is that same as in Lemma 2. To see that $cr_2(K) \leq cr_2(G)$ we suppose that G has been given an embedding that minimizes $cr_2(G)$. The total number of crossings in such an embedding is bounded above by $\binom{k}{2} < k^2/2$, and in turn the number of crossed edges is less than k^2 . Thus any maximal degree two path in G with length greater than $2k^2$ can be shortened.

(3) and (4) The proof follows by the same argument as in (1) and (2), noting that there always exists a drawing with at most k crossed edges. \square

We apply the straightforward exact algorithm to the kernel of the graph to achieve the following result:

Theorem 2. *If G is a graph with cyclomatic number k , then $cr_2(G)$ can be computed in $O(2^{6k^3}(6k^3)!+n)$ time, and $cre_2(G)$ can be computed in $O(2^{6k^2}(6k^2)!+n)$ time. If G is a k -almost-tree, then $cr_2(G)$ and $cre_2(G)$ can be computed in $O(2^{6k^3}(6k^3)!n)$ time and $O(2^{6k^2}(6k^2)!n)$ time respectively.*

6 Matrix Multiplication Improvement

The asymptotic run time for processing each biconnected component in both the one page and two page cases can be further improved using matrix multiplication to find the minimum weight triangle in a graph [27].

We begin with the 1-page case, in which we improve the run time to $O(k^{O(1)}(5k)!^{\omega/3})$ where $\omega < 2.3727$ is the exponent for matrix multiplication [28]. Let $N \leq 5k$ be the number of vertices in the kernel K , and for simplicity of exposition, assume that N is a multiple of 3. We construct a new graph G' as follows. For each subset $S \subset K$ of $N/3$ vertices in the original kernel K , and for each ordering of S , we create one vertex in G' . Thus, the number of vertices in G' is $(N/3)! \cdot \binom{N}{N/3} = O((N!)^{1/3})$. We add edges in G' between pairs of vertices that represent disjoint subsets. G' has a triangle for every triple of subsets that form a proper partition of V in G . Thus, each triangle corresponds to an assignment of the vertices to three uniformly sized regions and a distinct ordering of the vertices in each region, which together form a complete layout of G .

We assign a weight to each edge in G' based on the number of edge crossings in G between the vertices in the corresponding regions. There are four possible types of crossing, represented by α , β , γ and δ in Figure 6. For a crossing of type α , in which all endpoints of a pair of crossing edges in G are contained in the same region B , we add $1/2$ to the weights of edges AB and BC in G' . For β , in which

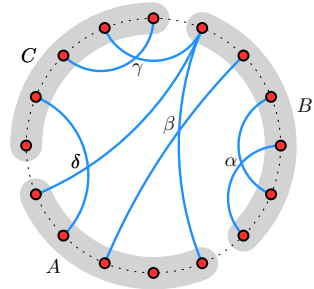


Fig. 6. Types of crossings

a pair of crossing edges in G both start in a region A and end in another region B , we add 1 to the weight of edge AB in G' . For γ , in which three endpoints of a pair of edges lie in the same region C , and the fourth lies in a different region B , we add 1 to the weight of edge BC in G' . Finally, for δ , in which a pair of crossed edges both have an endpoint in one region A , but their other endpoint in two different regions B and C , we add $1/2$ to the weight of edge AC and $1/2$ to the weight of edge AB in G' . With these weights, the total weight of a triangle in G' equals the number of edge crossings in the corresponding layout. The edge weights for G' can be computed in $O(k^{O(1)}(5k)!^{2/3})$ time.

To find the minimum weight triangle we construct the weighted adjacency matrix A , where $A_{i,j}$ is given the weight of the edge from i to j or infinity if no such edge exists. We then compute the min-plus matrix product of A with itself, which is defined by $[A \star A]_{i,j} = \min_k A_{i,k} + A_{k,j}$. The weight of a minimum weight triangle in A then corresponds to the minimum entry in $A + A \star A$. From the minimum weight and corresponding i and j the triangle can be found in linear time. Thus, the runtime is dominated by computing $A \star A$, which can be done in $O(k^{O(1)}(5k)!^{\omega/3})$ time using fast matrix multiplication [29, 30].

For the 2-page case we consider each of the 2^M edge page assignments separately, computing the minimum crossing drawing for this assignment using matrix multiplication. As before we construct a graph G' with weighted edges between compatible vertices, such that a minimum weight triangle in G' corresponds to a minimum weight drawing. Matrix multiplication is then used to find this minimal weight triangle for each page assignment, yielding a running time of $O(2^M(N!)^{\omega/3})$, where N is the number of vertices and M is the number of edges in the kernel. Thus, we have the following result:

Theorem 3. *If G is a graph with cyclomatic number k , then we can compute:*

- $\text{cr}_1(G)$ and $\text{cre}_1(G)$ in $O(k^{O(1)}(5k)!^{\omega/3} + n)$ time;
- $\text{cr}_2(G)$ in $O(2^{6k^3}(6k^3)!^{\omega/3} + n)$ time;
- $\text{cre}_2(G)$ in $O(2^{6k^2}(6k^2)!^{\omega/3} + n)$ time.

7 Conclusion

We have given new fixed parameter algorithms for computing the minimum number of edge crossings and minimum number of crossed edges in 1-page and 2-page embeddings of k -almost trees. To our knowledge, these are the only parameterized exact algorithms for these drawing styles.

We leave the following questions open to future research:

- For 2-page embeddings, the hardness of finding uncrossed drawings [13] shows that crossing minimization cannot be FPT in its natural parameter, the number of crossings. What about 1-page embeddings?
- Can the dependence on k be reduced to singly exponential?
- What other NP-hard problems in graph drawing are FPT with respect to k ?

Acknowledgements. We thank Emma S. Spiro whose work with social networks led us to consider drawing almost-trees, and an anonymous reviewer for helpful suggestions in our 2-page kernelization. This research was supported in part by the National Science Foundation under grant 0830403 and 1217322, and by the Office of Naval Research under MURI grant N00014-08-1-1015.

References

- [1] Pach, J., Tóth, G.: Which crossing number is it anyway? *J. Combin. Theory Ser. B* 80(2), 225–246 (2000)
- [2] Gurevich, Y., Stockmeyer, L., Vishkin, U.: Solving NP-hard problems on graphs that are almost trees and an application to facility location problems. *J. ACM* 31(3), 459–473 (1984)
- [3] Fernández-Baca, D.: Allocating modules to processors in a distributed system. *IEEE Transactions on Software Engineering* 15(11), 1427–1436 (1989)
- [4] Kloks, T., Bodlaender, H., Müller, H., Kratsch, D.: Computing treewidth and minimum fill-in: All you need are the minimal separators. In: Lengauer, T. (ed.) *ESA 1993. LNCS*, vol. 726, pp. 260–271. Springer, Heidelberg (1993)
- [5] Akutsu, T., Hayashida, M., Ching, W.K., Ng, M.K.: Control of Boolean networks: Hardness results and algorithms for tree structured networks. *J. Theor. Bio.* 244(4), 670–679 (2007)
- [6] Fiala, J., Kloks, T., Kratochvíl, J.: Fixed-parameter complexity of λ -labelings. *Discrete Appl. Math.* 113(1), 59–72 (2001)
- [7] Coppersmith, D., Vishkin, U.: Solving NP-hard problems in ‘almost trees’: Vertex cover. *Discrete Appl. Math.* 10(1), 27–45 (1985)
- [8] Bodlaender, H.: Dynamic algorithms for graphs with treewidth 2. In: van Leeuwen, J. (ed.) *WG 1993. LNCS*, vol. 790, pp. 112–124. Springer, Heidelberg (1994)
- [9] Fürer, M.: Counting perfect matchings in graphs of degree 3. In: Kranakis, E., Krizanc, D., Luccio, F. (eds.) *FUN 2012. LNCS*, vol. 7288, pp. 189–197. Springer, Heidelberg (2012)
- [10] Grohe, M.: Computing crossing numbers in quadratic time. In: *ACM Symp. Theory of Computing (STOC 2001)*, pp. 231–236 (2001)
- [11] Kawarabayashi, K.I., Reed, B.: Computing crossing number in linear time. In: *ACM Symp. Theory of Computing (STOC 2007)*, pp. 382–390 (2007)
- [12] Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Crossing number of graphs with rotation systems. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) *GD 2007. LNCS*, vol. 4875, pp. 3–12. Springer, Heidelberg (2008)
- [13] Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: Embedding graphs in books: A layout problem with applications to VLSI design. *SIAM J. Alg. Disc. Meth.* 8(1), 33–58 (1987)
- [14] Potterat, J.J., Phillips-Plummer, L., Muth, S.Q., Rothenberg, R.B., Woodhouse, D.E., Maldonado-Long, T.S., Zimmerman, H.P., Muth, J.B.: Risk network structure in the early epidemic phase of HIV transmission in Colorado Springs. *Sexually Transmitted Infections* 78(suppl. 1), i159–i163 (2002)
- [15] De, P., Singh, A.E., Wong, T., Yacoub, W., Jolly, A.M.: Sexual network analysis of a gonorrhoea outbreak. *Sexually Transmitted Infections* 80(4), 280–285 (2004)
- [16] Potterat, J.J., Muth, S.Q., Rothenberg, R.B., Zimmerman-Rogers, H., Green, D.L., Taylor, J.E., Bonney, M.S., White, H.A.: Sexual network structure as an indicator of epidemic phase. *Sexually Transmitted Infections* 78(suppl. 1), i152–i158 (2002)

- [17] Lombardi, M., Hobbs, R.: Mark Lombardi: Global Networks. Independent Curators (2003)
- [18] Butts, C.T., Acton, R.M., Marcum, C.S.: Interorganizational collaboration in the Hurricane Katrina response. *J. Social Structure* 13(1) (2012)
- [19] Seidman, S.B.: Network structure and minimum degree. *Social Networks* 5(3), 269–287 (1983)
- [20] de Nooy, W., Mrvar, A., Batagelj, V.: *Exploratory Social Network Analysis with Pajek*. Cambridge University Press (2012)
- [21] Watts, D., Strogatz, S.: Collective dynamics of ‘small-world’ networks. *Nature* 393, 440–442 (1998)
- [22] Bannister, M.J., Cabello, S., Eppstein, D.: Parameterized complexity of 1-planarity. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 97–108. Springer, Heidelberg (2013)
- [23] Masuda, S., Kashiwabara, T., Nakajima, K., Fujisawa, T.: On the NP-completeness of a computer network layout problem. In: 20th IEEE Symp. Circuits and Systems, pp. 292–295 (1987)
- [24] 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)
- [25] Sedgewick, R.: Permutation generation methods. *ACM Comput. Surv.* 9(2), 137–164 (1977)
- [26] Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Crossing minimization in linear embeddings of graphs. *IEEE Trans. Computers* 39(1), 124–127 (1990)
- [27] Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.* 348(2-3), 357–365 (2005)
- [28] Williams, V.V.: Multiplying matrices faster than Coppersmith–Winograd. In: ACM Symp. Theory of Computing (STOC 2012), pp. 887–898 (2012)
- [29] Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. *J. Comput. Sys. Sci.* 54(2), 255–262 (1997)
- [30] Yuval, G.: An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Inf. Proc. Lett.* 4(6), 155–156 (1976)

Strict Confluent Drawing

David Eppstein¹, Danny Holten², Maarten Löffler³,
Martin Nöllenburg⁴, Bettina Speckmann⁵, and Kevin Verbeek⁶

¹ Computer Science Department, University of California, Irvine, USA
epstein@uci.edu

² Synerscope BV, Eindhoven, the Netherlands
danny.holten@synerscope.com

³ Department of Computing and Information Sciences, Utrecht University, The Netherlands
m.loffler@uu.nl

⁴ Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
noellenburg@kit.edu

⁵ Department of Mathematics and Computer Science,
Technical University Eindhoven, The Netherlands
speckman@win.tue.nl

⁶ Department of Computer Science, University of California, Santa Barbara, USA
kverbeek@cs.ucsb.edu

Abstract. We define *strict confluent drawing*, a form of confluent drawing in which the existence of an edge is indicated by the presence of a smooth path through a system of arcs and junctions (without crossings), and in which such a path, if it exists, must be unique. We prove that it is NP-complete to determine whether a given graph has a strict confluent drawing but polynomial to determine whether it has an *outerplanar* strict confluent drawing with a fixed vertex ordering (a drawing within a disk, with the vertices placed in a given order on the boundary).

1 Introduction

Confluent drawing is a style of graph drawing in which edges are not drawn explicitly; instead vertex adjacency is indicated by the existence of a smooth path through a system of arcs and junctions that resemble train tracks. These types of drawings allow even very dense graphs, such as complete graphs and complete bipartite graphs, to be drawn in a planar way [4]. Since its introduction, there has been much subsequent work on confluent drawing [7,6,9,10,13,17], but the complexity of confluent drawing has remained unclear: how difficult is it to determine whether a given graph has a confluent drawing? Confluent drawings have a certain visual similarity to a graph drawing technique called *edge bundling* [3,5,11,12,14], in which “similar” edges are routed together in “bundles”, but we note that these drawings should be interpreted differently. In particular, sets of edges bundled together form visual junctions, however, interpreting them as confluent junctions can create false adjacencies.

Formally, a confluent drawing may be defined as a collection of *vertices*, *junctions* and *arcs* in the plane, such that all arcs are smooth and start and end at either a junction or a vertex, such that arcs intersect only at their endpoints, and such that all arcs that meet at a junction share the same tangent line there. A confluent drawing D represents a graph G defined as follows: the vertices of G are the vertices of D , and there is an

edge between two vertices u and v if and only if there exists a smooth path in D from u to v that does not pass any other vertex. (In some variants of confluent drawing an additional restriction is made that the smooth path may not intersect itself [13]; however, this constraint is not relevant for our work.)

Contribution. In this paper we introduce a subclass of confluent drawings, which we call *strict* confluent drawings. Strict confluent drawings are confluent drawings with the additional restrictions that between any pair of vertices there can be *at most one* smooth path, and there cannot be any paths from a vertex to itself. Figure 1 illustrates the forbidden configurations. To avoid irrelevant components in the drawing, we also require all arcs of the drawing to be part of at least one smooth path representing an edge. We believe that these restrictions may make strict drawings easier to read, by reducing the ambiguity caused by the existence of multiple paths between vertices. In addition, as we show, the assumption of strictness allows us to completely characterize their complexity, the first such characterization for any form of confluence on arbitrary undirected graphs.

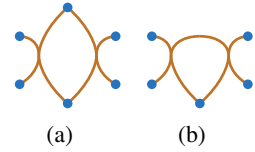


Fig. 1. (a) A drawing with a duplicate path. (b) A drawing with a self-loop.

We prove the following:

- It is NP-complete to determine whether a given graph has a strict confluent drawing.
- For a given graph, with a given cyclic ordering of its vertices, there is a polynomial time algorithm to find an *outerplanar* strict confluent drawing, if it exists: this is a drawing in a disk, with the vertices in the given order on the boundary of the disk
- When a graph has an outerplanar strict confluent drawing, an algorithm based on circle packing can construct a layout of the drawing in which every arc is drawn using at most two circular arcs.

See Fig. 2(a) for an example of an outerplanar strict confluent drawing. Previous work on *tree-confluent* [13] and *delta-confluent drawings* [6] characterized special cases of outerplanar strict confluent drawings as being the chordal bipartite graphs and distance-hereditary graphs respectively, so these graphs as well as the outerplanar graphs are all outerplanar strict confluent. The six-vertex wheel graph in Fig. 2(b) provides an example

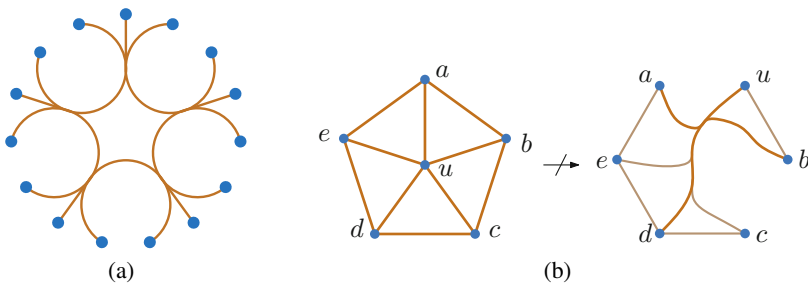


Fig. 2. (a) Outerplanar strict confluent drawing of the GD2011 contest graph. (b) A graph with no outerplanar strict confluent drawing.

of a graph that does not have an outerplanar strict confluent drawing. (The central vertex u needs to be placed between two of the outer vertices, say, a and b . The smooth path from u to the opposite vertex d separates a and b , so there must be a junction shared by the u - d and a - b paths, creating a wrong adjacency with d .)

2 Preliminaries

Let $G = (V, E)$ be a graph. We call an edge e in a drawing D *direct* if it consists only of a single arc (that does not pass through junctions). We call the angle between two consecutive arcs at a junction or vertex *sharp* if the two arcs do not form a smooth path; each junction has exactly two angles that are not sharp, and every angle at a vertex is sharp (so the number of sharp angles equals the degree of the vertex).

Lemma 1. *Let G be a graph, and let $E' \subseteq E$ be the edges of E that are incident to at least one vertex of degree 2. If G has a strict confluent drawing D , then it also has a strict confluent drawing D' in which all edges in E' are direct.*

Proof. Let v be a degree-2 vertex in G with two incident edges e and f . We consider the representation of e and f in D and modify D so that e and f are single arcs. There are two cases. If e and f leave v on two disjoint paths, then these paths have only merge junctions from v 's perspective. We can simply separate these junctions from e and f as shown in Fig. 3(a). If, on the other hand, e and f share the same path leaving v , then their paths split at some point. We need to reroute the merge junctions prior to the split and separate the merge junctions after the split as shown in Fig. 3(b). This is always possible since v has no other incident edges. Because D was strict and these changes do not affect strictness, D' is still a strict confluent drawing and edges e and f are direct. \square

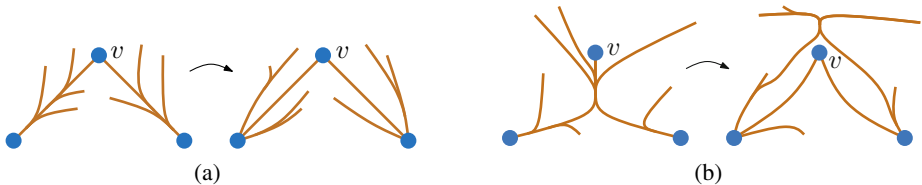


Fig. 3. The two cases of creating single arcs for edges incident to a degree-2 vertex

Lemma 2. *Let G be a graph. If G has no $K_{2,2}$ as a subgraph, whose vertices have degrees ≥ 3 in G , then G has a strict confluent drawing if and only if G is planar.*

Proof. Since every planar drawing is also a strict confluent drawing, that implication is obvious. So let D be a strict confluent drawing for a graph G without a $K_{2,2}$ subgraph, whose vertices have degrees ≥ 3 in G . Since larger junctions, where more than three arcs meet, can easily be transformed into an equivalent sequence of binary junctions, we can assume that every junction in D is binary, i.e., two arcs merge into one (or, from

a different perspective, one arc splits into two). By Lemma 1 we can further transform D so that all edges incident to degree-2 vertices are direct. Now for any vertex u in D none of its outgoing paths to some neighbor v can visit a merge junction before visiting a split junction as this would imply either a non-strict drawing or a $K_{2,2}$ subgraph with vertex degrees ≥ 3 . So the sequence of junctions on any u - v path consists of a number of split junctions followed by a number of merge junctions. But any such path can be unbundled from its junctions to the left and right and turned into a direct edge without creating arc intersections as illustrated in Fig. 4. This shows that D can be transformed into a standard planar drawing of G . \square



Fig. 4. Any strict confluent drawing of a graph without a $K_{2,2}$ subgraph can be transformed into a standard planar drawing

Lemma 3 characterizes the combinatorial complexity of strict confluent drawings. Its proof is found in the full paper [8] and uses Euler's formula and double counting.

Lemma 3. *The combinatorial complexity of any strict confluent drawing D of a graph G , i.e., the number of arcs, junctions, and faces in D , is linear in the number of vertices of G .*

Lemma 3 is in contrast to previous methods for confluent drawing interval graphs [4] and for drawing confluent Hasse diagrams [9], both of which may produce (non-strict) drawings with quadratically many features.

3 Computational Complexity

We will show by a reduction from planar 3-SAT [15] that it is NP-complete to decide whether a graph G has a strict confluent drawing in which all edges incident to degree-2 vertices are direct. By Lemma 1, this is enough to show that it is also NP-complete to decide if G has any strict confluent drawing.

Consider the subdivided grid graph (a grid with one extra vertex on each edge). In this graph, all edges are adjacent to a degree 2 vertex. Since a grid graph more than one square wide has only one fixed planar embedding (up to choice of the outer face), the subdivided grid graph has only one confluent embedding in which all edges are direct. We will base our construction on a number of such grids.

Let S be a planar 3-SAT formula. Globally speaking, we will create a grid graph for each variable of S , of size depending on the number of clauses that the variable appears in. The external edges of this grid graph are alternately colored green and red. We connect the variable graphs by identifying certain vertices: for each of the three variables that appear in a clause, we select one subdivided edge (that is, three vertices connected by two edges) on the outer face, and identify the endpoints of these edges

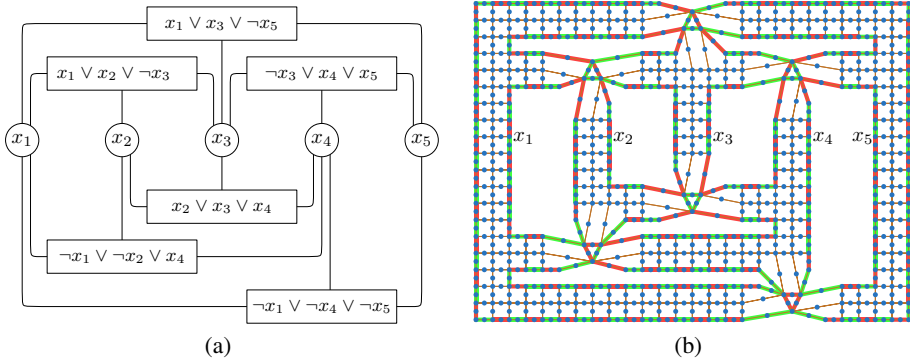


Fig. 5. (a) A planar 3-SAT formula. (b) The corresponding global frame of the construction: one grid graph per variable, with some vertices identified at each clause. Green boundary edges correspond to positive literals, red edges to negated literals. For easier readability the grids in this figure are larger than strictly necessary.

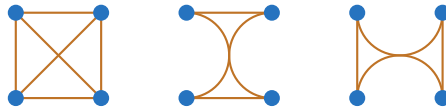


Fig. 6. K_4 and its two strict confluent drawings, without moving the vertices and keeping all arcs inside the convex hull of the vertices

into a triangle of subdivided edges (that is, a 6-cycle). We choose a green edge for a positive occurrence of the variable and a red edge for a negated occurrence. This will become clear below. We call the resulting graph F the *frame* of the construction; all edges of F are adjacent to a degree-2 vertex and F has only one planar embedding (up to choice of the outer face). Figure 5 shows an example.

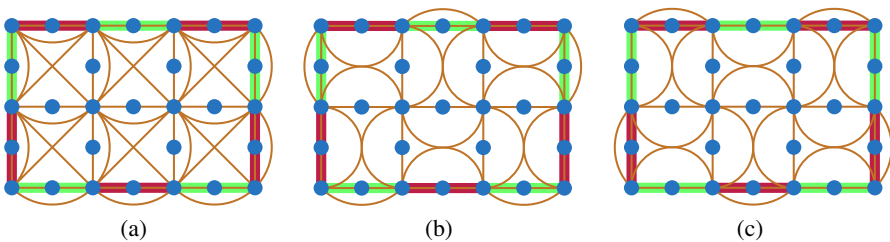


Fig. 7. (a) A variable gadget consists of a grid of K_4 's. Green (light) edges of the frame highlight normal literals, red (dark) edges negated ones. (b) One of the two possible strict confluent drawings, corresponding to the value *true*. (c) The other strict confluent drawing, corresponding to *false*.

The main idea of the construction is based on the fact that K_4 , when drawn with all four vertices on the outer face, has exactly two strict confluent drawings: we need to create a junction that merges the diagonal edges with one pair of opposite edges, and we can choose the pair. Figure 6 illustrates this. We will add a copy of K_4 to every cell of the frame graph F . Recall that every cell, except for the triangular clause faces, is a subdivided square (that is, an 8-cycle). We add K_4 on the four grid vertices (not the subdivision vertices). The edges that connect external grid vertices are called *literal edges*. Figure 7(a) shows this for a small grid. Since neighboring grid cells share a (subdivided) edge, the K_4 's are not edge-independent. This implies that in a strict confluent drawing, we cannot “use” such a common edge in both cells. Therefore, we need to orient the K_4 -junctions alternatingly, as illustrated in Figures 7(b) and 7(c). If the grid is sufficiently large (every cell is part of a larger at least size- (2×2) grid) these choices are completely propagated through the entire grid, so there are two structurally different possible embeddings, which we use to represent the values *true* and *false* of the corresponding variable. For every green edge of the frame in the *true* state and every red edge in the *false* state there is one remaining literal edge in the outer face, which can still be drawn either inside or outside their grid cells. In the opposite states these literal edges are needed inside the grid cells to create the K_4 junctions. The availability of at least one literal edge (corresponding to a *true* literal) is important for satisfying the clause gadgets, which we describe next.

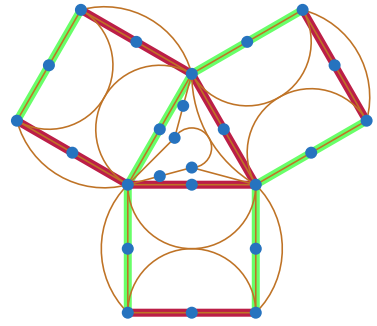


Fig. 8. Three variables attached to a clause gadget. The top left variable occurs in the clause as a positive literal, the others as negative literals. The clause can be satisfied because the top right variable is set to *false*.

Inside each triangular clause face, we add the graph depicted in Figure 9(a). This graph has several strict confluent drawings; however, in every drawing at least one of the three outer edges needs to be drawn inside the subdivided triangle.

Lemma 4. *There is no strict confluent drawing of the clause graph in which all three long edges are drawn outside. Moreover, there is a strict confluent drawing of the clause graph with two of these edges outside, for every pair.*

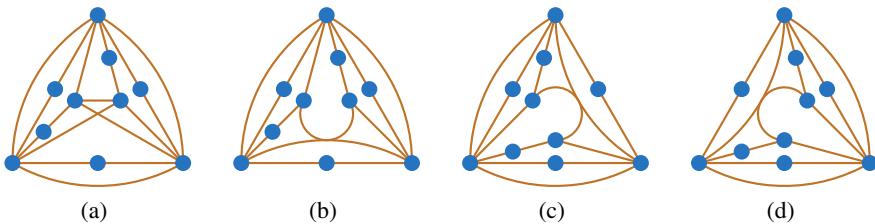


Fig. 9. (a) The input graph of the clause. (b, c, d) Three different strict confluent drawings

Proof. Recall that by Lemma 1 the subdivided triangle must be embedded as a 6-cycle of direct arcs. To prove the first part of the lemma, assume that the triangle edges are all drawn outside this cycle. The remainder of the graph has no 4-cycles without subdivision vertices (that is, no $K_{2,2}$ with higher-degree vertices), so by Lemma 2 it can only have a strict confluent drawing if it is planar. However, it is a subdivided K_5 , which is not planar. To prove the second part of the lemma, we refer to Figures 9(b), 9(c) and 9(d). \square

This describes the reduction from a planar 3-SAT instance to a graph consisting of variable and clause gadgets. Next we show that this graph has a strict confluent drawing if and only if the planar 3-SAT formula is satisfiable. For a given satisfying assignment we choose the corresponding embeddings of all variable gadgets. The assignment has at least one *true* literal per clause, and correspondingly in each clause gadget one of the three literal edges can be drawn inside the clause triangle, allowing a strict confluent drawing by Lemma 4. Conversely, in any strict confluent drawing, each clause must be drawn with at least one literal edge inside the clause triangle by Lemma 4, so translating the state of each variable gadget into its truth value yields a satisfying assignment.

To show that testing strict confluence is in NP, recall that by Lemma 3 the combinatorial complexity of the drawing is linear in the number of vertices. Thus the existence of a drawing can be verified by guessing its combinatorial structure and verifying that it is planar and a drawing of the correct graph.

Theorem 1. *Deciding whether a graph has a strict confluent drawing is NP-complete.*

4 Outerplanar Strict Confluent Drawings

For a graph G with a fixed cyclic ordering of its vertices, we can test in polynomial time whether an outerplanar strict confluent drawing with this vertex ordering exists, and, if so, construct one. This algorithm uses the closely related notion of a canonical diagram of G , which is unique and exists if and only if an outerplanar strict confluent drawing exists. From the canonical diagram a confluent drawing can be constructed. We further show that the drawing can be constructed such that every arc consists of at most two circular arcs.

4.1 Canonical Diagrams

We define a *canonical diagram* to be a collection of junctions and arcs connecting the vertices in the given order on the outer face (as in a confluent drawing), but with some of the faces of the diagram *marked*, satisfying additional constraints enumerated below. Figure 10 shows a canonical diagram and an outerplanar strict confluent drawing of the same graph. In such a diagram, a *trail* is a smooth curve from one vertex to another that follows the arcs (as in a confluent drawing) but is allowed to cross the interior of marked faces from one of its sharp corners to another. The constraints are:

- Every arc is part of at least one trail.
- No two trails between the same two vertices can follow different sequences of arcs and faces.

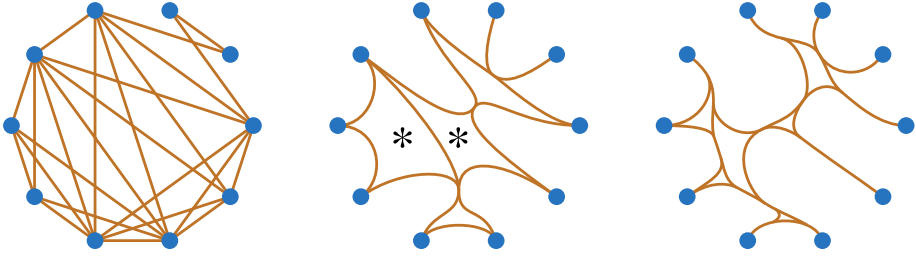


Fig. 10. Three views of the same graph as a node-link diagram (left), canonical diagram (center), and outerplanar strict confluent drawing (right)

- Each marked face must have at least four angles, all of which are sharp.
- Each arc must have either sharp angles or vertices at both of its ends.
- For each junction j with exactly two arcs in each direction, let f and f' be the two faces with sharp angles at j . Then it is not allowed for f and f' to both be either marked or to be a triangle (a face with three angles, all sharp).

Let j be a junction of a canonical diagram D . Then define the *funnel* of j to be the 4-tuple of vertices a, b, c, d where a is the vertex reached by a path that leaves j in one direction and continues as far clockwise as possible, b is the most counterclockwise vertex reachable in the same direction from j , c is the most clockwise vertex reachable in the other direction, and d is the most counterclockwise vertex reachable in the other direction. Note that none of the paths from j to $a, b, c,$ and d can intersect each other without contradicting the uniqueness of trails. We call the circular intervals of vertices $[a, b]$ and $[c, d]$ (in the counterclockwise direction) the *funnel intervals* of the respective funnel. We say a circular interval $[a, b]$ is *separated* if either a and b are not adjacent in G , or there exists a junction in the canonical diagram with funnel intervals $[a, e]$ and $[f, b]$, where $e, f \in [a, b]$.

A canonical diagram represents a graph G in which the edges in G correspond to trails in the diagram. As we show in the full paper [8], a graph G has a canonical diagram if and only if it has an outerplanar strict confluent drawing, and if a canonical diagram exists then it is unique.

4.2 Algorithm

By using the properties of canonical diagrams (see the full paper [8]), we may obtain an algorithm that constructs a canonical diagram and strict confluent drawing of a given cyclically-ordered graph G , or reports that no drawing exists, in time and space $O(n^2)$. This bound is optimal in the worst case, as it matches the input size of a graph that may have quadratically many edges.

Steps 1–3 of the algorithm, detailed below, build some simple data structures that speed up the subsequent computations. Step 4 discovers all of the funnels in the input, from which it constructs a list of all of the junctions of the canonical diagram. Step 5 connects these junctions into a planar drawing, a subset of the canonical diagram. Step 6

builds a graph for each face of this drawing that will be used to complete it into the entire canonical diagram, and step 7 uses these graphs to find the remaining arcs of the diagram and to determine which faces of the diagram are marked. Step 8 checks that the diagram constructed by the previous steps correctly represents the input graph, and step 9 splits the marked faces, converting the diagram into a strict confluent drawing.

1. Number the vertices clockwise around the boundary cycle from 0 to $n - 1$.
2. Build a table, containing for each pair i, j , the number of ordered pairs (i', j') with $i' \leq i, j' \leq j$, and vertices i' and j' adjacent in G . By performing a constant number of lookups in this table we may determine in constant time how many edges exist between any two disjoint intervals of the boundary cycle.
3. Build a table that lists, for each ordered pair u, v of vertices, the neighbor w of u that is closest in clockwise order to v . That is, w is adjacent to u , and the interval from v clockwise to w contains no other neighbors of u . The table entries for u can be found in linear time by a single counterclockwise scan. Repeat the same construction in the opposite orientation.
4. For each separated interval $[a, b]$, let c be the next neighbor of a that is counterclockwise of b , and let d be the next neighbor of b that is clockwise of a . If (i) c is a neighbor of b , (ii) d is a neighbor of a , (iii) a is the next neighbor of c that is counterclockwise of d , and (iv) b is the next neighbor of d that is clockwise of c , then (if a confluent diagram exists) a, b, c, d must form the funnel of a junction, and all funnels have this form. We check all circular intervals in increasing order of their cardinalities. For each discovered funnel, we mark the intervals that are separated by the corresponding junction. This way we can check in $O(1)$ time whether a circular interval is separated. If the number of funnels exceeds the linear bound of Lemma 3 on the number of junctions in a confluent drawing, abort the algorithm.
5. Create a junction for each of the funnels found in step 4. For each vertex v , make a set J_v of the junctions whose funnel includes that vertex; if they are to be drawn as part of a canonical diagram, the junctions of J_v need to be connected to v by a confluent tree. For any two junctions in J_v , it is possible to determine in constant time whether one is an ancestor of another in this tree, or if not whether one is clockwise of the other, by examining the cyclic ordering of vertices in their funnels. Construct the trees of junctions and their planar embedding in this way. The result of this stage of the algorithm should be a planar embedding of part of the canonical diagram consisting of all vertices and junctions, and the subset of the arcs that are part of a path from a junction to one of its funnel vertices. Check that the embedding is planar by computing its Euler characteristic, and abort the algorithm if it is not.
6. For each face f of the drawing created in step 5, and each pair j, j' of junctions belonging to f , use the data structure from step 2 to test whether there is an edge whose trail passes through both j and j' . This results in a graph H_f in which the vertices represent the vertices or junctions on the boundary of f and the edges represent pairs of vertices or junctions that must be connected, either by an arc or by shared membership in a marked face. The remaining arcs to be drawn in f will be exactly the edges of H_f that are not crossed by other edges of H_f ; the marked faces in f will be exactly the faces that contain pairs of crossing edges of H_f .
7. Within each face f of the drawing so far, build a table using the same construction as in step 2 that can be used to determine the existence of a crossing edge for an

- edge in H_f in constant time. Use this data structure to identify the crossed edges, and draw an arc in f for each uncrossed edge. For each face g of the resulting subdivision of f , if g has four or more vertices or junctions, find two pairs that would cross and test whether both pairs correspond to edges in H_f ; if so, mark g .
8. Construct a directed graph that has a vertex for each vertex of G , two vertices for each junction of the diagram (one in each direction), two directed edges for each arc, and a directed edge for each ordered pair of sharp angles that are non-consecutive in a marked face. By performing a depth-first search in this graph, determine whether there exist multiple smooth paths in the resulting drawing from any vertex of G to any other point in the drawing, and abort the algorithm if any such pair of paths is found. Determine the set of vertices of G reachable from v and verify that it is the same set of vertices that are reachable in the original graph. Additionally, verify that the diagram satisfies the requirements in the definition of a canonical diagram. Abort the algorithm if any inconsistency is found in this step.
 9. Convert the canonical diagram into a confluent drawing and return it.

Theorem 2. *For a given n -vertex graph G , and a given circular ordering of its vertices, it is possible to determine whether G has an outerplanar strict confluent drawing with the given vertex ordering, and if so to construct one, in time $O(n^2)$.*

4.3 Drawings with Low Curve Complexity

Suppose that we are given a topological description of an outerplanar strict confluent drawing D of a connected graph G , describing the tangency pattern and ordering of the arcs at each junction. It still remains to draw D (or possibly an equivalent but combinatorially different outerplanar strict confluent drawing) in the plane using concrete curves for its arcs. If we ignore the tangency requirements at its junctions, the arcs and junctions of D form a planar graph, but applying standard planar graph drawing methods will generate arcs that may not be smooth and that are not tangent to each other at the junctions. So how are we to draw D ? Here we use a circle packing method to draw D with a small number of circular arcs for each arc of D . Thus, these drawings have low *curve complexity* in the sense of Bekos et al. [1], but with this complexity measured along arcs of the confluent diagram rather than edges of another type of graph drawing.

Given such a drawing D , let D' be a modified version of D in which every junction is incident to exactly three arcs, formed from D by suppressing two-arc junctions and splitting junctions with more than three arcs. Assume also (again by adding more junctions if necessary) that each vertex in D' has only a single arc incident to it.

Given the topological diagram D' , we form a planar graph H that has a vertex for each vertex or junction of D' , and an edge for each arc of D' . Additionally, we create an edge in H for each two vertices that are consecutive in the cyclic ordering of the vertices around the disk containing the drawing.

Lemma 5. *H is planar, 3-regular, and 3-vertex-connected.*

Proof. Planarity and 3-regularity follow immediately from the construction of H . Every two vertices of G are connected by three vertex-disjoint paths in H : at least one (not necessarily a smooth path) through D , using the assumption that G is connected, and two

more around the boundary of the disk. Therefore, if H were not 3-vertex-connected, only one of its 3-connected components could contain vertices of G . The other components would either contain components of D that are not part of any smooth path between vertices of G (forbidden in a strict confluent drawing) or would contain more than one smooth path between the same sets of vertices (also forbidden). \square

Theorem 3. *Let D be an outerplanar strict confluent drawing of a graph G , given topologically but not geometrically. Then we can construct an outerplanar strict confluent drawing of G in which each arc of the drawing is represented by a smooth curve that is either a circular arc or the union of two circular arcs.*

Proof. By the Koebe–Thurston–Andreev circle packing theorem, there exists a system C of circles representing the faces of H , such that two circles are adjacent exactly when the corresponding faces share an edge. We may assume (by performing a Möbius transformation if necessary) that the outer circle of this circle packing corresponds to the outer face of H . C may be found efficiently (although not in strongly polynomial time) by a numerical iteration that quickly converges to the system of radii of the circles, from which their centers can also be computed easily [2,16].

Each vertex of G corresponds in C to one of the triangular gaps between the outer circle and two other circles, and may be placed at the point of tangency of the two non-outer circles (one of the vertices of this triangle); see Fig. 11. The junctions in D' lie at the meeting point of three faces of H , and correspond in C to the remaining triangular gaps between three circles. A confluent drawing of G may be formed by removing the outer circle, removing all circular arcs bounding the triangular gaps incident to the outer circle, and in each remaining triangular gap removing the arc that is on the other side of the sharp angle. The resulting drawing contracts some edges of D' to form junctions with four incident arcs, but this does not affect the correctness of the drawing. In the resulting drawing, arcs of the diagram that have merge points or vertices at both of their endpoints are drawn as two circular arcs (possibly both from the same circle); other arcs of the diagram are drawn as a single circular arc. \square

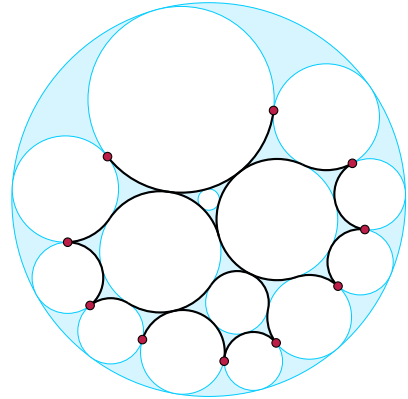


Fig. 11. Constructing an outerplanar strict confluent drawing from a circle packing. The vertices of the drawing correspond to triangular gaps adjacent to the outer circle, and the junctions to the remaining triangular gaps.

5 Conclusions

We have shown that, in confluent drawing, restricting attention to the strict drawings allows us to completely characterize their complexity, and we have also shown that outerplanar strict confluent drawings with a fixed vertex ordering may be constructed in

polynomial time. The most pressing problem left open by this research is to recognize the graphs that have outerplanar strict confluent drawings, without imposing a fixed vertex order. Can we recognize these graphs in polynomial time?

Acknowledgements. This work originated at Dagstuhl seminar 13151, *Drawing Graphs and Maps with Curves*. D.E. was supported in part by the National Science Foundation under grants 0830403 and 1217322, and by the Office of Naval Research under MURI grant N00014-08-1-1015. M.L. was supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123. M.N. received financial support by the ‘Concept for the Future’ of KIT under grant YIG 10-209.

References

1. Bekos, M.A., Kaufmann, M., Kobourov, S.G., Symvonis, A.: Smooth orthogonal layouts. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 150–161. Springer, Heidelberg (2013)
2. Collins, C.R., Stephenson, K.: A circle packing algorithm. *Comput. Geom. Theory Appl.* 25(3), 233–256 (2003)
3. Cui, W., Zhou, H., Qu, H., Wong, P.C., Li, X.: Geometry-based edge clustering for graph visualization. *IEEE TVCG* 14(6), 1277–1284 (2008)
4. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. *J. Graph. Algorithms Appl.* 9(1), 31–52 (2005)
5. 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)
6. Eppstein, D., Goodrich, M.T., Meng, J.Y.: Delta-confluent drawings. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 165–176. Springer, Heidelberg (2006)
7. Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent layered drawings. *Algorithmica* 47(4), 439–452 (2007)
8. Eppstein, D., Holten, D., Löffler, M., Nöllenburg, M., Speckmann, B., Verbeek, K.: Strict confluent drawing. *CoRR*, abs/1308.6824 (2013)
9. Eppstein, D., Simons, J.A.: Confluent hasse diagrams. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 2–13. Springer, Heidelberg (2011)
10. Hirsch, M., Meijer, H., Rappaport, D.: Biclique edge cover graphs and confluent drawings. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 405–416. Springer, Heidelberg (2007)
11. Holten, D.: Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE TVCG* 12(5), 741–748 (2006)
12. Holten, D., van Wijk, J.J.: Force-Directed Edge Bundling for Graph Visualization. *Computer Graphics Forum* 28(3), 983–990 (2009)
13. Hui, P., Pelsmajer, M.J., Schaefer, M., Štefankovič, D.: Train tracks and confluent drawings. *Algorithmica* 47(4), 465–479 (2007)
14. Hurter, C., Ersoy, O., Telea, A.: Graph Bundling by Kernel Density Estimation. *Computer Graphics Forum* 31(3pt. 1), 865–874 (2012)
15. Lichtenstein, D.: Planar formulae and their uses. *SIAM J. Comput.* 11(2), 329–343 (1982)
16. Mohar, B.: A polynomial time circle packing algorithm. *Discrete Math.* 117(1-3), 257–263 (1993)
17. Quercini, G., Ancona, M.: Confluent drawing algorithms using rectangular dualization. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 341–352. Springer, Heidelberg (2011)

A Ramsey-Type Result for Geometric ℓ -Hypergraphs

Dhruv Mubayi^{1,*} and Andrew Suk^{2,**}

¹ University of Illinois at Chicago, Chicago, IL
mubayi@math.uic.edu

² Massachusetts Institute of Technology, Cambridge, MA
asuk@math.mit.edu

Abstract. Let $n \geq \ell \geq 2$ and $q \geq 2$. We consider the minimum N such that whenever we have N points in the plane in general position and the ℓ -subsets of these points are colored with q colors, there is a subset S of n points all of whose ℓ -subsets have the same color and furthermore S is in convex position. This combines two classical areas of intense study over the last 75 years: the Ramsey problem for hypergraphs and the Erdős-Szekeres theorem on convex configurations in the plane. For the special case $\ell = 2$, we establish a single exponential bound on the minimum N such that every complete N -vertex geometric graph whose edges are colored with q colors, yields a monochromatic convex geometric graph on n vertices.

For fixed $\ell \geq 2$ and $q \geq 4$, our results determine the correct exponential tower growth rate for N as a function of n , similar to the usual hypergraph Ramsey problem, even though we require our monochromatic set to be in convex position. Our results also apply to the case of $\ell = 3$ and $q = 2$ by using a geometric variation of the Stepping-up lemma of Erdős and Hajnal. This is in contrast to the fact that the upper and lower bounds for the usual 3-uniform hypergraph Ramsey problem for two colors differ by one exponential in the tower.

1 Introduction

The classic 1935 paper of Erdős and Szekeres [12] entitled *A Combinatorial Problem in Geometry* was the starting point of a very rich discipline within combinatorics: Ramsey theory (see, e.g., [15]). The term *Ramsey theory* refers to a large body of deep results in mathematics which have a common theme: "Every large system contains a large well-organized subsystem." Motivated by the observation that any five points in the plane in general position¹ must contain four members in convex position, Esther Klein asked the following.

Problem 1. For every integer $n \geq 2$, determine the minimum $f(n)$ such that any set of $f(n)$ points in the plane in general position contains n members in convex position.

* Research partially supported by NSF grant DMS-0969092 and DMS-1300138.

** Supported by an NSF Postdoctoral Fellowship and by Swiss National Science Foundation Grant 200021-125287/1.

¹ A planar point set P is in *general position* if no three members are collinear.

Celebrated results of Erdős and Szekeres [12,13] imply that

$$2^{n-1} + 1 \leq f(n) \leq \binom{2n-4}{n-2} = 2^{2n(1+o(1))}. \tag{1}$$

They conjectured that $f(n) = 2^{n-1} + 1$, and Erdős offered a \$500 reward for a proof. Despite much attention over the last 75 years, the constant factors in the exponents have not been improved.

In the same paper, Erdős and Szekeres [12] gave another proof of a classic result due to Ramsey [23] on hypergraphs. An ℓ -uniform hypergraph H is a pair (V, E) , where V is the vertex set and $E \subset \binom{V}{\ell}$ is the set of edges. We denote $K_n^\ell = (V, E)$ to be the complete ℓ -uniform hypergraph on an n -element set V , where $E = \binom{V}{\ell}$. When $\ell = 2$, we write $K_n^2 = K_n$. Motivated by obtaining good quantitative bounds on $f(n)$, Erdős and Szekeres looked at the following problem.

Problem 2. For every integer $n \geq 2$, determine the minimum integer $r(K_n, K_n)$ such that any two-coloring on the edges of a complete graph G on $r(K_n, K_n)$ vertices yields a monochromatic copy of K_n .

Erdős and Szekeres [12] showed that $r(K_n, K_n) \leq 2^{2n}$. Later, Erdős [9] gave a construction showing that $r(K_n, K_n) > 2^{n/2}$. Despite much attention over the last 65 years, the constant factors in the exponents have not been improved.

Generalizing Problem 2 to q -color and ℓ -uniform hypergraphs has also be studied extensively. Let $r(K_n^\ell; q)$ be the least integer N such that any q -coloring on the edges of a complete N -vertex ℓ -uniform hypergraph H yields a monochromatic copy of K_n^ℓ . We will also write

$$r(K_n^\ell; q) = r(\underbrace{K_n^\ell, K_n^\ell, \dots, K_n^\ell}_{q \text{ times}}).$$

Erdős et al. [10,11] showed that there are positive constants c and c' such that

$$2^{cn^2} < r(K_n^3, K_n^3) < 2^{2^{c'n}}. \tag{2}$$

They also conjectured that $r(K_n^3, K_n^3) > 2^{2^{cn}}$ for some constant $c > 0$, and Erdős offered a \$500 reward for a proof. For $\ell \geq 4$, there is also a difference of one exponential between the known upper and lower bounds for $r(K_n^\ell, K_n^\ell)$, namely,

$$\text{twr}_{\ell-1}(cn^2) \leq r(K_n^\ell, K_n^\ell) \leq \text{twr}_\ell(c'n), \tag{3}$$

where c and c' depend only on ℓ , and the tower function $\text{twr}_\ell(x)$ is defined by $\text{twr}_1(x) = x$ and $\text{twr}_{i+1} = 2^{\text{twr}_i(x)}$. As Erdős and Rado [11] have shown, the upper bound in equation (3) easily generalizes to q colors, implying that $r(K_n^\ell; q) \leq \text{twr}_\ell(c'n)$, where $c' = c'(\ell, q)$. On the other hand, for $q \geq 4$ colors, Erdős and Hajnal (see [15]) showed that $r(K_n^\ell; q)$ does indeed grow as a ℓ -fold exponential tower in n , proving that $r(K_n^\ell; q) = \text{twr}_\ell(\Theta(n))$. For $q = 3$ colors, Conlon et al. [6] modified the construction of Erdős and Hajnal to show that $r(K_n^\ell, K_n^\ell, K_n^\ell) > \text{twr}_\ell(c \log^2 n)$.

Interestingly, both Problems 1 and 2 can be asked simultaneously for geometric graphs, and a similar-type problem can be asked for geometric ℓ -hypergraphs. A *geometric ℓ -hypergraph H in the plane* is a pair (V, E) , where V is a set of points in the plane in general position, and $E \subset \binom{V}{\ell}$ is a collection of ℓ -tuples from V . When $\ell = 2$ ($\ell = 3$), edges are represented by straight line segments (triangles) induced by the corresponding vertices. The sets V and E are called the *vertex set* and *edge set* of H , respectively. A geometric hypergraph H is *convex*, if its vertices are in convex position.

Geometric graphs ($\ell = 2$) have been studied extensively, due to their wide range of applications in combinatorial and computational geometry (see [17,18,22]). Complete convex geometric graphs are very well understood, and are some of the most *well-organized* geometric graphs (if not the most). Many long-standing problems on complete geometric graphs, such as its crossing number [2], number of halving-edges [26], and size of crossing families [3], become trivial when its vertices are in convex position. There has also been a lot of research on geometric 3-hypergraphs in the plane, due to its connection to the *k-set problem* in \mathbb{R}^3 (see [7,21,24]). In this paper, we study the following problem which combines Problems 1 and 2.

Problem 3. Determine the minimum integer $g(K_n^\ell; q)$ such that any q -coloring of the edges of a complete geometric ℓ -hypergraph H on $g(K_n^\ell; q)$ vertices yields a complete monochromatic convex ℓ -hypergraph on n vertices.

We will also write

$$g(K_n^\ell; q) = g(\underbrace{K_n^\ell, \dots, K_n^\ell}_{q \text{ times}}).$$

Clearly we have $g(K_n^\ell; q) \geq \max\{r(K_n^\ell; q), f(n)\}$. An easy observation shows that by combining equations (1) and (3), we also have

$$g(K_n^\ell; q) \leq f(r(K_n^\ell; q)) \leq \text{twr}_{\ell+1}(cn),$$

where $c = c(\ell, q)$. Our main results are the following two exponential improvements on the upper bound of $g(K_n^\ell; q)$.

Theorem 1. *For geometric graphs, we have*

$$2^{q(n-1)} < g(K_n; q) \leq 2^{8qn^2 \log(qn)}.$$

The argument used in the proof of Theorem 1 above extends easily to hypergraphs, and for each fixed $\ell \geq 3$ it gives the bound $g(K_n^\ell; q) < \text{twr}_\ell(O(n^2))$. David Conlon pointed out to us that one can improve this slightly as follows.

Theorem 2. *For geometric ℓ -hypergraphs, when $\ell \geq 3$ and fixed, we have*

$$g(K_n^\ell; q) \leq \text{twr}_\ell(cn),$$

where $c = O(q \log q)$.

By combining Theorems 1, 2, and the fact that $g(K_n^\ell; q) \geq r(K_n^\ell; q)$, we have the following corollary.

Corollary 1. For fixed ℓ and $q \geq 4$, we have $g(K_n^\ell; q) = \text{twr}_\ell(\Theta(n))$.

As mentioned above, there is an exponential difference between the known upper and lower bounds for $r(K_n^3, K_n^3)$. Hence, for two-colorings on geometric 3-hypergraphs in the plane, equation (2) implies

$$g(K_n^3, K_n^3) \geq r(K_n^3, K_n^3) \geq 2^{cn^2}.$$

Our next result establishes an exponential improvement in the lower bound of $g(K_n^3, K_n^3)$, showing that $g(K_n^3, K_n^3)$ does indeed grow as a 3-fold exponential tower in a power of n . One noteworthy aspect of this lower bound is that the construction is a geometric version of the famous Stepping-up lemma of Erdős and Hajnal [10] for sets. While it is a major open problem to apply this method to $r(K_n^3, K_n^3)$ and improve the lower bound in equation (2), we are able to achieve this in the geometric setting as shown below.

Theorem 3. For geometric 3-hypergraphs in the plane, we have

$$g(K_n^3, K_n^3) \geq 2^{2^{cn}},$$

where c is an absolute constant. In particular, $g(K_n^3, K_n^3) = \text{twr}_3(\Theta(n))$.

2 Proof of Theorems 1 and 2

Before proving Theorems 1 and 2, we will first define some notation. We let $V = \{p_1, \dots, p_N\}$ be a set of N points in the plane in general position ordered from left to right according to x -coordinate, that is, for $p_i = (x_i, y_i) \in \mathbb{R}^2$, we have $x_i < x_{i+1}$ for all i . For $i_1 < \dots < i_t$, we say that $X = (p_{i_1}, \dots, p_{i_t})$ forms an t -cup (t -cap) if X is in convex position and its convex hull is bounded above (below) by a single edge. See Figure 1. When $t = 3$, we will just say X is a cup or a cap.



Fig. 1. A 4-cup and a 5-cap

Proof of Theorem 1. We first prove the upper bound. Let $G = (V, E)$ be a complete geometric graph on $N = 2^{8qn^2 \lceil \log(qn) \rceil}$ vertices such that the vertices $V = \{v_1, \dots, v_N\}$ are ordered from left to right according to x -coordinate. Let χ be a q -coloring on the edge set E . We will recursively construct a sequence of vertices p_1, \dots, p_t from V and a subset $S_t \subset V$, where $t = 0, 1, \dots, qn^2$ (when $t = 0$ the sequence is empty), such that the following holds.

1. for any vertex $p_i, i = 1, \dots, qn^2$, all pairs (p_i, p) where $p \in \{p_j : j > i\} \cup S_t$ have the same color, which we denote by $\chi'(p_i)$,
2. for every pair of vertices p_i and p_j , where $i < j$, either (p_i, p_j, p) is a cap for all $p \in \{p_k : k > j\} \cup S_t$, or (p_i, p_j, p) is a cup for all $p \in \{p_k : k > j\} \cup S_t$,
3. the set of points S_t lies to the right of the point p_t , and
4. $|S_t| \geq \frac{N}{q^t t!} - t$.

We start with no vertices in the sequence ($t = 0$), and set $S_0 = V$. After obtaining vertices $\{p_1, \dots, p_t\}$ and S_t , we define p_{t+1} and S_{t+1} as follows. Let $p_{t+1} = (x_{t+1}, y_{t+1}) \in \mathbb{R}^2$ be the smallest indexed element in S_t (the left-most point), and let H be the right half-plane $x > x_{t+1}$. We define t lines l_1, \dots, l_t such that l_i is the line going through points p_i, p_{t+1} . Note that the arrangement $\bigcup_{i=1}^t l_i$ partitions the right half-plane H into $t + 1$ cells. See Figure 2. Since V is in general position, by the pigeonhole principle, there exists a cell $\Delta \subset H$ that contains at least $(|S_t| - 1)/(t + 1)$ points of S_t .

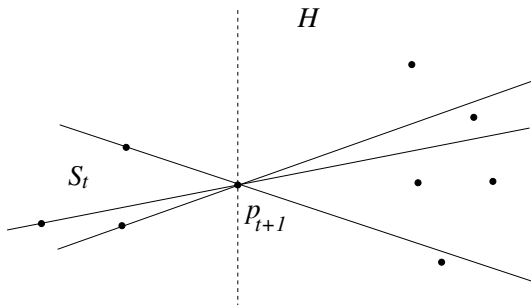


Fig. 2. Lines l_1, \dots, l_t partitioning the half-plane H

Let us call two elements $v'_1, v'_2 \in \Delta \cap S_t$ *equivalent* if $\chi(p_{t+1}, v'_1) = \chi(p_{t+1}, v'_2)$. Hence, there are at most q equivalence classes. By setting S_{t+1} to be the largest of those classes, we have the recursive formula

$$|S_{t+1}| \geq \frac{|S_t| - 1}{(t + 1)q}.$$

Substituting in the lower bound on $|S_t|$, we obtain the desired bound

$$|S_{t+1}| \geq \frac{N}{(t + 1)!q^{t+1}} - (t + 1).$$

This shows that we can construct the sequence p_1, \dots, p_{t+1} and the set S_{t+1} with the desired properties. For $N = 2^{8qn^2 \lceil \log(qn) \rceil}$, we have

$$|S_{qn^2}| \geq \frac{2^{8qn^2 \log(qn)}}{(qn^2)!q^{qn^2}} - qn^2 \geq 1. \tag{4}$$

Hence, $P_1 = \{p_1, \dots, p_{qn^2}\}$ is well defined. Since χ' is a q -coloring on P_1 , by the pigeonhole principle, there exists a subset $P_2 \subset P_1$ such that $|P_2| = n^2$, and every vertex has the same color. By construction of P_2 , every pair in P_2 has the same color. Hence these vertices induce a monochromatic geometric graph.

Now let $P_2 = \{p'_1, \dots, p'_{n^2}\}$. We define partial orders \prec_1, \prec_2 on P_2 , where $p'_i \prec_1 p'_j$ ($p'_i \prec_2 p'_j$) if and only if $i < j$ and the set of points $P_2 \setminus \{p'_1, \dots, p'_j\}$ lies above (below) the line going through points p'_i and p'_j . See Figure 3. By construction of P_2 , \prec_1, \prec_2 are indeed partial orders and every two elements in P_2 are comparable by either \prec_1 or \prec_2 . By Dilworth's theorem [8] (see also Theorem 1.1 in [14]), there exists a chain p^*_1, \dots, p^*_n of length n with respect to one of the partial orders. Hence (p^*_1, \dots, p^*_n) forms either an n -cap or an n -cup. Therefore, these vertices induce a complete monochromatic convex geometric graph.

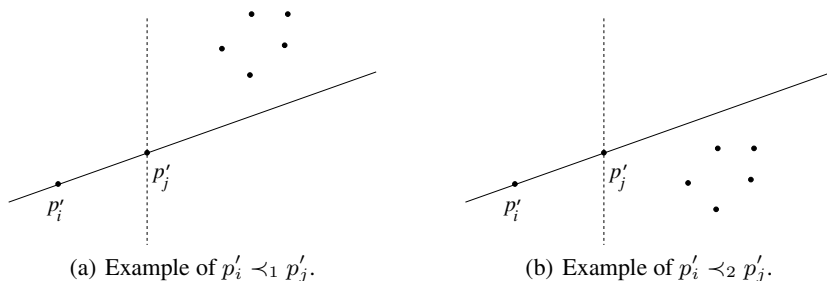


Fig. 3. Partial orders \prec_1, \prec_2

For the lower bound, we proceed by induction on q . The base case $q = 1$ follows by taking the complete geometric graph on 2^{n-1} vertices, whose vertex set does not have n members in convex position. This is possible by the construction of Erdős and Szekeres [13]. Let G_0 denote this geometric graph. For $q > 1$, we inductively construct a complete geometric graph $G = (V, E)$ on $2^{(q-1)(n-1)}$ vertices, and a coloring $\chi : E \rightarrow \{1, 2, \dots, q-1\}$ on the edges of G such that G does not contain a monochromatic convex geometric graph on n vertices. Now we replace each vertex $v_i \in G$ with a very small copy² of G_0 , which we will denote as G_i , where all edges in G_i are colored with the color q , and all edges between G_i and G_j have color $\chi(v_i v_j)$. Then we have a complete geometric graph G' on

$$2^{(q-1)(n-1)} 2^{n-1} = 2^{q(n-1)}$$

vertices, such that G' does not contain a monochromatic convex graph on n vertices. \square

By following the proof above, one can show that $g(K_n^\ell; q) \leq \text{twr}_\ell(O(n^2))$. However, the following short argument due to David Conlon gives a better bound. The proof uses an old idea of Tarsi (see [21] Chapter 3) that yields an upper bound on $f(n)$.

Lemma 1. *For geometric 3-hypergraphs, we have $g(K_n^3; q) \leq r(K_n^3; 2q) \leq 2^{2^{cn}}$, where $c = O(q \log q)$.*

² Obtained by an affine transformation.

Proof. Let $H = (V, E)$ be a complete geometric 3-hypergraph on $N = r(K_n^3; 2q)$ vertices, and let χ be a q coloring on the edges of H . By fixing an ordering on the vertices $V = \{v_1, \dots, v_N\}$, we say that a triple $(v_{i_1}, v_{i_2}, v_{i_3}), i_1 < i_2 < i_3$, has a *clockwise (counterclockwise) orientation*, if $v_{i_1}, v_{i_2}, v_{i_3}$ appear in clockwise (counterclockwise) order along the boundary of $\text{conv}(v_{i_1} \cup v_{i_2} \cup v_{i_3})$. Hence by Ramsey's theorem, there are n points from V for which every triple has the same color and the same orientation. As observed by Tarsi (see Theorem 3.8 in [25]), these vertices must be in convex position. \square

Lemma 2. For $\ell \geq 4$ and $n \geq 4^\ell$, we have $g(K_n^\ell; q) \leq r(K_n^\ell; q + 1) \leq \text{twr}_\ell(cn)$, where $c = O(q \log q)$.

Proof. Let $H = (V, E)$ be a complete geometric ℓ -hypergraph on $N = r(K_n^\ell; q + 1)$ vertices, and let χ be a q coloring on the ℓ -tuples of V with colors $1, 2, \dots, q$. Now if an ℓ -tuple from V is not in convex position, we change its color to the new color $q + 1$. By Ramsey's theorem, there is a set $S \subset V$ of n points for which every ℓ -tuple has the same color. Since $n \geq 4^\ell$, by the Erdős-Szekeres Theorem, S contains ℓ members in convex position. Hence, every ℓ -tuple in S is in convex position, and has the same color which is not the new color $q + 1$. Therefore S induces a monochromatic convex geometric ℓ -hypergraph. \square

Theorem 2 now follows by combining Lemmas 1 and 2.

3 A Lower Bound Construction for Geometric 3-hypergraphs

In this section, we will prove Theorem 3, which follows immediately from the following lemma.

Lemma 3. For sufficiently large n , there exists a complete geometric 3-hypergraph $H = (V, E)$ in the plane with $2^{2^{n/2}}$ vertices, and a two-coloring χ' on the edge set E , such that H does not contain a monochromatic convex 3-hypergraph on $2n$ vertices.

Proof. Let G be the complete graph on $2^{n/2}$ vertices, where $V(G) = \{1, \dots, 2^{n/2}\}$, and let χ be a red-blue coloring on the edges of G such that G does not contain a monochromatic complete subgraph on n vertices. Such a graph does indeed exist by a result of Erdős [9], who showed that $r(K_n, K_n) > 2^{n/2}$. We will use G and χ to construct a complete geometric 3-hypergraph H on $2^{2^{n/2}}$ vertices, and a coloring χ' on the edges of H , with the desired properties.

Set $M = 2^{n/2}$. We will recursively construct a point set P_t of 2^t points in the plane as follows. Let P_1 be a set of two points in the plane with distinct x -coordinates. After obtaining the point set P_t , we define P_{t+1} as follows. We inductively construct two copies of P_t , $L = P_t$ and $R = P_t$, and place L to the left of R such that all lines determined by pairs of points in L go below R and all lines determined by pairs of points of R go above L . Then we set $P_{t+1} = L \cup R$. See Figure 4.

Let $P_M = \{p_1, \dots, p_{2^M}\}$ be the set of 2^M points in the plane, ordered by increasing x -coordinate, from our construction. Note that P_M contains 2^{M-t} disjoint copies of P_t . For $i < j$, we define

$$\delta(p_i, p_j) = \max\{t : p_i, p_j \text{ lies inside a copy of } P_t = L \cup R, \text{ and } p_i \in L, p_j \in R\}.$$

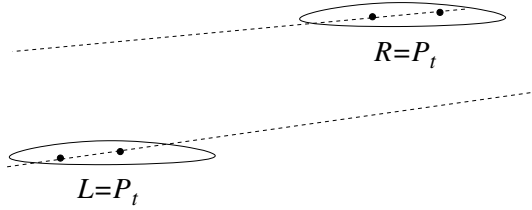


Fig. 4. Constructing P_{t+1} from P_t

Note that

- Property A:** $\delta(p_i, p_j) \neq \delta(p_j, p_k)$ for every triple $i < j < k$,
- Property B:** for $i_1 < \dots < i_n$, $\delta(p_{i_1}, p_{i_n}) = \max_{1 \leq j \leq n-1} \delta(p_{i_j}, p_{i_{j+1}})$.

Now we define a red-blue coloring χ' on the triples of P_M as follows. For $i < j < k$,

$$\chi'(p_i, p_j, p_k) = \chi(\delta(p_i, p_j), \delta(p_j, p_k)).$$

Now we claim that the geometric 3-hypergraph $H = (P_M, E)$ does not contain a monochromatic convex 3-hypergraph on $2n$ vertices. For sake of contradiction, let $S = \{q_1, \dots, q_{2n}\}$ be a set of $2n$ points from P_M , ordered by increasing x -coordinate, that induces a red convex 3-hypergraph. Set $\delta_i = \delta(q_i, q_{i+1})$.

Case 1. Suppose that there exists a j such that $\delta_j, \delta_{j+1}, \dots, \delta_{j+n-1}$ forms a monotone sequence. First assume that

$$\delta_j > \delta_{j+1} > \dots > \delta_{j+n-1}.$$

Since G does not contain a red complete subgraph on n vertices, there exists a pair $j \leq i_1 < i_2 \leq j + n - 1$ such that $(\delta_{i_1}, \delta_{i_2})$ is blue. But then the triple $(q_{i_1}, q_{i_2}, q_{i_2+1})$ is blue. Indeed, by Property B,

$$\delta(q_{i_1}, q_{i_2}) = \delta(q_{i_1}, q_{i_1+1}) = \delta_{i_1}.$$

Therefore, since $\delta_{i_1} > \delta_{i_2}$ and $(\delta_{i_1}, \delta_{i_2})$ is blue, the triple $(q_{i_1}, q_{i_2}, q_{i_2+1})$ must also be blue which is a contradiction. A similar argument holds if $\delta_j < \delta_{j+1} < \dots < \delta_{j+n-1}$.

Case 2. Suppose we are not in Case 1. For $2 \leq i \leq 2n$, we say that i is a *local minimum* if $\delta_{i-1} > \delta_i < \delta_{i+1}$, a *local maximum* if $\delta_{i-1} < \delta_i > \delta_{i+1}$, and a *local extremum* if it is either a local minimum or a local maximum. This is well defined by Property A.

Observation 4. For $2 \leq i \leq 2n$, i is never a local minimum.

Proof. Suppose $\delta_{i-1} > \delta_i < \delta_{i+1}$ for some i , and suppose that $\delta_{i-1} \geq \delta_{i+1}$. We claim that $q_{i+1} \in \text{conv}(q_{i-1}, q_i, q_{i+2})$. Indeed, since $\delta_{i-1} \geq \delta_{i+1} > \delta_i$, this implies that $q_{i-1}, q_i, q_{i+1}, q_{i+2}$ lies inside a copy of $P_{\delta_{i-1}} = L \cup R$, where $q_{i-1} \in L$ and $q_i, q_{i+1}, q_{i+2} \in R$. Since $\delta_{i+1} > \delta_i$, this implies that q_i, q_{i+1}, q_{i+2} lie inside a copy $P_{\delta_{i+1}} = L' \cup R' \subset R$, where $q_i, q_{i+1} \in L'$ and $q_{i+2} \in R'$.

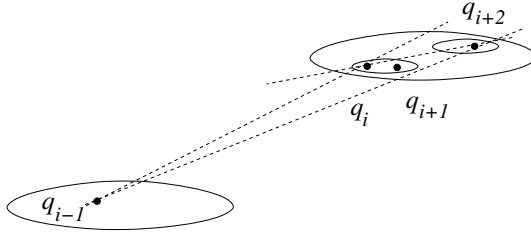


Fig. 5. Point $q_{i+1} \in \text{conv}(q_{i-1}, q_i, q_{i+2})$

Notice that all lines determined by q_i, q_{i+1}, q_{i+2} go above the point q_{i-1} . Therefore q_{i+1} must lie above the line that goes through the points q_{i-1}, q_{i+2} , and furthermore, q_{i+1} must lie below the line that goes through the points q_{i-1}, q_i . Since $\delta_{i+1} > \delta_i$, the line through q_i, q_{i+1} must go below the point q_{i+2} , and therefore $q_{i+1} \in \text{conv}(q_{i-1}, q_i, q_{i+2})$. See Figure 5. If $\delta_{i-1} < \delta_{i+1}$, then a similar argument shows that $q_i \in \text{conv}(q_{i-1}, q_{i+1}, q_{i+2})$. □

Since $\delta_1, \dots, \delta_{2n}$ does not have a monotone subsequence of length n , it must have at least two local extrema. Since between any two local maximums there must be a local minimum, we have a contradiction by Observation 4. This completes the proof. □

4 Concluding Remarks

For $q \geq 4$ colors and $\ell \geq 2$, we showed that $g(K_n^\ell; q) = \text{twr}_\ell(\Theta(n))$. Our bounds on $g(K_n^\ell; q)$ for $q \leq 3$ can be summarized in the following table.

	$q = 2$	$q = 3$
$\ell = 2$	$2^{\Omega(n)} < g(K_n, K_n) \leq 2^{O(n^2 \log n)}$	$2^{\Omega(n)} < g(K_n; 3) \leq 2^{O(n^2 \log n)}$
$\ell = 3$	$g(K_n^3, K_n^3) = 2^{2^{\Theta(n)}}$	$g(K_n^3; 3) = 2^{2^{\Theta(n)}}$
$\ell \geq 4$	$g(K_n^\ell, K_n^\ell) \geq \text{twr}_{\ell-1}(\Omega(n^2))$ $g(K_n^\ell, K_n^\ell) \leq \text{twr}_\ell(O(n))$	$g(K_n^\ell; 3) \geq \text{twr}_\ell(\Omega(\log^2 n))$ $g(K_n^\ell; 3) \leq \text{twr}_\ell(O(n))$

Off-diagonal. The Ramsey number $r(K_s, K_n)$ is the minimum integer N such that every red-blue coloring on the edges of a complete N -vertex graph G , contains either

a red clique of size s , or a blue clique of size n . The off-diagonal Ramsey numbers, $r(K_s, K_n)$ with s fixed and n tending to infinity, have been intensively studied. For example, it is known [1,4,5,20] that $R_2(3, n) = \Theta(n^2/\log n)$ and, for fixed $s > 3$,

$$c_1(\log n)^{1/(s-2)} \left(\frac{n}{\log n}\right)^{(s+1)/2} \leq r(K_s, K_n) \leq c_2 \frac{n^{s-1}}{\log^{s-2} n}. \tag{5}$$

Another interesting variant of Problem 3 is the following off-diagonal version.

Problem 4. Determine the minimum integer $g(K_s, K_n)$, such that any red-blue coloring on the edges of a complete geometric graph G on $g(K_s, K_n)$ vertices, yields either a red convex geometric graph on s vertices, or a blue convex geometric graph on n vertices.

For fixed s , one can show that $g(K_s, K_n)$ grows single exponentially in n . In particular

$$2^{n-1} + 1 \leq g(K_s, K_n) \leq 4^{4^s n}.$$

The lower bound follows from the fact that $g(K_s, K_n) \geq f(n)$. The upper bound follows from the inequalities

$$g(K_s, K_n) \leq r(K_{4^s}, K_{4^n}) \leq (4^n)^{4^s}.$$

Indeed if G contains a red-clique of size 4^s , then by the Erdos-Szkeres Theorem there must be a red convex geometric graph on s vertices. Likewise, If G contains a blue clique of size 4^n , then there must be a blue convex geometric graph on n vertices.

Higher Dimensions. Generalizing Problem 1 to higher dimensions has also been studied. Let $f_d(n)$ be the smallest integer such that any set of at least $f_d(n)$ points in \mathbb{R}^d in general position³ contains n members in convex position. The following upper and lower bounds were obtained by Károlyi [16] and Károlyi and Valtr [19] respectively,

$$2^{cn^{1/(d-1)}} \leq f_d(n) \leq \binom{2n - 2d - 1}{n - d} + d = 2^{2n(1+o(1))}.$$

A *geometric ℓ -hypergraph H in d -space* is a pair (V, E) , where V is a set of points in general position in \mathbb{R}^d , and $E \subset \binom{V}{\ell}$ is a collection of ℓ -tuples from V . When $\ell \leq d+1$, ℓ -tuples are represented by $(\ell - 1)$ -dimensional simplices induced by the corresponding vertices.

Problem 5. Determine the minimum integer $g_d(K_n^\ell; q)$, such that any q -coloring on the edges of a complete geometric ℓ -hypergraph H in d -space on $g_d(K_n^\ell; q)$ vertices, yields a monochromatic complete convex ℓ -hypergraph on n vertices.

When $d = 2$, we write $g_2(K_n^\ell; q) = g(K_n^\ell; q)$. Clearly

$$g_d(K_n^\ell; q) \geq \max\{f_d(n), R(K_n^\ell; q)\}.$$

³ A point set P in \mathbb{R}^d is in general position, if no $d + 1$ members lie on a common hyperplane.

One can also show that $g_d(K_n^\ell; q) \leq g(K_n^\ell; q)$. Indeed, for any complete geometric ℓ -hypergraph $H = (V, E)$ in d -space with a q -coloring χ on $E(H)$, one can obtain a complete geometric ℓ -hypergraph in the plane $H' = (V', E')$, by projecting H onto a 2-dimensional subspace $L \subset \mathbb{R}^d$ such that V' is in general position in L . Thus we have

$$g_d(K_n; q) \leq g(K_n; q) \leq 2^{cn^2 \log n},$$

where $c = O(q \log q)$, and for $\ell \geq 3$

$$g_d(K_n^\ell; q) \leq g(K_n^\ell; q) \leq \text{twr}_\ell(c'n^2),$$

where $c' = c'(q, \ell)$.

Acknowledgment. We thank David Conlon for showing us an improved version of Theorem 2, shown in Section 2.

References

1. Ajtai, M., Komlós, J., Szemerédi, E.: A note on Ramsey numbers. *J. Combin. Theory Ser. A* 29, 354–360 (1980)
2. Ábrego, B.M., Fernández-Merchant, S., Salazar, G.: The rectilinear crossing number of K_n : Closing in (or are we?), Thirty Essays on Geometric Graph Theory. In: Pach, J. (ed.) *Algorithms and Combinatorics*, vol. 29, pp. 5–18. Springer (2012)
3. Aronov, B., Erdős, P., Goddard, W., Kleitman, D.J., Klugerman, M., Pach, J., Schulman, L.J.: Crossing families. *Combinatorica* 14, 127–134 (1994)
4. Bohman, T.: The triangle-free process. *Adv. Math.* 221, 1653–1677 (2009)
5. Bohman, T., Keevash, P.: The early evolution of the H -free process. *Invent. Math.* 181, 291–336 (2010)
6. Conlon, D., Fox, J., Sudakov, B.: *Journal of the American Mathematical Society* 23, 247–266 (2010)
7. Dey, T., Pach, J.: Extremal problems for geometric hypergraphs. *Discrete Comput. Geom.* 19, 473–484 (1998)
8. Dilworth, R.P.: A decomposition theorem for partially ordered sets. *Ann. of Math.* 51, 161–166 (1950)
9. Erdős, P.: Some remarks on the theory of graphs. *Bull. Amer. Math. Soc.* 53, 292–294 (1947)
10. Erdős, P., Hajnal, A., Rado, R.: Partition relations for cardinal numbers. *Acta Math. Acad. Sci. Hungar.* 16, 93–196 (1965)
11. Erdős, P., Rado, R.: Combinatorial theorems on classifications of subsets of a given set. *Proc. London Math. Soc.* 3, 417–439 (1952)
12. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. *Compos. Math.* 2, 463–470 (1935)
13. Erdős, P., Szekeres, G.: On some extremum problems in elementary geometry. *Ann. Univ. Sci. Budapest. Eötvös Sect. Math.* 3(4), 53–62 (1960-1961)
14. Fox, J., Pach, J., Sudakov, B., Suk, A.: Erdős-Szekeres-type theorems for monotone paths and convex bodies. *Proceedings of the London Mathematical Society* 105, 953–982 (2012)
15. Graham, R.L., Rothschild, B.L., Spencer, J.H.: *Ramsey Theory*, 2nd edn. Wiley, New York (1990)
16. Károlyi, G.: Ramsey-remainder for convex sets and the Erdős-Szekeres theorem. *Discrete Appl. Math.* 109, 163–175 (2001)

17. Károlyi, G., Pach, J., Tóth, G.: Ramsey-type results for geometric graphs, I. *Disc. Comp. Geom.* 18, 247–255 (1997)
18. Károlyi, G., Pach, J., Tóth, G., Valtr, P.: Ramsey-type results for geometric graphs, II. *Disc. Comp. Geom.* 20, 375–388 (1998)
19. Károlyi, G., Valtr, P.: Point configurations in d -space without large subsets in convex position. *Disc. Comp. Geom.* 30, 277–286 (2003)
20. Kim, J.H.: The Ramsey number $R(3, t)$ has order of magnitude $t^2 / \log t$. *Random Structures Algorithms* 7, 173–207 (1995)
21. Matoušek, J.: *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc. (2002)
22. Pach, J., Agarwal, P.: *Combinatorial geometry*. Wiley-Interscience, New York (1995)
23. Ramsey, F.P.: On a problem in formal logic. *Proc. London Math. Soc.* 30, 264–286 (1930)
24. Suk, A.: A note on geometric 3-hypergraphs, *Thirty Essays on Geometric Graph Theory*. In: Pach, J. (ed.) *Algorithms and Combinatorics*, vol. 29, pp. 489–498. Springer (2012)
25. Van Lint, J.H., Wilson, R.M.: *A Course in Combinatorics*. Cambridge University Press (2001)
26. Wagner, U.: k -sets and k -facets, *Discrete and Computational Geometry - 20 Years Later*. In: Goodman, E., Pach, J., Pollack, R. (eds.) *Contemporary Mathematics*, vol. 453, pp. 443–514. American Mathematical Society (2008)

Minimum Length Embedding of Planar Graphs at Fixed Vertex Locations

Timothy M. Chan, Hella-Franziska Hoffmann,
Stephen Kiazzyk, and Anna Lubiw

David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, ON, Canada
{tmchan,hrhoffmann,skiazzyk,alubiw}@uwaterloo.ca

Abstract. We consider the problem of finding a planar embedding of a graph at fixed vertex locations that minimizes the total edge length. The problem is known to be NP-hard. We give polynomial time algorithms achieving an $O(\sqrt{n} \log n)$ approximation for paths and matchings, and an $O(n)$ approximation for general graphs.

1 Introduction

Suppose we want to draw a planar graph and the vertex locations are specified. Such a planar drawing always exists, although not necessarily with straight line edges. Pach and Wenger [1] showed how to construct a drawing using $O(n)$ bends on each edge, where n is the number of vertices. We consider an equally natural optimization criterion—to minimize the total edge length.

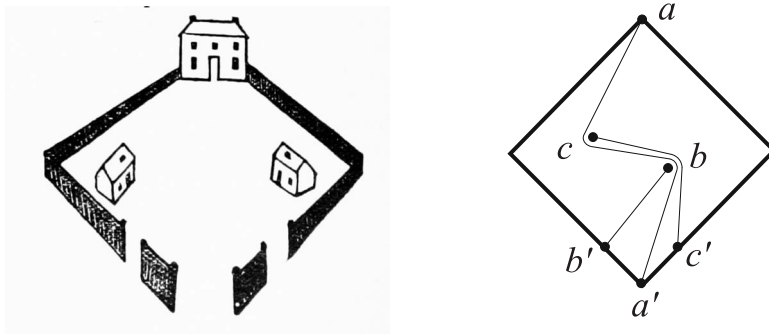


Fig. 1. A puzzle from Loyd [2]—connect each house to the opposite gate with non-crossing paths. On the left is the minimum length solution to an asymmetric version.

For example, Figure 1 shows a puzzle disseminated by Sam Loyd [2, p. 27]. The goal is to connect each house with the gate opposite its door via non-crossing paths. There are two distinct solutions but if the points are shifted as shown on the right in Figure 1, there is a unique shortest solution.

In this example the fixed outer wall plays a significant role, so the example demonstrates a more general problem—to extend a planar drawing of a subgraph to a planar drawing of the whole graph minimizing the total length. Fixed edges in the drawing act as *obstacles*. We call this *Minimum Length Planar Drawing of Partially Embedded Graphs*.

Angelini et al. [3] gave a linear time algorithm to decide if a planar drawing of a subgraph can be extended to a planar drawing of the whole graph. Our problem is the optimization version, to minimize the total edge length.

We will restrict attention to the case where all vertex positions are fixed. Furthermore, most of our results are for the case when none of the edges are fixed. We call this *Minimum Length Planar Drawing [or Embedding] at Fixed Vertex Locations*. This problem is very interesting even for special graphs such as matchings and paths.

When the edges to be added form a matching, the problem is to join specified pairs of points via non-crossing paths of minimum total length. The case when there are no obstacles was considered by Liebling et al. [4] in 1995. They gave some heuristics based on finding a short non-crossing tour of the points and then “wrapping” the matching edges around the tour. We use this same technique for our approximation results. They also proved that for points in the unit square a shortest non-crossing matching has length $O(n\sqrt{n})$ and there are examples realizing this bound. The lower bound (due to Peter Shor) relies on the existence of expander graphs with large crossing number. In 1996 Bastert and Fekete [5] proved that the problem is NP-hard, even with no obstacles. There is also substantial work on the case where there are obstacles (i.e. when some edges are fixed)—specifically when the points lie on the boundary of a polygon [6], or multiple polygons [7] (in which case the run time is exponential in the number of polygons). We give more details below in Section 1.1, and also discuss related work on finding “thick” paths that are separated from each other.

The problem of Minimum Length Planar Embedding at Fixed Vertex Locations is also interesting when the graph we want to embed is a path. This version of the problem was formulated by Polishchuk and Mitchell [8]. Their main goal was to find a minimum length tour that visits a given sequence of convex bodies in \mathbb{R}^d (see [9] for the planar case), without regard to whether the path is self-intersecting, but in their conclusion section they ask about finding a minimum length non-crossing tour for a sequence of points.

Our Contributions. We give polynomial time approximation algorithms for Minimum Length Planar Embedding at Fixed Vertex Locations. In the case of general planar graphs we achieve an approximation ratio of $O(n)$. In the case of a matching or a path we achieve an approximation ratio of $O(\sqrt{n} \log n)$. Our main technique is to route graph edges around a carefully chosen path or tree defined on the input points in the plane.

1.1 Related Work

Bastert and Fekete [5] prove that Minimum Length Planar Drawing at Fixed Vertex Locations is NP-hard when the graph is a matching.

Patrignani [10] proved that it is NP-hard to decide if a planar drawing of a subgraph can be extended to a planar straight-line drawing of the whole graph.

Papadopoulou [6] gave an efficient algorithm for finding minimum length non-crossing paths joining pairs of points on the boundary of a polygon. In this case each path is a shortest path, but Papadopoulou finds them more efficiently than the obvious approach. Erickson and Nayyeri [7] extended this to points on the boundaries of h polygonal obstacles. Their algorithm has a running time that is linear for fixed h , but grows exponentially in h .

The difficulty with multiple polygons is deciding which homotopy of the paths gives minimum length. If the homotopy is specified the problem is easy [11,12].

In the aforementioned results on shortest non-crossing paths, one issue is that paths will overlap in general even though crossings are forbidden (see Figure 1 for an example). In practical applications we often need paths that are disjoint and maintain some minimum separation from each other. This issue is addressed in papers about drawing graphs with “thick” edges. Duncan et al. [13] show how to find thick shortest homotopic paths. Polishcuk and Mitchell [14] show how to find shortest thick disjoint paths joining endpoints on the boundaries of polygonal obstacles (with exponential dependence on the number of obstacles). They also show hardness results, including hardness of approximation.

In our problem the correspondence between the vertices and the fixed points in the plane is given. There is a substantial body of work where the correspondence of vertices to points is not fixed. Cabello [15] showed that it is NP-hard to decide if there is a correspondence that allows a straight-line planar drawing. Many special cases have been classified as polynomial time or NP-complete. A related problem is to find small *universal point sets* on which all planar graphs can be straight-line embedded (see [16]).

A problem related to minimum length planar embedding at fixed vertex locations is to draw planar graphs so that each edge is a monotone path of axis-parallel line segments. Any such path is a shortest path in the L_1 or Manhattan metric, and these drawings are called Manhattan-geodesic embeddings. This concept was introduced by Katz et al. [17]. They considered the case where the graph is a matching and showed that the problem is NP-hard when the drawing is restricted to a grid, but solvable in polynomial time otherwise.

We restricted the general problem of extending a partial planar embedding to the case where all vertex positions are fixed. The case where some vertices are free to move is also very interesting and is related to work on Steiner trees with fixed tree topology [18] and Steiner trees with obstacles [19]. Finally, one may consider drawing graphs at fixed vertex locations but allowing edges to cross. This is interesting and non-trivial when crossings must have large angles [20].

For other geometric graph augmentation problems see the survey by Hurtado and Tóth [21].

1.2 Definitions and Basic Observations

We consider the following problem called *Minimum Length Planar Embedding at Fixed Vertex Locations*: Given a planar graph $G = (\{v_1, \dots, v_n\}, E)$ and a

set of points $P = \{p_1, \dots, p_n\}$, find a planar embedding of G in the plane that places vertex v_i at point p_i and minimizes the total edge length.

Edges of the embedding are allowed to overlap, but they must be non-crossing (i.e. infinitesimally deformable into disjoint paths). In the following we will refer to the vertices and their respective fixed locations interchangeably. The Euclidean distance between two points $p, q \in \mathbb{R}^2$ is denoted $d(p, q)$.

Observation 1. $L = \sum_{(v_i, v_j) \in E} d(p_i, p_j)$ is a trivial lower bound for the total length of any planar embedding of graph G at fixed vertex locations P .

One approach for finding short embeddings at fixed vertex locations is to draw each edge (v_i, v_j) as a curve whose length is within a constant factor of distance $d(p_i, p_j)$. Unfortunately such a planar drawing does not always exist; see Fig. 2.

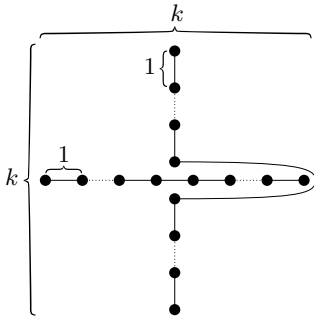


Fig. 2. Example for which any solution contains at least one edge of length greater than $k = n/2 - 1$

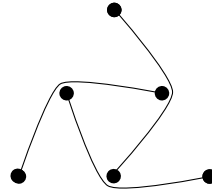


Fig. 3. Example for which any optimal solution contains no straight line edges

The example in Fig. 3 shows an instance where no straight line edge is included in any optimal solution, which means that obvious greedy algorithms for the problem fail. This was first observed by Liebling et al. [4].

Note that any edge of an optimal embedding bends only at vertex locations. Thus any optimal embedding lives in some underlying triangulation of the point set. Given the triangulation, the problem becomes that of finding short non-crossing paths in a planar graph. This problem was first proposed by Takahashi et al. [22] who considered the case of terminal points on two faces. Erickson and Nayyeri [7] say that the general problem is NP-hard, citing Bastert and Fekete [5]. Unfortunately, we cannot find this result in the version of the report that we have.

Instead of fixing the underlying triangulation, we use a carefully chosen path or tree as the layout for our embeddings.

2 Embedding a Path or Matching

In this section we give polynomial time approximation algorithms for the case where G is a path or a matching. Our starting point is a 1-dimensional version

of the problem that will be the basis for all further results. We give a polynomial time (exact) algorithm for the case where G is a path, and the points lie on a line. We note that Liebling et al. [4] apparently knew the analogous result for the case when G is a matching, since they say that the edges of a matching can be “wrapped around” any non-self-intersecting tour of the points. They give no details of how to do the wrapping, and we consider the details worth explaining.

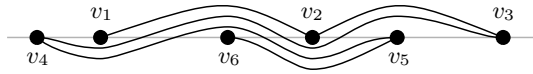


Fig. 4. A minimum length embedding of a path on fixed points that lie on a line. Edges are drawn with gaps between them for clarity only.

Lemma 1. *There is a polynomial time algorithm to find a minimum length embedding of a path on fixed vertex positions that lie on a line.*

Proof. Without loss of generality, assume that all the points lie on a horizontal line. See Fig. 4. This allows us to speak of “above” and “below”. We draw the edges in order along the path. Draw edge (v_i, v_{i+1}) as a curve from point p_i to point p_{i+1} that stays below all edges drawn so far, but stays above all later points $p_j, j > i + 1$. This ensures that later edges of the path can reach their endpoints without crossing earlier edges. \square

As noted by Liebling et al., this idea of “weaving” the edges through the points can be extended to the the case where the points lie on a simple (i.e. non-self-intersecting) curve in the plane. (In fact, the idea even extends to a tree, as we shall see in Section 3.) If one can find a simple curve C passing through every point in P , then “weaving” the edges of the path along the curve creates a path of length at most

$$\sum_{i=1}^{n-1} d_C(p_i, p_{i+1}), \tag{1}$$

where $d_C(p_i, p_{i+1})$ denotes the length of the subcurve of C from p_i to p_{i+1} . This sum is trivially upper-bounded by n times the length of C .

We can choose the curve C to be an $O(1)$ -approximation to the minimum-length Hamiltonian path (i.e., the traveling salesman path) for P (e.g., the simplest option would be the standard 2-approximate solution obtained from the minimum spanning tree). Since the length of the traveling salesman path is a lower bound for the problem in the path case, this would give an $O(n)$ -approximate solution overall.

In Section 3 we will extend this idea to obtain an $O(n)$ approximation for general planar graphs. In the remainder of the current section we show how to improve the approximation factor for a path or matching by choosing a better curve C . The property we need is that points that are close in the plane are

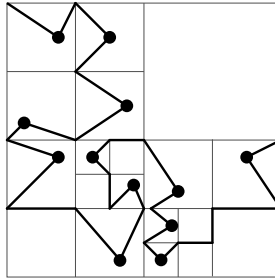


Fig. 5. The curve that is used as routing layout for embedding paths and matchings

close on the curve. The idea is to use a construction based on *shifted quadtrees*, similar to certain well known families of space-filling curves such as the *Z-order curve* or the *Hilbert curve*.

Let D_0 be the diameter of P . Without loss of generality, assume that $P \subset [0, D_0]^2$. We initially shift all the points in P by a random vector $v \in [0, D_0]^2$. Now, $P \subset [0, 2D_0]^2$.

Given a square S , the following procedure returns a simple polygonal curve, with the property that the curve starts at one corner of S , ends at another corner, and stays inside S while visiting all points of $P \cap S$.

CURVE(S):

1. if $|P \cap S| \leq 1$ then
2. return a curve with the stated property, using at most 2 line segments
3. divide S into 4 subsquares S_1, \dots, S_4
4. for $i = 1, \dots, 4$, compute $C_i = \text{CURVE}(S_i)$
5. return a curve with the stated property by joining C_1, \dots, C_4 , using $O(1)$ connecting line segments

Slight perturbation may be needed in line 5 to ensure that we obtain a simple curve. There is flexibility as to which corner we choose to start or end. (If we always start at the upper left corner and end at the lower right corner, the construction is similar to the Z-order curve. If we choose starting and ending corners to be adjacent in a manner similar to the Hilbert curve instead, the connecting line segments in line 5 may even be avoided; see Fig. 5. The main difference with standard space-filling curves is that we terminate the recursion as soon as we reach a square containing zero or one point.)

Lemma 2. *The length of the curve returned by CURVE(S) is at most $O(D_S \sqrt{n_S})$, where $n_S = |P \cap S|$ and D_S is the side length of S .*

Proof. Let L_i be the sum of the lengths of all line segments generated in line 2 or 5 at the i -th level of the recursion. The squares at the i -th level have side length $D_S/2^i$, and the number of squares at the i -th level is upper-bounded by both 4^i and n_S . Thus,

$$L_i \leq O(D_S/2^i) \cdot \min\{4^i, n_S\} = O(\min\{D_S 2^i, D_S n_S/2^i\}).$$

The total length of the curve is then at most

$$\sum_{i=0}^{\infty} L_i \leq \sum_{i=0}^{\lceil (1/2) \log n_S \rceil - 1} D_S 2^i + \sum_{i=\lceil (1/2) \log n_S \rceil}^{\infty} D_S n_S / 2^i = O(D_S \sqrt{n_S}). \quad \square$$

We now let C be the curve returned by $\text{CURVE}(S_0)$ with $S_0 = [0, 2D_0]^2$. All the squares generated by the recursive calls are *quadtrees squares*.

Define $D(p, q)$ to be the side length of the smallest quadtree square enclosing p and q . Note that $D(p, q) \geq d(p, q)/\sqrt{2}$. It is known that after random shifting, $D(p, q)$ approximates $d(p, q)$ to within a logarithmic factor in expectation (e.g., see [23, Lemma 5.1]). We include a quick proof for the sake of completeness.

Lemma 3. *For a fixed pair of points $p, q \in P$,*

$$\mathbf{E}[D(p, q)] \leq O(\log(D_0/d(p, q))) \cdot d(p, q).$$

Proof. $D(p, q) > D_0/2^i$ if and only if \overline{pq} crosses a horizontal or vertical grid line in the grid formed by the quadtree squares of side length $D_0/2^i$. The probability that this happens is $O\left(\frac{d(p, q)}{D_0/2^i}\right)$. Thus,

$$\mathbf{E}[D(p, q)] \leq O\left(\sum_{i=0}^{\lceil \log(D_0/d(p, q)) \rceil} \frac{d(p, q)}{D_0/2^i} \cdot D_0/2^i\right) \leq O(\log(D_0/d(p, q))) \cdot d(p, q).$$

□

Lemma 4. *For a fixed pair of points $p, q \in P$,*

$$\mathbf{E}[d_C(p, q)] \leq O(\sqrt{n} \log(D_0/d(p, q))) \cdot d(p, q).$$

Proof. The portion of the curve C from p to q lies inside a quadtree square with side length $D(p, q)$. Thus, by Lemma 2, $d_C(p, q)$ is at most $O(D(p, q)\sqrt{n})$. The conclusion then follows from Lemma 3. □

Theorem 2. *For a path G with fixed vertex locations, there is a polynomial-time randomized algorithm which computes a planar embedding of expected length at most $O(\sqrt{n} \log n) \cdot L$ where $L = \sum_{(p, q) \in E} d(p, q)$.*

Proof. By (1) and linearity of expectation, we obtain an embedding of expected length at most $(1 + \epsilon) \sum_{(p, q) \in E} \mathbf{E}[d_C(p, q)]$. By Lemma 4, this quantity is at most

$$\sum_{(p, q) \in E} O(\sqrt{n} \log(D_0/d(p, q))) \cdot d(p, q) \leq O(\sqrt{n} \log(nD_0/L)) \cdot L$$

because the logarithm function is concave. The theorem follows since $L \geq \Omega(D_0)$ for the case of a path G . □

Since L is a lower bound on the optimal cost, we obtain an $O(\sqrt{n} \log n)$ -approximation algorithm for the case of a path G . For the case of a matching, we obtain the same result with just slightly more effort:

Theorem 3. *For a matching G with fixed vertex locations, there is a polynomial-time randomized algorithm which computes a planar embedding of expected length at most $O(\sqrt{n} \log n) \cdot L$ where $L = \sum_{(p,q) \in E} d(p, q)$.*

Proof. We can use essentially the same algorithm. The cost of the solution is still bounded by (1) and the same analysis goes through, except that $L \geq \Omega(D_0)$ may no longer be true.

Observe that if there is a vertical line that separates the line segments $\{\overline{pq} : (p, q) \in E\}$ into two nonempty parts, then we can just recursively compute a planar embedding on both sides, since each embedding can be shrunk to lie within the minimum (axis-aligned) bounding box of its points. We may thus assume that no such vertical separating line exists, which implies that L is at least the width of the bounding box of P . Similarly, we may assume that no horizontal separating line exists, which implies that L is at least the height of the bounding box of P . These two assumptions imply $L \geq \Omega(D_0)$ and we may proceed as before. \square

Remarks. To obtain a time bound not sensitive to the bit complexity of the input, we can adopt a variant of the method where we compress long chains of degree-1 nodes in the tree (called the *compressed quadtree*), to ensure that the number of recursive calls is $O(n)$.

On the other hand, if input coordinate values are $O(\log n)$ bits long, we can derandomize the algorithm in polynomial time by trying all possible shifts.

The upper bound relative to L in Theorems 2 and 3 is tight up to a logarithmic factor: Liebling et al. [4] provide examples (due to Peter Shor) with points in the unit square for which any shortest non-crossing matching has length $\Omega(n\sqrt{n})$, proving a lower bound of $\Omega(\sqrt{n}) \cdot L$.

3 Embedding General Planar Graphs

In this section we give an $O(n)$ -approximation algorithm for constructing a planar embedding of a planar graph G at fixed vertex locations P .

The construction is based on the algorithm by Pach and Wenger [1] for finding a planar polygonal embedding of a graph with fixed vertex locations and with $O(n)$ bends per edge. Pach and Wenger draw the edges of the graph by tracing around a tree of n edges drawn in the plane. Each edge of the graph is drawn as a curve that walks around the tree a constant number of times, which gives the bound of $O(n)$ bends per edge. For their tree Pach and Wenger use a star with a leaf at each vertex.

In our case we want to bound the length of each edge, which can be done by bounding the length of the tree. We cannot use a star; instead, we will use a tree that is a subset of the (non-planar) drawing of G where each edge is drawn

as a straight line segment. This ensures that the tree has total length at most $L = \sum_{(p,q) \in E} d(p, q)$. Because of connectivity issues, we will actually use a set of disjoint trees:

Lemma 5. *Given a graph G , and fixed vertex locations P , we can construct in $O(n^2)$ time an embedded forest F with $O(n)$ vertices and total length at most L , such that for every edge (p, q) in G , p and q are in the same tree of F .*

Proof. We construct the forest F iteratively by adding edges of the graph one by one. For each edge $(p, q) \in E$ we will add some subsegments of the line segment pq to F . The forest will be a subset of an arrangement of $O(n)$ lines. The arrangement can be constructed in $O(n^2)$ time [24]. Consider edge (p, q) . If p and q are already in the same tree of F , we are done. Otherwise consider the line segment pq . It crosses at most n segments of F , and these crossing points subdivide it into $p = p_1, p_2, \dots, p_k = q$ with $k \leq n$. We treat these segments one by one in order. Consider segment $p_i p_{i+1}$. If p_i and p_{i+1} are already in the same tree of F , we are done. Otherwise we add segment $p_i p_{i+1}$ to F . Fig. 6 illustrates this idea. We use a union-find data structure to test if points are in the same tree of F . By construction, the length of F is bounded by L . Furthermore, we only add a segment when we join two trees of F , and this can happen at most n times. Thus F has $O(n)$ vertices and is a subset of an arrangement of n lines. \square

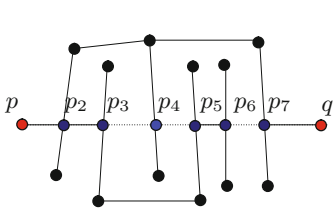


Fig. 6. Constructing the forest F in Lemma 5. Segment pq crosses multiple components of F . Segments $p_3 p_4$, $p_4 p_5$, and $p_6 p_7$ are not added to F .

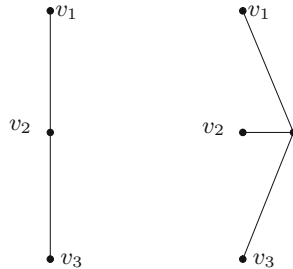


Fig. 7. Perturbing the tree to change v_2 from an internal vertex to a leaf.

Theorem 4. *Given a planar graph G with n vertices and fixed vertex locations P , there is an $O(n^2)$ -time approximation algorithm to construct a planar embedding of G on P with total length $O(n) \cdot L$ where $L = \sum_{(p,q) \in E} d(p, q)$.*

Proof. Use Lemma 5 to construct a forest F , that will serve as the basis for our edge routing. Because we do not want paths to travel through intermediate vertices, we perturb the trees in F slightly so that each vertex of G is a leaf of the tree that contains it. See Fig. 7. This can be done while keeping the trees disjoint and of total length $O(L)$.

Consider a single tree T of the forest F , together with the induced graph G_T on the vertices of G that lie in T . We will follow the approach of Pach and Wenger and draw the edges of G_T as paths hugging the tree T . Because every edge of G lies in some G_T , and the trees are disjoint (as objects in the plane), it suffices to describe the solution for a single tree T . To simplify notation, we will assume for the remainder of the proof that we have a single tree T and $G = G_T$.

We now follow Pach and Wenger’s solution, the main difference being that we have a more general tree than their star. We outline their solution and remark on the modifications required for our situation.

Pach and Wenger’s solution is based on a Hamiltonian cycle that they construct by adding vertices and edges to the graph. Specifically, they subdivide each edge of the graph by at most two new vertices and add some edges between vertices to obtain a planar graph with a Hamiltonian cycle [1, Lemma 5]. (Note that the new edges do not appear in the final drawing.) The Hamiltonian cycle C partitions edges of the planar graph relative to some (arbitrary) planar embedding into the edges *inside* C and the edges *outside* C . They first draw the edges of the Hamiltonian cycle C and then draw the inside and outside edges.

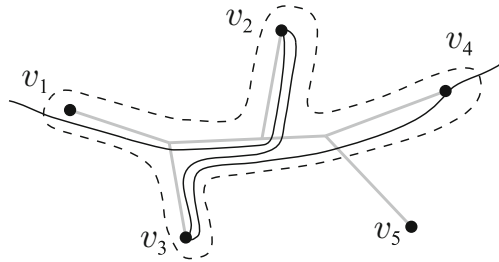


Fig. 8. Drawing the graph G around the tree T (drawn in gray) whose leaves are the graph vertices. The portion of the Hamiltonian cycle C from v_1 to v_4 is drawn as a solid curve. The dashed curve A_4 surrounds T_4 and is split by C into two paths between v_4 and v_1 , one inside C and one outside C .

To draw the edges of C they use an approach similar to the weaving technique described in Lemma 1. Renumber vertices so they appear in the order v_1, v_2, \dots, v_n along the Hamiltonian cycle. Some of these are new vertices that were added to create the Hamiltonian cycle. Pach and Wenger assign arbitrary locations to the new vertices, but we locate them very close to the tree T , adding them as leaves of T and keeping the length of T in $O(L)$. We will use v_i to refer to the vertex of G , the corresponding point in the plane, and the corresponding leaf of T . Edge (v_{i-1}, v_i) of C will be drawn around a subtree T_i of T . We define T_i more carefully for our situation: T_i is the connected subtree of T induced on leaves v_1, v_2, \dots, v_i . With these modifications, the rest of Pach and Wenger’s solution applies unaltered.

As they draw C they add (multiple copies of) auxiliary paths A_i from v_i to v_1 , one inside and one outside C . See Fig. 8. Then each edge (v_i, v_j) of G inside [outside] C is routed using the paths inside [outside] C from v_i to v_1 and from v_1 to v_j . For further details please refer to their paper [1]. The end result is a planar drawing of G on vertex locations P . Every original edge e of G has been subdivided by at most two new vertices, and each of the resulting three edges has been drawn as two paths in the tree. The total length of the drawing of e is therefore bounded by 6 times the length of T , and thus in $O(L)$.

Pach and Wenger's algorithm takes $O(n^2)$ time so our overall running time is $O(n^2)$ as well. \square

4 Conclusion and Open Problems

The problem of drawing a planar graph at fixed vertex locations while minimizing the total edge length seems to be very difficult although we are not aware of any hardness of approximation results. In fact, for the case of a path, even an NP-hardness result is lacking. Our algorithms achieve approximation factors of $O(n)$ for general graphs and $O(\sqrt{n} \log n)$ for paths and matchings. Besides the obvious question of improving these approximation factors (or proving hardness), we suggest looking at: (1) the problem of drawing a graph at fixed vertex locations with thick edges; and (2) looking at the case where some vertex locations are not fixed, which is related to drawing Steiner trees with fixed topology [18].

Acknowledgments. This work was done in the Algorithms Problem Session at Waterloo and we thank the other participants for good discussions. We learned about Sam Loyd's disjoint paths puzzle (which is not original to him) from Marcus Schaefer who has studied the history of such planarity puzzles.

References

1. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics* 17(4), 717–728 (2001)
2. Sam Loyd, J.: *Sam Loyd's Cyclopedia of 5000 Puzzles Tricks and Conundrums*. Lamb Publishing Company (1914)
3. Angelini, P., Di Battista, G., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 202–221 (2010)
4. Liebling, T.M., Margot, F., Müller, D., Prodon, A., Stauffer, L.: Disjoint paths in the plane. *ORSA Journal on Computing* 7(1), 84–88 (1995)
5. Bastert, O., Fekete, S.P.: *Geometric wire routing*. Technical Report 332, *Ange wandte Mathematik und Informatik, Universität zu Köln* (1996) (in German)
6. Papadopoulou, E.: k -pairs non-crossing shortest paths in a simple polygon. In: Nagamochi, H., Suri, S., Igarashi, Y., Miyano, S., Asano, T. (eds.) *ISAAC 1996*. LNCS, vol. 1178, pp. 305–314. Springer, Heidelberg (1996)

7. Erickson, J., Nayyeri, A.: Shortest non-crossing walks in the plane. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 297–308 (2011)
8. Polishchuk, V., Mitchell, J.S.: Touring convex bodies – a conic programming solution. In: Proceedings of the 17th Canadian Conference on Computational Geometry, pp. 290–293 (2005)
9. Dror, M., Efrat, A., Lubiw, A., Mitchell, J.S.B.: Touring a sequence of polygons. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), pp. 473–482 (2003)
10. Patrignani, M.: On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science* 17(5), 1061–1069 (2006)
11. Bespamyatnikh, S.: Computing homotopic shortest paths in the plane. *Journal of Algorithms* 49(2), 284–303 (2003)
12. Efrat, A., Kobourov, S.G., Lubiw, A.: Computing homotopic shortest paths efficiently. *Computational Geometry* 35(3), 162–172 (2006)
13. Duncan, C.A., Efrat, A., Kobourov, S.G., Wenk, C.: Drawing with fat edges. *International Journal of Foundations of Computer Science* 17(5), 1143–1164 (2006)
14. Mitchell, J.S., Polishchuk, V.: Thick non-crossing paths and minimum-cost flows in polygonal domains. In: Proceedings of the 23rd Annual Symposium on Computational Geometry (SoCG), pp. 56–65 (2007)
15. Cabello, S.: Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *Journal of Graph Algorithms and Applications* 10(2), 353–363 (2006)
16. Dujmović, V., Evans, W.S., Lazard, S., Lenhart, W., Liotta, G., Rappaport, D., Wismath, S.K.: On point-sets that support planar graphs. *Computational Geometry* 46(1), 29–50 (2013)
17. Katz, B., Krug, M., Rutter, I., Wolff, A.: Manhattan-geodesic embedding of planar graphs. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 207–218. Springer, Heidelberg (2010)
18. Hwang, F., Weng, J.: The shortest network under a given topology. *Journal of Algorithms* 13(3), 468–488 (1992)
19. Winter, P.: Euclidean Steiner minimal trees with obstacles and Steiner visibility graphs. *Discrete Applied Mathematics* 47(2), 187–206 (1993)
20. Fink, M., Haunert, J.-H., Mchedlidze, T., Spoerhase, J., Wolff, A.: Drawing graphs with vertices at specified positions and crossings at large angles. In: Rahman, M. S., Nakano, S.-i. (eds.) WALCOM 2012. LNCS, vol. 7157, pp. 186–197. Springer, Heidelberg (2012)
21. Hurtado, F., Tóth, C.: Plane geometric graph augmentation: A generic perspective. In: Pach, J. (ed.) *Thirty Essays on Geometric Graph Theory*, pp. 327–354. Springer, New York (2013)
22. Takahashi, J., Suzuki, H., Nishizeki, T.: Shortest noncrossing paths in plane graphs. *Algorithmica* 16(3), 339–357 (1996)
23. Kamousi, P., Chan, T.M., Suri, S.: Stochastic minimum spanning trees in Euclidean spaces. In: Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG), pp. 65–74 (2011)
24. Edelsbrunner, H., O’Rourke, J., Seidel, R.: Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing* 15(2), 341–363 (1986)

Stub Bundling and Confluent Spirals for Geographic Networks^{*}

Arlind Nocaj and Ulrik Brandes

Department of Computer & Information Science, University of Konstanz

Abstract. Edge bundling is a technique to reduce clutter by routing parts of several edges along a shared path. In particular, it is used for visualization of geographic networks where vertices have fixed coordinates. Two main drawbacks of the common approach of bundling the interior of edges are that (i) tangents at endpoints deviate from the line connecting the two endpoints in an uncontrolled way and (ii) there is ambiguity as to which pairs of vertices are actually connected. Both severely reduce the interpretability of geographic network visualizations.

We therefore propose methods that bundle edges at their ends rather than their interior. This way, tangents at vertices point in the general direction of all neighbors of edges in the bundle, and ambiguity is avoided altogether. For undirected graphs our approach yields curves with no more than one turning point. For directed graphs we introduce a new drawing style, confluent spiral drawings, in which the direction of edges can be inferred from monotonically increasing curvature along each spiral segment.

1 Introduction

We are interested in visualizing geographic networks given as a graph with fixed vertex coordinates and possibly other attributes. Although, for substantive reasons, there is often a relationship between the graph's adjacency structure and the spatial arrangement of its vertices, straight-line drawings are generally cluttered with areas of high edge-density and small-angle crossings.

A technique to reduce such clutter is edge bundling. Generalizing the idea of edge concentrators [17], edge bundles have been introduced in the context of hierarchically clustered graphs [11]. Sets of related edges are routed so that they meet, run concurrently, and then separate again, where edges are considered related if their projections on the cluster tree share a subpath. Note that the nodes of the cluster tree directly yield shared edge control points. Different bundling strategies have been introduced in force-directed layout of general graphs [12,24,19] and layered layout of directed acyclic graphs [23].

For graphs with given vertex coordinates, relatedness of edges is usually defined in terms of similarity of their straight-line realizations. Examples include similarities obtained from grid approximations [6,16,15], visibility graphs [22], or clusters in the four-dimensional space of pairs of vertex coordinates [9]. In the extreme, bundling techniques operate on the pixel level [25,8,14,27,13].

^{*} Research supported in part by DFG under grant GRK 1024. We are grateful to Sabine Cornelsen for valuable comments and suggestions on earlier drafts of this paper.

Because of the shared inner segments, it cannot be inferred from the drawing which subgraph of a bipartite clique a bundle actually represents, i.e., we cannot know whether the drawing is faithful [18]. Moreover, having edges meet requires that they deviate from the line through their vertices in a way that has no substantive meaning.

Both these problems can be avoided by bundling edges at their ends rather than in their interior. This idea has indeed been introduced in the context of geographic networks [4,20] and also forms the basis of flow maps [21,5,26] which can be seen as drawings of in- or out-stars.

We present novel such methods for drawing geographic networks with edges bundled at their ends. For undirected graphs, we refine the approach of Peng et al. [20]. Our main contribution is an approach for directed graphs based on a new drawing style for in-/out-stars called *confluent spiral drawings*. Confluent spirals consist of smooth drawings of each bundle in which edge directions are represented by increasing curvature so that no ambiguity is created in a combined drawing of all, say, in-stars of a directed graph.

The remainder of this paper is divided into three sections. In Sect. 2, we outline how edges are assigned to bundles. Our approaches for undirected and directed graphs are then described in Sects. 3 and 4 with a short discussion in Sect. 5.

2 Stub Bundling

We consider geographic networks consisting of a graph $G = (V, E)$ with fixed vertex coordinates $p = (p_v)_{v \in V}$ where $p_v = \begin{pmatrix} x_v \\ y_v \end{pmatrix} \in \mathbb{R}^2$. Coordinates might be defined extrinsically by, say, geographic locations, or derived from, say, a precomputed layout.

Our goal is to route the edges in such a way that readability of the network is improved over the corresponding straight-line drawing. The means in this work are bundled edges, curved routing, and color gradients.

A common objective of edge bundling is to reduce the total length of edges drawn. Since multiplicity along shared paths is ignored, bundling of interior segments of edges is attempted. For geographic networks, however, a more substantively relevant criterion is to be able to read off the general direction in which adjacent vertices are located. To represent this more accurately, and in addition to provide a faithful representation of adjacency, we bundle edges only at their ends, i.e., only with edges that share an endpoint. This type of bundling is referred to here as *stub bundling* to distinguish it from the bundling of edge interiors, or *interior bundling*.

The first step is to find a partition of the edges around each vertex into bundles (see fig. 1). To preserve their general direction, we use the angles between consecutive straight-line edges as our partition criterion. Each bundle is a set of half-edges incident on the same vertex and with similar direction.

For given angles α, γ , an (α, γ) -bundling is a coarsest partition such that

- the angle between any two half-edges in a bundle is at most α , and
- the angle between two consecutive edges in a bundle is at most γ .

Such bundlings are obtained easily by iteratively splitting adjacency lists at maximum angles (between consecutive edges). For each vertex $v \in V$, we start with a bundle containing all incident half-edges. The bundle is split at all occurrences of the maximum angle between consecutive edges in this bundle; in case of equiangular half-edges,

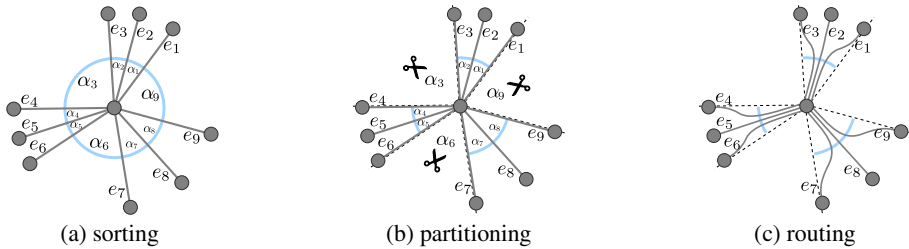


Fig. 1. Stub bundling: a cyclic sequence of edges is (a) split at maximum angles ($\alpha_3, \alpha_6, \alpha_9$) until (b) the angle range of each bundle is below a given threshold; then, (c) the first segment of each half-edge in a bundle is routed with the same tangent

where the consecutive angles are equal inside a bundle we split symmetrically into two (for bundles with even cardinality) or three (odd cardinality) smaller bundles. In the latter case the resulting bundles may have different (but symmetrical) cardinalities. This process is iterated until we obtain an (α, γ) -bundling. Note that, in contrast to the counterclockwise greedy splitting of [20], we do not accidentally split at small angles and we maintain a higher degree of symmetry.

The entire bundling step is thus carried out in time $\mathcal{O}(m \log \Delta)$, where m is the number of edges and Δ the maximum degree of a vertex, by sorting adjacency lists and splitting them hierarchically. Clearly, other bundling strategies, e.g. also based on distances rather than just directions, may be more appropriate for specific applications. It remains to show how to route the edges beyond the constraint that half-edges in the same bundle share an initial path.

3 Undirected Graphs

After bundling as described in the previous section, we need to decide on two things: in which direction to route the stub of a bundle, and how to connect the two extremal segments of each edge.

To ensure that edges can be followed easily, we allow only one turning point in the routing of an edge. More precisely, we draw each half-edge as a cubic Bézier curve (without turning point) to gain more control over the curve shape. Bézier curves are especially convenient because their tangents at endpoints can be prescribed so that edges in a bundle start in parallel and the two half edges of an edge can be linked smoothly.

Stub directions are determined from the straight-line segment connecting a vertex with the centroid of all neighbors in a bundle. This incorporates not only their directions but also their distances. For the present purpose this is considered a good approximation to the general direction of all edges in a bundle, but more general nodal templates for outgoing edges could be used [3].

We now describe in detail how to choose the control points of the Bézier curves. See fig. 2 for illustration. Let $e = \{v, w\} \in E$ be an edge and let $\Gamma(e, v)$ and $\Gamma(e, w)$ be

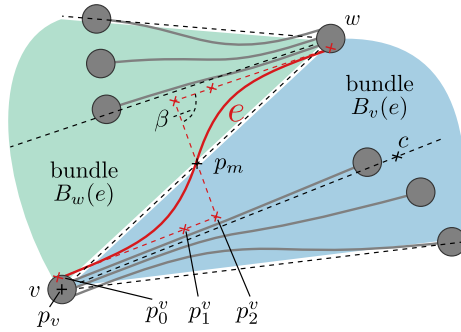


Fig. 2. Control points for routing undirected edge e

the two Bézier curves representing the half-edges of e . Further, let $B_v(e)$ and $B_w(e)$ be the bundles around v and w containing e . Let

$$p_m = \frac{1}{2}(p_v + p_w) + \left(\frac{|B_v(e)|}{|B_v(e)| + |B_w(e)|} - \frac{1}{2} \right) \cdot t_{\text{shift}} \cdot (p_w - p_v)$$

be the weighted midpoint on the segment between p_v and p_w and $t_{\text{shift}} \in [0, 1]$. More intuitively p_m is closer to p_v if $B_w(e)$ contains more edges than $B_v(e)$. This has the effect that larger bundles have longer parallel parts. Let c be the centroid of the end vertices of the edges bundled in $B_v(e)$ (excluding v). In order to define the control points of $\Gamma(e, v)$ we consider the *baseline* going through p_v and the centroid c . We first compute temporary control points, which lie on the baseline. Later these points will be shifted left and right to obtain a parallel routing.

We choose a branching angle β which is the same for every edge. This angle determines how long the edge will stay with the bundle until it branches off to enter the other bundle, and thus the smoothness of edges. Denote by p_2^v the point on the baseline such that the angle $\sphericalangle(p_v, p_2^v, p_m)$ is β . Another intermediate control point p_1^v is chosen on the segment between p_v and p_2^v with a smoothing parameter $t \in (0, 1)$, i.e., $p_1^v = p_v + t \cdot (p_2^v - p_v)$.

So far we have determined temporary control points p_v, p_1^v, p_2^v , and $\frac{1}{2}(p_2^v + p_2^w)$ for $\Gamma(e, v)$ and symmetrically for $\Gamma(e, w)$. These are refined to avoid overlap without introducing many crossings. Ordering edges around each vertex is a special case of the more general metro-line crossing minimization problem [2] but we find the simple heuristic of ordering stubs in a bundle $B_v(e)$ according to the opposite control point p_2^w to work sufficiently well. Control points are shifted left and right according to this ordering. Determining control points and ordering stubs in the same bundle does not increase the asymptotic running time of $\mathcal{O}(m \log \Delta)$ already caused by bundling.

Stub bundling is motivated by faithfulness and the substantive interest in directions at the ends of edges. Therefore, non-uniform rendering of edges can be used to highlight bundles and reduce the visual dominance of the less important interior of an edge by fading out colors toward the middle of an edge. Note the emphasis this creates in fig. 3 without eliminating the possibility to trace individual edges. In addition to the alternate bundling strategy, non-overlapping stub routing distinguishes our approach from that

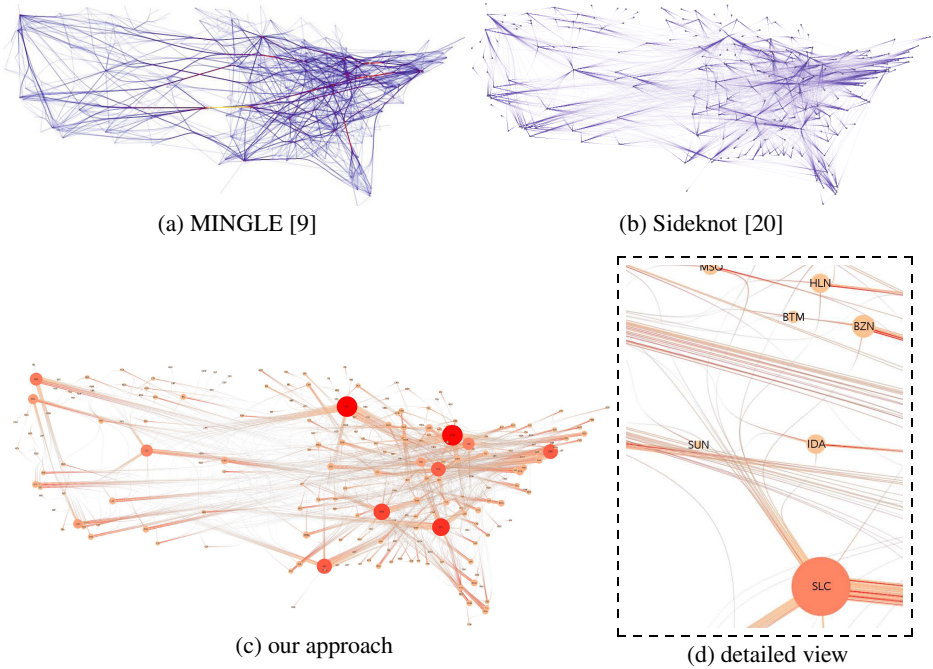


Fig. 3. US airlines graph: In our approach main edge directions and their strengths are visible from an overview perspective (c) but single edges can still be traced in a detailed view (d). Unlike the approach of (b), ours uses parallel routing of edges to facilitate the display of additional data attributes by varying width or color.

of [20] and facilitates the use of different widths and colors for edges in the same bundle to convey additional attribute information such as volume, frequency, time, and so on.

The total bundling process, edge partitioning and control point computation, of the US airlines graph took 0.07 seconds on an Intel Core i7-2600K CPU@3.40GHz with a single core (impl. in Java 6).

4 Directed Graphs: Confluent Spiral Drawings

To visualize directed geographic networks we break the symmetry of the previous approach as arrows and color gradients do not seem to work well for displaying orientation of stub-bundled edges. Depending on the meaning of edge orientations, we bundle only incoming or outgoing stubs. The problem of drawing a directed graph is thus reduced to the problem of drawing one in- or out-star per vertex.

We introduce a new drawing convention for such star-configurations. It is a variation of spiral trees [26] which have been introduced for flow maps [21], but based on confluent logarithmic spirals for smoother appearance and easier inference of orientation.

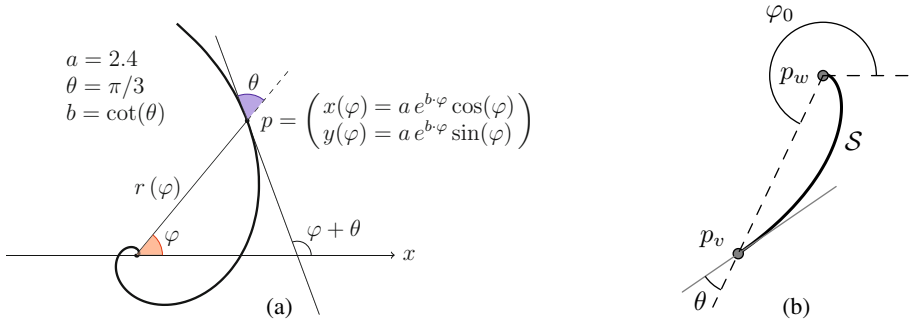


Fig. 4. (a) Logarithmic spiral with angle $\theta = \pi/3$, centered at origin, going through a point p . (b) Spiral used to represent a directed edge (v, w) . The constantly increasing curvature on the path from p_v towards p_w unambiguously indicates the orientation.

4.1 Logarithmic Spirals

A *logarithmic* (or *equiangular*) *spiral* is a curve which winds around a center, or vortex, and approaches it with an exponentially increasing curvature. The increase in curvature is determined by a constant θ . In effect, all rays out of the vortex are at angle θ to the tangents of intersection points with the spiral.

Formally, a logarithmic spiral in Euclidean space can be defined in polar coordinates (r, φ) relative to its vortex by $r(\varphi) = a \cdot e^{b \cdot \varphi}$, where $b = \cot \theta$ and $a \in \mathbb{R} \setminus \{0\}$ are fixed, and $\varphi \in \mathbb{R}$. Figure 4(a) shows an example.

We use a sequence of spiral segments to represent an edge between two vertices, and the vortex of each spiral corresponds to a target vertex. We define a spiral segment S from a start point $p_v \in \mathbb{R}^2$ to an end point $p_w \in \mathbb{R}^2$ with tangent angle $\theta \in (-\frac{\pi}{2}, 0) \cup (0, \frac{\pi}{2})$ as

$$S(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = p_w + |p_v - p_w| e^{-|b| \cdot t} \begin{pmatrix} \cos\left(\frac{b}{|b|}(\varphi_0 + t)\right) \\ \sin\left(\frac{b}{|b|}(\varphi_0 + t)\right) \end{pmatrix}, t \in [0, \infty),$$

where $b = \cot \theta$ and $\varphi_0 = \angle(\overrightarrow{p_w p_v}, \begin{pmatrix} 1 \\ 0 \end{pmatrix})$ is counterclockwise around p_w .

Note that $S(0) = p_v$. Although the curve has finite length, it never reaches p_w . Practically this is not a problem, since vertex w is represented by a graphical element with non-zero dimensions such as a disc. The spiral S goes clockwise around p_w if $\theta < 0$ and counterclockwise if $\theta > 0$. Figure 4(b) shows how a logarithmic spiral can be used to represent a directed edge from v to w .



Fig. 5. Two drawings of the same graph. The absence of edge (w_1, w_2) is apparent because the curve from p_{w_1} to p_{w_2} is not smooth.

Confluent Spirals. The term *confluent* was introduced in Dickerson et al. [7] for a drawing style that allows to draw larger classes of graphs in a planar way. We here use it more loosely, not requiring planarity. We say a drawing is a *confluent spiral drawing*, if each edge $e = (v, w) \in E$ is represented as follows:

- There is a continuously differentiable curve from p_v to p_w consisting of logarithmic-spiral segments.
- The logarithmic spiral of the last segment has vortex $\begin{pmatrix} x_w \\ y_w \end{pmatrix}$ and the segment starts at p where
 - either $p = \begin{pmatrix} x_v \\ y_v \end{pmatrix}$ or
 - p lies in the interior of another edge $(v, w') \in E$ with $w' \neq w$.

Furthermore, we do require that the curves representing outgoing edges of the same vertex do not intersect but in a shared prefix. Figure 5 shows a small graph and a corresponding drawing with confluent spirals.

4.2 Determining Confluent Spiral Trees

In this section, we introduce an algorithm to compute a confluent spiral drawing by computing a confluent spiral tree for each vertex. Later, we extend this algorithm to handle obstacles by adding further constraints to it.

The main difference compared to the spiral trees suggested by Buchin et al. [5] is that we want to have confluent drawings, which means that the intersection angle between two spirals is zero. This means that following a path from the root to some other vertex one never has to make a sharp turn. Due to this property it is not possible to apply the method of Buchin et al. [5]. Later the same authors [26] use a spiral tree as a basis to generate flow maps by minimizing a complex cost function to smooth the curves.

In contrast to the previous approach, we require the vortex of spirals not to be on the source but on the target vertex of an edge, which directly results in smooth curves, see fig. 6 for an example.

As a first step we apply the edge partitioning, as described in section 2. The result is for each vertex $v \in V$ a set of bundles containing outgoing edges of v . Each bundle is handled separately. Let B be a bundle with outgoing edges of v . For every edge $e = (v, w) \in B$ we determine a logarithmic spiral \mathcal{S} that is centered at p_w and either starts at p_v or branches out of another spiral in a confluent way such that $\sum_{e \in B} \text{length}(\mathcal{S}_e)$ is minimal. Note that although a spiral never reaches the vortex its length is finite:

$$\text{length}(\mathcal{S}) = \int_0^\infty |\mathcal{S}'(t)| dt = \|p_v - p_w\| \frac{\sqrt{1 + b^2}}{|b|}.$$

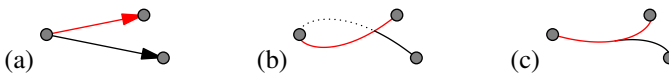


Fig. 6. (a) directed graph, (b) spiral tree approach of Buchin et al. [5]: spiral vortices at source, postprocessing required for smoothness, (c) our confluent spiral tree approach: spiral vortices at targets, smoothness inherent in confluent design

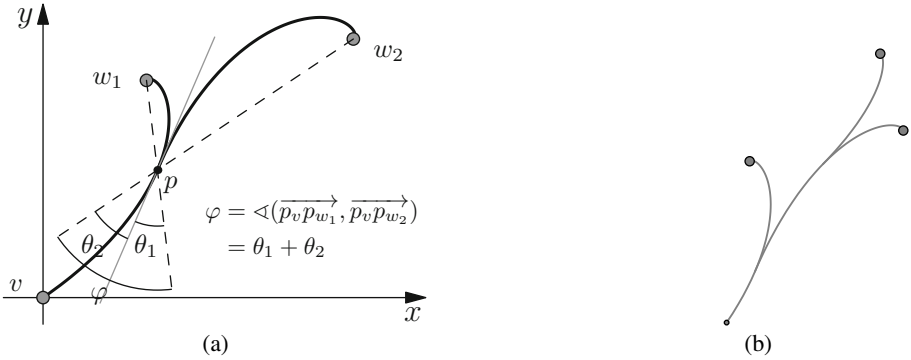


Fig. 7. (a) Construction of a confluent spiral segment with vortex w_2 branching off the parent spiral for (v, w_1) at p . The branching spiral is defined by $p, p w_2$ and the angle θ_2 which is determined by the parent spiral and the angle φ at p . (b) When a parent spiral is changed due to vertex movement, local adaptation by the other spirals is immediate. (NB: in the electronic version, this figure can be animated)

If a spiral \mathcal{S} branches off another spiral \mathcal{T} we refer to the latter as the *parent* spiral. The following heuristic is used to decide on the tree structure. Intuitively, we want edges leading to local targets to branch out earlier in the tree. Thus, consider the edges of a bundle B ordered by the distance of their targets from p_v . Other meaningful orderings, e.g., from data attributes, could be used too here, resulting in different trees.

We start with the first edge $(v, w) \in B$ as the *trunk*. This edge is represented by a logarithmic spiral with a predefined *trunk angle* $0 < \theta_0 < \pm\pi/6$. The upper bound $\pi/6$ ensures that the spiral approaches its vortex more directly, without orbiting around it.

Spiral segments for the other edges are allowed to branch off any already existing spiral segment \mathcal{T} subject to two constraints on the new spiral \mathcal{S} :

- \mathcal{S} must have an angle $\theta \in [\theta_{\min}, \theta_{\max}]$ (typically $\theta \in (0, \pm\pi/6)$).
- The tangent of \mathcal{S} at p must have the same slope and direction as that of \mathcal{T} .

The branching point p on the parent spiral \mathcal{T} is determined by trying k candidate points $(p_i = \mathcal{T}(i \frac{\pi}{k}), i \in 0, \dots, k-1)$ on \mathcal{T} and choosing the point that satisfies all constraints and results in the shortest spiral length. Note that \mathcal{T} and p completely determine \mathcal{S} as illustrated in Figure 7(a). If we cannot find a spiral satisfying the constraints, we postpone the current edge temporarily. Our experience so far is that edges need to be postponed rarely so that the overall runtime is in $\mathcal{O}(\sum_{B \in \mathcal{B}} k \cdot |B|^2) \subset \mathcal{O}(k n \Delta^2)$, where \mathcal{B} is the set of all stub bundles and $n = |V|$.

4.3 Avoiding Obstacles and Crossings

Avoiding edge crossings is very important to reduce visual complexity and improve readability. Furthermore, it is very important that edges not connected to a vertex have a certain distance to that vertex. This can be modelled by placing obstacles on the vertex positions. We extend our framework to deal with obstacles and crossings by adding them as constraints during the search of a parent spiral. We use an R-tree [10] as spatial index and add the vertices with their shape as obstacles into it. The spirals of the

already finished edges are approximated by s line segments and stored in the index too. For a possible branching point p we query the spatial index with s segments of the corresponding spiral to check whether they intersect with obstacles or other edges in the index. The creation of the R-tree index needs $\mathcal{O}(n \log n)$ time on average while maintaining and querying it takes $\mathcal{O}(s \log(n + s \Delta))$.

The intuitive interpretation of this method is that, if there is an obstacle for a desired branching point we will branch out in an earlier or later phase of the parent spiral to miss that obstacle. Although the resulting spirals will be longer, the readability will be improved. See fig. 8 for an illustration.

4.4 Edge Ordering and Parallel Routing

At this stage we have a confluent spiral tree for each bundle B . To reflect the data, in this case the different number of edges in the bundle, we route them in parallel until they branch to their targets.

Intuitively, walking along the outer contour of our tree gives us the required edge ordering. We determine this ordering by sorting the edges according to the branching point and branching side when traversing the underlying tree from the root vertex. With this ordering we then compute an offset curve to the approximated spiral, which is very similar to polygon offsetting. The offset will determine the thickness of the edge, which in turn can be used to represent, e.g., an edge attribute. See fig. 10 and fig. 9 for an example. Note that after applying an offset the result is not a true logarithmic spiral anymore. In practice this is not a problem as logarithmic spirals are eventually only approximated with cubic splines anyway [1].

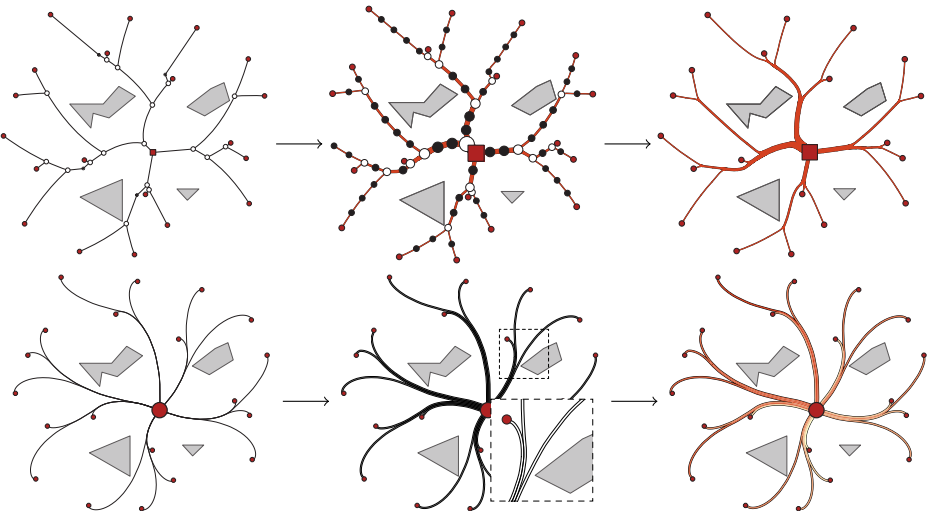


Fig. 8. Avoiding obstacles: Approach of Verbeek et al. [26] (top) and ours (bottom); spirals branch out *smoothly* from trunk at appropriate point to miss the obstacles. Parallel edge routing allows to map an edge attribute to the color; here node distance to the root is mapped (bottom-right).

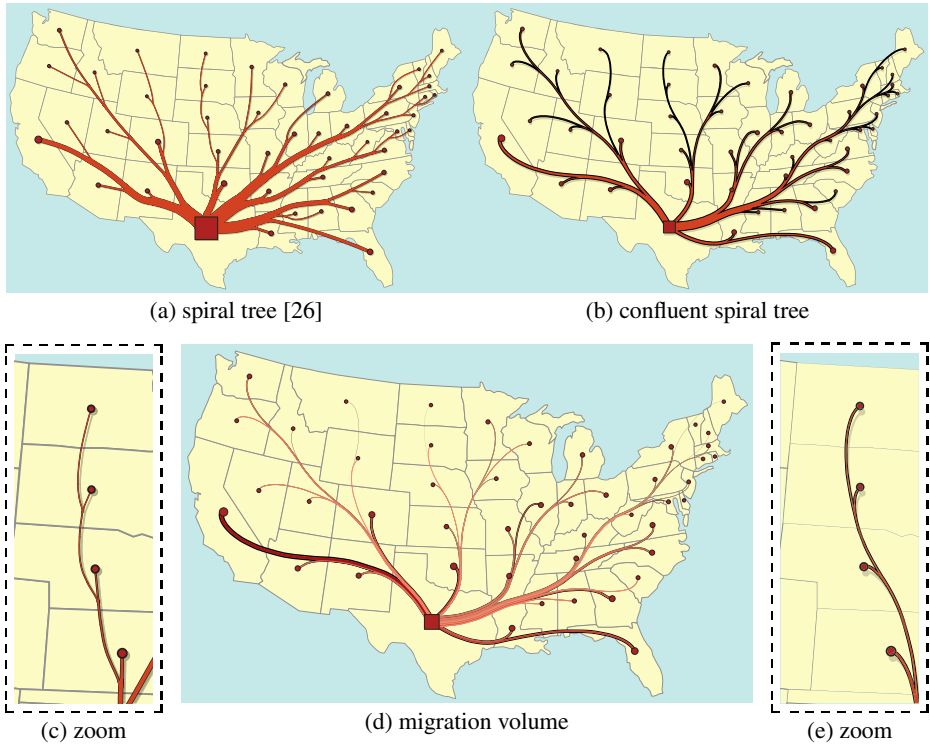


Fig. 9. Flow map of migration from Texas (1995-2000). The smooth linkage of confluent spirals (b) eliminates turns and thus not only yields more pleasing drawings but also facilitates the display of edge attributes (d). Note that edge direction can be inferred locally from every segment (e).

5 Discussion

We have presented drawing styles for the routing of undirected and directed edges in geographic networks using edge bundling at ends rather than interiors, and logarithmic-spiral trees that yield confluent drawings. Their main benefits are

- faithfulness (unambiguous representation of edges)
- stubs point in general direction of destinations
- edge widths and colors are still available for data attributes
- confluent flow maps of in- or out-stars

While initial feedback indicates that confluent spiral drawings are visually appealing, controlled user studies will have to show that they are effective.

As future work we plan to explore other, more data-driven, approaches to partition stubs into bundles, and we would like to prove guarantees on the tree structure of spiral segments and on avoidance of obstacles. For now, our application-oriented implementation is based largely on heuristics, but does layout networks with several thousands of edges essentially at interactive speed.

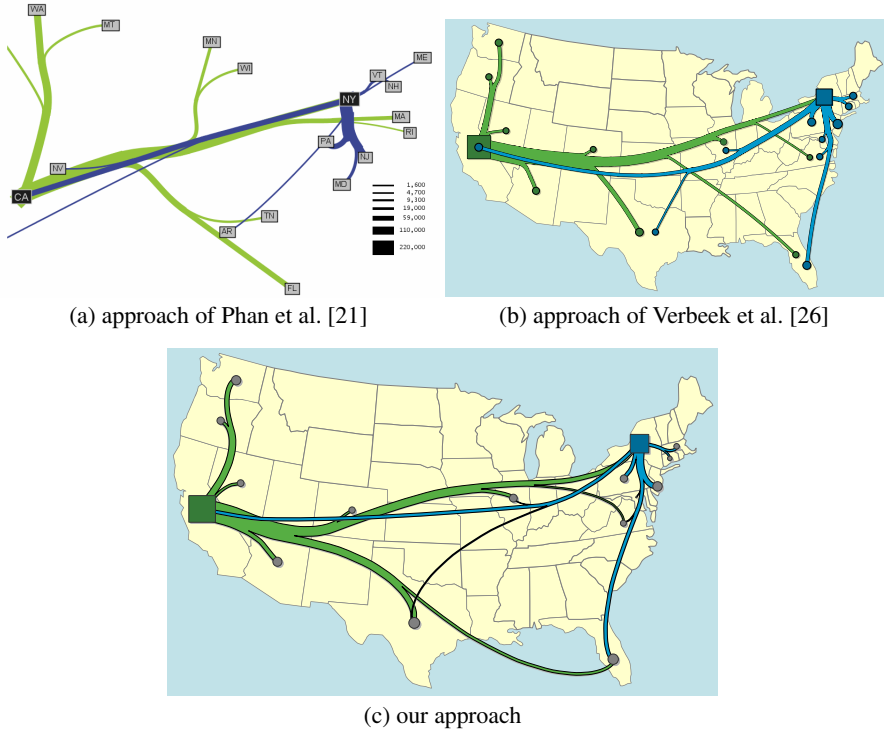


Fig. 10. Flow map of migration to California and New York (1995-2000, top 10 states of origin). Line widths indicate migration volume and are to scale across figures.

References

1. Baumgarten, C., Farin, G.: Approximation of logarithmic spirals. *Computer Aided Geometric Design* 14(6), 515–532 (1997)
2. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In: Kaufmann, M., Wagner, D. (eds.) *GD 2006*. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
3. Brandes, U., Shubina, G., Tamassia, R.: Improving angular resolution in visualizations of geographic networks. In: de Leeuw, W.C., van Liere, R. (eds.) *VisSym 2000*, pp. 23–32. Springer (2000)
4. Brandes, U., Wagner, D.: Using graph layout to visualize train interconnection data. *Journal of Graph Algorithms and Applications* 4(3), 135–155 (2000)
5. Buchin, K., Speckmann, B., Verbeek, K.: Angle-restricted steiner arborescences for flow map layout. In: Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.) *ISAAC 2011*. LNCS, vol. 7074, pp. 250–259. Springer, Heidelberg (2011)
6. Cui, W., Zhou, H., Qu, H., Wong, P.C., Li, X.: Geometry-based edge clustering for graph visualization. *IEEE Trans. on Visualization and Computer Graphics* 14(6), 1277–1284 (2008)
7. Dickerson, M., Eppstein, D., Goodrich, M.T., Meng, J.Y.: Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications* 9(1), 31–52 (2005)

8. Ersoy, O., Hurter, C., Paulovich, F.V., Cantareiro, G., Telea, A.: Skeleton-based edge bundling for graph visualization. *IEEE Trans. on Visualization and Computer Graphics* 17(12), 2364–2373 (2011)
9. Gansner, E.R., Hu, Y., North, S.C., Scheidegger, C.E.: Multilevel agglomerative edge bundling for visualizing large graphs. In: *Proc. of the IEEE Pacific Visualization Symposium 2011*, pp. 187–194. IEEE (2011)
10. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: *SIGMOD 1984*, pp. 47–57. ACM Press (1984)
11. Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. on Visualization and Computer Graphics* 12, 741–748 (2006)
12. Holten, D., van Wijk, J.J.: Force-directed edge bundling for graph visualization. *Computer Graphics Forum* 28(3), 983–990 (2009)
13. Hurter, C., Ersoy, O., Telea, A.: Smooth bundling of large streaming and sequence graphs. In: *Proc. of the IEEE Pacific Visualization Symposium*. IEEE (to appear, 2013)
14. Hurter, C., Ersoy, O., Telea, A.: Graph bundling by kernel density estimation. *Computer Graphics Forum* 31(3pt. 1), 865–874 (2012)
15. Lambert, A., Bourqui, R., Auber, D.: 3d edge bundling for geographical data visualization. *IEEE Trans. on Visualization and Computer Graphics*, 329–335 (2010)
16. Lambert, A., Bourqui, R., Auber, D.: Winding roads: Routing edges into bundles. *Computer Graphics Forum* 29(3), 853–862 (2010)
17. Newberry, F.J.: Edge concentration: A method for clustering directed graphs. *SIGSOFT Software Engineering Notes* 14(7), 76–85 (1989)
18. Nguyen, Q.H., Eades, P., Hong, S.: On the faithfulness of graph visualizations. In: *Proc. of the IEEE Pacific Visualization Symposium*. IEEE (to appear, 2013)
19. Nguyen, Q., Hong, S.-H., Eades, P.: TGI-EB: A new framework for edge bundling integrating topology, geometry and importance. In: Speckmann, B. (ed.) *GD 2011*. LNCS, vol. 7034, pp. 123–135. Springer, Heidelberg (2011)
20. Peng, D., Lu, N., Chen, W., Peng, Q.: Sideknot: Revealing relation patterns for graph visualization. In: *Proc. of the IEEE Pacific Visualization Symposium 2012*, pp. 65–72. IEEE (2012)
21. Phan, D., Xiao, L., Yeh, R., Hanrahan, P., Winograd, T.: Flow map layout. In: *Proc. of IEEE Symposium of Information Visualization 2005*, p. 29. IEEE (2005)
22. Pupyrev, S., Nachmanson, L., Bereg, S., Holroyd, A.E.: Edge routing with ordered bundles. In: Speckmann, B. (ed.) *GD 2011*. LNCS, vol. 7034, pp. 136–147. Springer, Heidelberg (2011)
23. Pupyrev, S., Nachmanson, L., Kaufmann, M.: Improving layered graph layouts with edge bundling. In: Brandes, U., Cornelsen, S. (eds.) *GD 2010*. LNCS, vol. 6502, pp. 329–340. Springer, Heidelberg (2011)
24. Selassie, D., Heller, B., Heer, J.: Divided edge bundling for directional network data. *IEEE Trans. on Visualization and Computer Graphics* 17 (2011)
25. Telea, A., Ersoy, O.: Image-based edge bundles: Simplified visualization of large graphs. *Computer Graphics Forum* 29(3), 843–852 (2010)
26. Verbeek, K., Buchin, K., Speckmann, B.: Flow map layout via spiral trees. *IEEE Trans. on Visualization and Computer Graphics* 17(12), 2536–2544 (2011)
27. Zinsmaier, M., Brandes, U., Deussen, O., Strobel, H.: Interactive level-of-detail rendering of large graphs. *IEEE Trans. on Visualization and Computer Graphics* 18(12), 2486–2495 (2012)

On Orthogonally Convex Drawings of Plane Graphs (Extended Abstract)

Yi-Jun Chang and Hsu-Chun Yen*

Dept. of Electrical Engineering, National Taiwan University
Taipei, Taiwan 106, Republic of China

Abstract. We investigate the bend minimization problem with respect to a new drawing style called *orthogonally convex drawing*, which is orthogonal drawing with an additional requirement that each inner face is drawn as an *orthogonally convex polygon*. For the class of bi-connected plane graphs of maximum degree 3, we give a necessary and sufficient condition for the existence of a no-bend orthogonally convex drawing, which in turn, enables a linear time algorithm to check and construct such a drawing if one exists. We also develop a flow network formulation for bend-minimization in orthogonally convex drawings, yielding a polynomial time solution for the problem. An interesting application of our orthogonally convex drawing is to characterize internally triangulated plane graphs that admit floorplans using only orthogonally convex modules subject to certain boundary constraints.

Keywords: Bend minimization, floorplan, orthogonally convex drawing.

1. Introduction

An *orthogonal drawing* of a plane graph is a planar drawing such that each edge is composed of a sequence of horizontal and vertical line segments with no crossings. A classic optimization problem in orthogonal drawing is to minimize the number of *bends*, namely, the *bend-minimization* problem. The problem is NP-complete in the most general setting, i.e., for planar graphs of maximum degree 4 [5]. Subclasses of graphs with bend-minimization of orthogonal drawing tractable include planar graphs of maximum degree 3, series-parallel graphs, and graphs with fixed embeddings [3,12], etc.

Most of the orthogonal drawing algorithms reported in the literature can be roughly divided into two categories, one uses flow or matching to model the problem (e.g., [2,3,12]), whereas the other tackles the problem in a more graph-theoretic way by taking advantage of structure properties of graphs (e.g., [9,10,8]). The former usually solves a more general problem, but requires higher

* Corresponding author (e-mail: yen@cc.ee.ntu.edu.tw). Research supported in part by National Science Council of Taiwan under Grants NSC-100-2221-E-002-132-MY3 and NSC-100-2923-E-002-001-MY3.

time complexity. On the contrary, algorithms in the latter focus on specific kinds of graphs, resulting in linear time complexity in many cases.

In this paper, we introduce a new type of orthogonal drawing called *orthogonally convex drawing*, which requires that each inner face be an *orthogonally convex polygon*. A polygon is *orthogonally convex* if for any horizontal or vertical line, if two points on the line are inside a polygonal region, then the entire line segment between these two points is also inside the polygonal region. The study of this new drawing style is motivated by an attempt to learn more about the geometric aspect of orthogonal drawing, which, in the dual setting, is closely related to *rectangular dual* and *rectilinear dual* which are well-studied in floor-planning and contact graph representations [6,11,14]. Note that if we consider standard convexity instead of orthogonal convexity in the setting of no-bend orthogonal drawing, the problem becomes the "inner rectangular drawing" studied in [7]. There are several recent results on rectilinear duals in cartographic applications, see, e.g., [1]. For other perspectives of orthogonal drawing, the reader is referred to [4] for a survey chapter.

Our contributions include the following:

1. A new drawing style called *orthogonally convex drawing* is introduced, and a necessary and sufficient condition, along with a linear time testing algorithm, is given for a bi-connected plane 3-graph (i.e., of maximum degree 3) to admit a no-bend orthogonally convex drawing.
2. A flow network formulation is devised for the bend-minimization problem of orthogonally convex drawing.
3. By combining the above no-bend orthogonally convex drawing algorithm and the flow network formulation, a polynomial time algorithm (in $O(n^{1.5} \log^3 n)$ time) for constructing a bend-optimal orthogonally convex drawing is presented.
4. We apply our analysis of no-bend orthogonally convex drawing to characterizing internally triangulated graphs that admit floorplans using only orthogonally convex modules that can be embedded into a rectilinear region with its boundary order-equivalent to a given orthogonally convex polygon.

2. Preliminaries

Given a graph $G = (V, E)$, we write $\Delta(G)$ to denote the maximum degree of G . Graph G is called a d -graph if $\Delta(G) \leq d$. A *path* P of G is a sequence of vertices (v_1, v_2, \dots, v_n) such that $\forall 1 \leq i \leq n, v_i \in V$ and $\forall 1 \leq i \leq n-1, (v_i, v_{i+1}) \in E$. We write $V(P)$ to denote the set of vertices $\{v_1, \dots, v_n\}$, and $E(P)$ to denote the edge set $\{(v_i, v_{i+1}) | 1 \leq i < n\}$ of P . Given two paths P' and P , we write $P' \subseteq P$ if P' is a subsequence of P , and $P' \subset P$ if $P' \subseteq P$ and $P' \neq P$. P is called a *cycle* if $v_1 = v_n$. Unless stated otherwise, paths and cycles are assumed to be *simple* throughout this paper, in the sense that there are no repeated vertices other than the starting and ending vertices. A drawing of a planar graph divides the plane into a set of connected regions, called *faces*. A *contour* of a face F is

the cycle formed by vertices and edges along the boundary of F . A cycle that is the boundary, i.e., contour, of a face is called a *facial cycle*. The contour of the outer face is denoted as C_O . If G is bi-connected, contours of all the faces are simple cycles.

In our subsequent discussion, we adopt some of the notations and definitions used in [9,10]. A cycle C divides a plane graph G into two regions. The one that is inside (resp., outside) cycle C is called the *interior region* (resp., *outer region*) of C . We use $G(C)$ to denote the subgraph of G that contains exactly C and vertices and edges residing in its interior region. An edge $e = (u, v)$ in the outer region of C is called a *leg* of C if at least one of the two vertices u and v belongs to C . C is *k-legged* if C contains exactly k vertices that are incident to some legs of C . These k vertices are called *legged-vertices* of C . If $\Delta(G) \leq 3$, every legged-vertex v of C is incident to exactly one leg e of C . Note that 3-legged cycles coincide with the so-called *complex triangles* in the dual setting, which play a crucial role in the study of rectilinear duals [11,14].

We call a face or a cycle *inner* if it is not the outer one. If an inner face or inner cycle intersects with the outer one, then we call it *boundary face* or *boundary cycle*. A *contour path* P of a cycle C is a path on C such that P includes exactly two legged-vertices x and y of C , and x and y are the two endpoints of P . Therefore, each k -legged cycle has exactly k contour paths. If a contour path intersects with (i.e., shares some edges with) the outer cycle, we call it *boundary contour path*. In fact, each boundary contour path is a subpath of C_O . Each contour path P of C is incident to exactly one face, denoted as $F_{C,P}$, in the outer region of C . As an illustrating example, consider Figure 1. F_0 is the outer face of G . Consider two cycles $C_1 = (s, t, u, v, s)$ and $C_2 = (x, b, i, a, z, y, c, x)$ (both drawn in bold line). C_1 is a non-boundary 2-legged cycle, of which two legged-vertices are t and v , and two legs are (t, q) and (v, r) . C_1 is also a facial cycle, which is the contour of F_1 . C_2 is a boundary 3-legged cycle, of which three legged-vertices are x, y , and z . $P_1 = (t, u, v)$ is a contour path of C_1 . $P_2 = (z, a, i, b, x)$ is the boundary contour path of C_2 . We have $F_{C_1, P_1} = F_2$ and $F_{C_2, P_2} = F_0$.

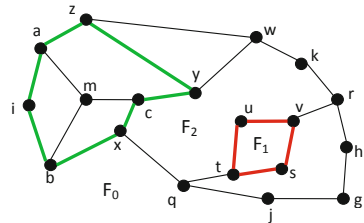


Fig. 1. Illustration of some terms about cycles and paths

Let $D(G)$ be an orthogonal drawing of plane graph G with outer cycle C_O . Given a cycle C , we use $D(C)$ (or equivalently $D(F)$ if C is the contour of a face F) to denote the drawing of C in $D(G)$. $D(C)$ is always a simple polygon as long as C is simple. We call $D(G)$ an *orthogonally convex drawing* of G if $D(F)$ is an *orthogonally convex polygon* for every face F other than the outer one. We use $bc(D(G))$ to denote the bend count, i.e., the total number of bends, of $D(G)$.

In an orthogonal drawing $D(G)$, $ang_G(v)$ denotes the interior angle of v in polygon $D(C)$. We called v a *convex corner*, *non-corner*, and *concave corner* of

C if $ang_C(v)$ is 90° , 180° , and 270° , respectively. A *corner* in the drawing $D(G)$ is either a bend on some edge, or a vertex v of G such that $ang_C(v) \neq 180^\circ$ for some C . If v is a non-corner of C , v is on a side of polygon $D(C)$.

From Section 3 to Section 5, graphs under the name G are assumed to be bi-connected with $\Delta(G) \leq 3$, and may have multi-edges.

3. No-Bend Orthogonally Convex Drawing

Among existing results concerning orthogonal drawings, Rahman et al. [10] gave a necessary and sufficient condition for a bi-connected plane 3-graph to admit a no-bend orthogonal drawing, and they devised an algorithm to test the condition, and subsequently construct the drawing if one exists.

Theorem 1 ([10]). *A bi-connected plane 3-graph G has a no-bend orthogonal drawing iff G satisfies the following three conditions:*

1. *There are four or more 2-vertices (i.e., vertices of degree 2) of G on $C_O(G)$.*
2. *Every 2-legged cycle contains at least two 2-vertices.*
3. *Every 3-legged cycle contains at least one 2-vertex.*

Theorem 1 clearly holds even when G has multi-edges, as such graphs do not have no-bend orthogonal drawings. The no-bend orthogonal drawing algorithm in [10] performs the following steps recursively: (1) reducing the original graph G into a structurally simpler graph G^* by collapsing the so-called "maximal bad cycles", (2) drawing G^* in a rectangular fashion, and (3) plugging in the orthogonal drawings of those maximal bad cycles to the rectangular drawing¹ of G^* to yield a no-bend orthogonal drawing of G .

Bad cycles in Step (1) are cycles that are 2-legged or 3-legged if the four designated corner vertices in C_O are considered as legged-vertices. Intuitively, bad cycles are cycles that violate the conditions under which a graph admits a rectangular drawing. For instance, consider the graph in Figure 1. If $\{h, i, j, k\}$ are the 4 designated vertices, then $(w, z, a, i, b, x, c, y, w)$ (a 3-legged cycle as i is considered a legged-vertex) is a bad cycle, whereas $(r, v, u, t, q, j, g, h, r)$ (a 4-legged cycle including legged-vertices h and j) is not a bad cycle. *Maximal bad cycles* are bad cycles that are not contained in $G(C)$ for another bad cycle C . Step (2) involves computing the rectangular drawing of an input graph with four designated corner vertices on $C_O(G)$. It is known that such a graph with four designated vertices admits a rectangular drawing if and only if every 2-legged cycle contains at least two designated vertices, and every 3-legged cycle contains at least one designated vertex [13]. As shown in [10], the G^* (with each of the maximal bad cycles contracted to a single vertex) always meets the condition for the existence of a rectangular drawing. The reader is referred to [10] for more.

Our goal in this section is to give a similar necessary and sufficient condition for graphs to have no-bend orthogonally convex drawings.

¹ A *rectangular drawing* of a graph is a no-bend orthogonal drawing such that each interior face is a rectangle and the boundary of the outer face also forms a rectangle.

Lemma 1. *Consider a no-bend orthogonally convex drawing $D(G)$ of a graph G . For every 2-legged cycle C with legged-vertices x and y and a contour path P of C , the number of convex corners of $D(C)$ in $V(P) \setminus \{x, y\}$ (i.e., the set of vertices along path P excluding x and y) must be at least 1 more than that of concave corners, if either (1) C is a boundary cycle and P is its boundary contour path, or (2) C is non-boundary and P is any of its contour paths.*

We are now in a position to give one of our main results.

Theorem 2. *A bi-connected plane 3-graph G admits a no-bend orthogonally convex drawing if and only if the three conditions in Theorem 1 and the following two additional conditions hold: (1) every non-boundary 2-legged cycle contains at least one 2-vertex on each of its contour paths, and (2) every boundary 2-legged cycle contains at least one 2-vertex on its boundary contour path.*

The necessity of Theorem 2 follows from Lemma 1. A modification to the no-bend orthogonal drawing algorithm described above yields a constructive proof of the sufficiency of Theorem 2. Based on an implementation described in [10], we have the following result.

Theorem 3. *There is a linear time algorithm to construct a no-bend orthogonally convex drawing $D(G)$ if G admits one.*

4. An Alternative Condition

An alternative necessary and sufficient condition is given in this section to characterize bi-connected 3-plane graphs admitting no-bend orthogonally convex drawings, facilitating a min-cost flow formulation for the bend-minimization problem. As Theorem 2 indicates, contour paths along (boundary or non-boundary) 2-legged cycles play a vital role in orthogonally convex drawing. Due to possible overlaps of 2-legged cycles and complex intersections between contour paths, it becomes difficult to capture the amount of convex/concave corners along contour paths in a min-cost flow formulation. To ease this problem, we identify two types of cycles, namely, *proper* and *improper* cycles, which are later used to characterize the presence of orthogonally convex drawings.

Let G^c denote the graph resulting from contracting every 2-vertex of G . Since we require G to be of maximum degree 3, G^c must be 3-regular. A 2-legged cycle of G is called *improper* if its two legs correspond to the same edge in G^c ; otherwise, it is called *proper*. See Figure 2 for instance. Due to 3-regularness of G^c and the fact that the two legs of an improper cycle C are the same edge e in G^c , there remains nothing outside $G(C)$ except the leg e . Therefore, improper cycles must be boundary cycles, or conversely, all non-boundary 2-legged cycles are proper. It is also easy to observe that a 2-legged cycle C of G with two leg-vertices x and y is improper iff for the non-boundary contour path P of C , the boundary of $F_{C,P}$ intersects C_O of G in exactly 1 path. Again consider Figure 2. Note that F_1 and F_2 correspond to the $F_{C,P}$ of the 2-legged cycles drawn as bold lines in the left and right figures, respectively. The boundary of F_1 intersects C_O

in exactly one path (x, y) , whereas the boundary of F_2 intersects C_O in two paths (z, a) and (y, b, c) .

Definition 1. A path P of G is called critical if there is a proper 2-legged cycle C such that: (1) P is a contour path of C , (2) if C is a boundary 2-legged cycle, P is the boundary contour path of C , and (3) P does not edge-intersect with any proper 2-legged cycle other than C that is contained in $G(C)$.

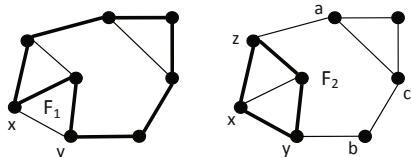


Fig. 2. Proper and improper 2-legged cycles. Left: An improper 2-legged cycle (drawn as a bold line) with leg-vertices x, y . Right: A proper 2-legged cycle (drawn as a bold line) with leg-vertices y, z .

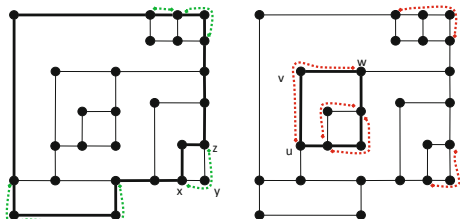


Fig. 3. Critical paths and S_G in a plane graph. Left: Paths in S_G . Right: Critical Paths

To proceed further, we require the following two lemmas.

Lemma 2. For any bi-connected plane 3-graph G , the critical paths of G are edge-disjoint.

Lemma 3. Let P be a path satisfying (1) and (2) in Definition 1. If P is not critical, there must be a critical path P' such that $P' \subset P$.

Note that the requirement of properness of 2-legged cycles in Definition 1 is essential in the sense that Lemma 2, which is crucial in the subsequent context, is not true if we remove that requirement. Given a path P with endpoints x and y , we write $P_{(x \frown y)}$ to denote the "open" version of P , i.e., excluding x and y . That is, $P_{(x \frown y)}$ consists of $V(P) \setminus \{x, y\}$ and $E(P)$.

Instead of basing on contour paths as in Theorem 2, our new characterization for no-bend orthogonally convex drawing is based upon two kinds of paths defined over proper and improper 2-legged cycles, namely, critical paths defined above for proper 2-legged cycles and a set of paths called S_G associated with improper 2-legged cycles in graph G . S_G is defined to be the set of all paths $C_O \setminus P_{(x \frown y)}$ for every boundary contour path P of an improper 2-legged cycle with two legged-vertices x, y . Note that internal vertices in paths of S_G must have degree 2, and paths in S_G must be in C_O , and hence $P \in S_G$ iff P is a boundary contour path of a facial cycle C that has only one boundary contour path. The following fact summarize the observation.

Fact 1. Let P be a path of G with two end-vertices x and y . The following three statements are equivalent: (1) P is in S_G ; (2) P is the boundary contour path of a facial cycle C that intersects C_O of G in exactly one path; (3) P is $C_O \setminus P'_{(x \frown y)}$, for some boundary contour path P' of an improper 2-legged cycle with two legged-vertices x and y .

To have better grasp of critical paths and S_G , consider Figure 3 in which a no-bend orthogonally convex drawing of a plane graph G is shown. In the left figure, the four dotted paths are those in S_G , which are edge-disjoint. Let C be the 2-legged cycle drawn as a bold line, and P be its boundary contour path. We have $C_O \setminus P_{(x \frown z)} = (x, y, z)$. In the right figure, the five dotted paths are critical paths, which are edge-disjoint. Let C be the 2-legged cycle drawn as a bold line. We have (1) the path (u, v, w) is one of its contour paths, (2) C is a non-boundary 2-legged cycle, and (3) P does not edge-intersect with any proper 2-legged cycle other than C that is contained in $G(C)$. A path in S_G is either contained in exactly one critical path or intersects with no critical path. The reader is encouraged to verify that the graph satisfies the conditions stated in Theorem 2 and Theorem 4, and the orthogonally convex drawing satisfies the conditions in Lemma 4.

The following theorem enables us to characterize no-bend orthogonally convex drawings in terms of critical paths and S_G .

Theorem 4. Suppose a bi-connected plane 3-graph G has a no-bend orthogonal drawing. G has a no-bend orthogonally convex drawing iff the following conditions are satisfied:

1. Every critical path of G contains at least one 2-vertex.
2. $C_O \setminus P$ contains at least one 2-vertex for every $P \in S_G$.

We note that both S_G and the set of all critical paths can be found in linear time. The algorithm is basically a contour edge-traversal of each face with a mechanism of detecting repeated adjacent faces.

5. Flow Formulation for Bend-Minimization

In this section, we tailor the planar min-cost flow formulation originally designed for orthogonal drawing [12] to coping with orthogonal convexity. To make our subsequent discussion clear, we use *arc* and *node* instead of edge and vertex, respectively, in describing a flow network. A *min-cost flow network* is a directed multi-graph $N = (W, A)$ associated with four functions: *lower bounds* $\lambda : A \rightarrow \mathbb{Z}_{\geq 0}$, *capacities* $\mu : A \rightarrow \mathbb{Z}_{\geq 0} \cup \{\infty\}$, *costs* $c : A \rightarrow \mathbb{Z}_{\geq 0}$, *demands* $b : W \rightarrow \mathbb{Z}$. A map $f : A \rightarrow \mathbb{Z}_{\geq 0}$ is a *flow* if the following constraints are met for each node v and arc a :

$$b(v) + \sum_{(u,v) \in A} f(u, v) - \sum_{(v,u) \in A} f(v, u) = 0, \quad \lambda(a) \leq f(a) \leq \mu(a)$$

The cost of a flow f is $c(f) = \sum_{a \in A} f(a) \times c(a)$. The flow network $N_G = (W_G, A_G)$ associated with a bi-connected plane 3-graph G is

- $W_G = W_V \cup W_F$, where W_V and W_F are the vertex set and face set (including the outer face) of G , respectively, Furthermore, $\forall u_v \in W_V, b(u_v) = 2$ if $deg_G(v) = 3$; $b(u_v) = 0$ if $deg_G(v) = 2$. $\forall u_F \in W_F, b(u_F) = -4$ if F is an inner face; $b(u_F) = 4$ if F is the outer face.
- $A_G = A_V \cup A_F$, where
 - $A_V = \{(u_v, u_F), (u_F, u_v) | deg(v) = 2\} \cup \{(u_v, u_F) | deg(v) = 3\}$, where $v \in V(G), F \in \text{face}(G), v$ incident to F . $\forall a \in A_V, \lambda(a) = 0, \mu(a) = 1$, and $c(a) = 0$.
 - $A_F = \{(u_F, u_H) | F, H \in \text{face}(G), \text{ and } F \text{ adjacent to } H\}$ is a multi-set of arcs between faces, and the number of (u_F, u_H) in A_F equals the number of shared edges e in contours of F and H . We use $(u_F, u_H)_e$ to indicate the specific arc that corresponds to the shared edges e . $\forall a \in A_F, \lambda(a) = 0, \mu(a) = \infty$, and $c(a) = 1$.

Although our definition of N_G is slightly different from the original one given in [12], the validity of N_G is apparent as the following explains. Every flow f in N_G corresponds to an orthogonal drawing $D(G)$, and vice versa, such that

- $f(u_v, u_F) - f(u_F, u_v) = -1, 0, 1$ means v is a concave corner, non-corner, convex corner in $D(F)$, respectively,
- $f(u_F, u_H)_e$ is the number of bends on e that are concave corners in $D(F)$ and convex corners in $D(H)$, and
- the total number of bends in $D(G)$ equals $c(f)$.

Fact 2. *Let S_1 (resp., S_2) be any subset of edges (resp., vertices) along the contour of a face F . For any $e \in S_1$, we write F_e to denote the face incident to e other than F . For a flow f in N_G and its corresponding orthogonal drawing D , we must have $\sum_{e \in S_1} [f(u_{F_e}, u_F)_e - f(u_F, u_{F_e})] + \sum_{v \in S_2} [f(u_v, u_F) - f(u_F, u_v)]$ equals the difference between the numbers of convex corners and concave corners in S_1 and S_2 of $D(F)$.*

Lemma 4. *A bi-connected plane 3-graph G admits a no-bend orthogonally convex drawing iff there is a no-bend orthogonal drawing (not necessarily orthogonally convex) such that (1) for every critical path P along a contour path of 2-legged cycle C , $\#_{cc}(P_{(x \frown y)}) > \#_{cv}(P_{(x \frown y)})$ in $F_{C,P}$, and (2) for every P in S_G , $\#_{cc}(P_{(x \frown y)}) \leq 3 + \#_{cv}(P_{(x \frown y)})$ in the outer face, where P has endpoints x and y , and $\#_{cv}(P_{(x \frown y)})$ and $\#_{cc}(P_{(x \frown y)})$ represent the numbers of convex and concave corners, respectively, of $P_{(x \frown y)}$.*

In what follows, we show how to construct a flow network N'_G from N_G in such a way that a flow of N'_G corresponds to an orthogonal drawing meeting the conditions stated in Lemma 4. Initially we set $N'_G = N_G$.

- $\forall P \in S_G$ with endpoints x, y and outer face F' , add a new node u_P to $W(N'_G)$, and two arcs $(u_{F'}, u_P), (u_P, u_{F'})$ to $A(N'_G)$. We set $b(u_P) = 0, \lambda(u_{F'}, u_P) = \lambda(u_P, u_{F'}) = 0, \mu(u_{F'}, u_P) = 3, \mu(u_P, u_{F'}) = \infty$, and $c(u_{F'}, u_P) = c(u_P, u_{F'}) = 0$. We redirect all the arcs in the current $A(N'_G)$ of the following forms: $(u_{F'}, u_v), (u_v, u_{F'}), (u_{F'}, u_F)_e, (u_F, u_{F'})_e$ for all

$v \in V(P) \setminus \{x, y\}$, $F \in S_{P,F'}$, $e \in E(P)$ by replacing $u_{F'}$ with u_P . Due to Fact 2, Statement 2 of Lemma 4 holds.

- \forall critical path P with endpoints x, y , C the 2-legged cycle for which P is its contour path, and S the set of faces in $G(C)$ that border P , add a new node u_P to $W(N'_G)$, and a new arc $(u_{F_C,P}, u_P)$ to $A(N'_G)$. We set $b(u_P) = 0$, $\lambda(u_{F_C,P}, u_P) = 1$, $\mu(u_{F_C,P}, u_P) = \infty$, and $c(u_{F_C,P}, u_P) = 0$. We redirect all the arcs in the current $A(N'_G)$ of the following forms: $(u_{F_C,P}, u_{P'})$, $(u_{P'}, u_{F_C,P})$, $(u_{F_C,P}, u_v)$, $(u_v, u_{F_C,P})$, $(u_{F_C,P}, u_F)_e$, $(u_F, u_{F_C,P})_e$ for all $P' \in S_G$ such that $P' \subseteq P$, $v \in V(P) \setminus \{x, y\}$, $F \in S$, $e \in E(P)$ by replacing $u_{F_C,P}$ with u_P . Due to Fact 2, Statement 1 of Lemma 4 holds.

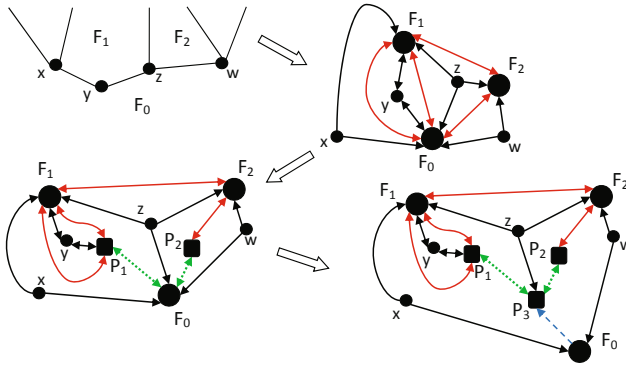


Fig. 4. Illustration of the construction of N'_G : F_0 is the outer face, $P_1 = (x, y, z)$ and $P_2 = (z, w)$ are two paths in S_G , $P_3 = (x, y, z, w)$ is a critical path

For an illustrating example, consider Figure 4 in which the up-left picture is a portion of a graph G with N_G depicted in the up-right. The down-left one illustrates the result of adding two additional nodes representing P_1 and P_2 (the newly added arcs are drawn in dotted line). The down-right one illustrates the result of adding an additional node representing critical path P_3 (the newly added arc is drawn in dashed line).


The validity of the above construction follows from critical paths being mutually edge-disjoint (Lemma 2), and every path in S_G is either a subpath of a critical path or intersects with no critical paths. Note that the number of newly added arcs and nodes is linear in $n = V(G)$, and the maximum possible value of the minimum cost is also $O(n)$. Following an $O(n^{1.5} \log^3 n)$ time algorithm in [2], we have

Theorem 5. *For any bi-connected plane 3-graph G , we can construct a bend-minimized orthogonally convex drawing in $O(n^{1.5} \log^3 n)$ time.*

6. An Application to Floor Planning

In this section, we show an application of orthogonally convex drawing to floor planning. A plane graph is *internally triangulated* if all the inner faces are triangles. For any internally triangulated plane graph $G = (V, E)$, a *rectilinear dual* is a partition of a simple orthogonal polygon (denoted as R) into $|V(G)|$ simple orthogonal regions, one for each vertex, such that two region have a side-contact iff their corresponding vertices adjacent to each other.

Two polygons are said to be *order-equivalent* if they admit the same circular order (in counter-clockwise orientation) of angles. For instance, the following

two figures  are order-equivalent. Let Q be an orthogonal polygon, we use Q -*floorplan* to denote a rectilinear dual whose boundary (the R in the definition of rectilinear dual) is order-equivalent to Q . A floorplan is called orthogonally convex if all the boundaries of $|V(G)|$ simple orthogonal regions are orthogonally convex polygons. In this section, graphs under the name G_{dual} are assumed to be simple, connected, internally triangulated plane graph.

Lemma 5. *For any simple, connected, internally triangulated plane graph G_{dual} , there is a unique bi-connected 3-regular plane multi-graph G_{primal} such that G_{dual} is the weak dual² of G_{primal} , and the following properties are hold: (1) G_{primal} does not have any non-boundary 2-legged cycle, and (2) internal faces (which are orthogonal polygons) of an orthogonal drawing of G_{primal} form a rectilinear dual of G_{dual} .*

We remark that although G_{dual} is required to be simple, G_{primal} may still have multi-edges. Since G_{primal} is bi-connected and $\Delta(G_{primal}) \leq 3$, the results in the previous sections can be applied.

Let $C_O = (v_1, v_2, \dots, v_s, v_1)$ be the boundary cycle of G_{dual} , which need not be a simple cycle. Then, a triangulated plane multi-graph G' is constructed by adding a new vertex t in the outer face of G_{dual} , and then triangulate the outer face by adding edge (v_i, t) for $1 \leq i \leq s$. Take the dual of G' yields G_{primal} . See Figure 5 for an illustration.

Given an orthogonally convex polygon Q , our goal is to characterize graphs that admit orthogonally convex Q -floorplans, and subsequently realize such floorplans. We use $\text{numSide}(P)$ to denote the number of sides of polygon P with non-corner vertices neglected.

Lemma 6. *Let G be a bi-connected plane 3-graph (may have multi-edges) with k boundary critical paths. We have $\min\{\text{numSide}(D(C_O)) \mid D \text{ is an orthogonally convex drawing of } G\} = \max\{4, 2k - 4\}$. Further, for any orthogonally convex polygon Q of $\text{numSide}(Q) \geq \max\{4, 2k - 4\}$, there is an orthogonally convex drawing $D(G)$ such that $D(C_O)$ is order-equivalent to Q .*

The concept of critical paths turns out to be pretty clean in the dual setting. We use T_G to denote the block-cutvertex tree of G . We will see in Lemma 7

² The weak dual of a plane graph is the subgraph of the dual graph excluding the vertex (and edges) corresponding to the unbounded (i.e., outer) face.

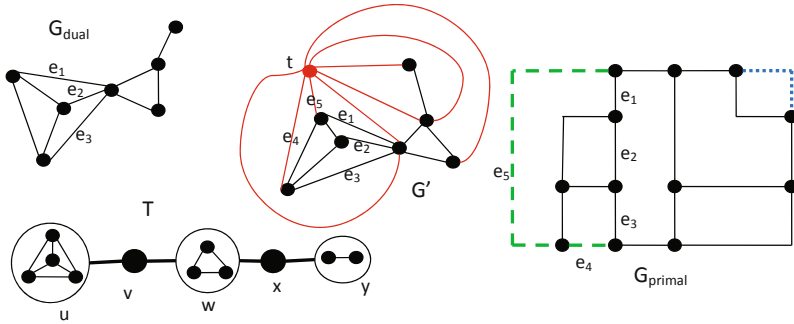


Fig. 5. The construction of G_{primal}

that leaves in $T_{G_{\text{dual}}}$ actually have one-to-one correspondence to critical paths in G_{primal} . Let (v, u) be an edge in $E(T_{G_{\text{dual}}})$ such that v is a cut-vertex. Now u must be a block. Let $V_{v,u}$ be the vertex set of the component in $G_{\text{dual}} \setminus \{v\}$ that contains some vertices in block u , and $F_{v,u}$ denote the corresponding face set in G_{primal} . Since G_{dual} is internally triangulated, the edges in $E(G_{\text{dual}})$ that link v to vertices in $V_{v,u}$ must be located consecutively in the circular list of edges incident to v that describes the combinatorial embedding of G_{dual} . We denote the edge set as $E_{v,u}$. According to the definition of duality of plane graphs and the algorithm for constructing G_{primal} from G_{dual} , these edges form a path in G_{primal} . We write $C_{v,u}$ to denote the cycle that is the boundary of union of faces in $F_{v,u}$. For instance, in Figure 5 the set $E_{v,u}$ is $\{e_1, e_2, e_3\}$, which forms the non-boundary contour path with respect to $C_{v,u} = (e_1, e_2, e_3, e_4, e_5)$.

Lemma 7. $\{ \text{Boundary contour path of } C_{v,u} \mid u \text{ is a leaf of } T_{G_{\text{dual}}}, (v, u) \in E(T_{G_{\text{dual}}}) \}$ is the set of boundary critical paths in G_{primal} .

In Figure 5, the boundary contour paths of $C_{v,u}$ and $C_{x,y}$ are the paths drawn in dashed and dotted lines, respectively. These two paths are the boundary critical paths of G_{primal} . Following Lemmas 5, 6, 7 and Theorem 3, we have

Theorem 6. For any internally triangulated graph G_{dual} and orthogonally convex polygon Q , let k be the number of leaves in the block-cutvertex tree of G_{dual} . G_{dual} admits an orthogonally convex Q -floorplan iff $\text{numSide}(Q) \geq \max\{4, 2k - 4\}$. The floorplan can be constructed in linear time.

7. Conclusion

We studied a new drawing style called orthogonally convex drawing from both combinatorial and algorithmic viewpoints. It would be interesting to see whether results/techniques developed in our work could be extended to other types of convex versions of contact graph representations or floorplans.

References

1. Alam, Md. J., Biedl, T., Felsner, S., Kaufmann, M., Kobourov, S. G., Ueckert, T.: Computing Cartograms with Optimal Complexity. In: Symposium on Computational Geometry (SoCG 2012), pp. 21–30 (2012)
2. Cornelsen, S., Karrenbauer, A.: Accelerated Bend Minimization. In: Speckmann, B. (ed.) GD 2011. LNCS, vol. 7034, pp. 111–122. Springer, Heidelberg (2011)
3. Di Battista, G., Liotta, G., Vargiu, F.: Spirality and Optimal Orthogonal Drawings. *SIAM Journal on Computing* 27(6), 1764–1811 (1998)
4. Duncan, C.A., Goodrich, M.T.: Planar Orthogonal and Polyline Drawing Algorithms. In: Tamassia, R. (ed.) *Handbook of Graph Drawing and Visualization*, ch. 7. CRC Press (2013)
5. Garg, A., Tamassia, R.: On the Computational Complexity of Upward and Rectilinear Planarity Testing. *SIAM Journal on Computing* 31(2), 601–625 (2001)
6. Kozminski, K., Kinnen, E.: Rectangular Dual of Planar Graphs. *Networks* 15, 145–157 (1985)
7. Miura, K., Haga, H., Nishizeki, T.: Inner Rectangular Drawings of Plane Graphs. *International Journal of Computational Geometry and Applications* 16(2-3), 249–270 (2006)
8. Rahman, M. S., Egi, N., Nishizeki, T.: No-bend Orthogonal Drawings of Series-Parallel Graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 409–420. Springer, Heidelberg (2006)
9. Rahman, M.S., Nakano, S., Nishizeki, T.: A Linear Algorithm for Bend-Optimal Orthogonal Drawings of Triconnected Cubic Plane Graphs. *Journal of Graph Algorithms and Applications* 3, 31–62 (1999)
10. Rahman, M.S., Nishizeki, T.: Orthogonal Drawings of Plane Graphs Without Bends. *Journal of Graph Algorithms and Applications* 7, 335–362 (2003)
11. Sun, Y., Sarrafzadeh, M.: Floorplanning by Graph Dualization: L-shaped Modules. *Algorithmica* 10, 429–456 (1993)
12. Tamassia, R.: On Embedding a Graph in the Grid with the Minimum Number of Bends. *SIAM Journal on Computing* 16, 421–444 (1987)
13. Thomassen, C.: Plane Representations of Graphs. In: Bondy, J.A., Murty, U.S.R. (eds.) *Progress in Graph Theory*, pp. 43–69. Academic Press, Canada (1984)
14. Yeap, K., Sarrafzadeh, M.: Floor-Planning by Graph Dualization: 2-Concave Rectilinear Modules. *SIAM Journal on Computing* 22, 500–526 (1993)

Planar and Plane Slope Number of Partial 2-Trees

William Lenhart¹, Giuseppe Liotta²,
Debajyoti Mondal³, and Rahnuma Islam Nishat⁴

¹Department of Computer Science, Williams College, MA, USA

²Dipartimento di Ingegneria Elettronica e dell'Informazione,
Universita' degli Studi di Perugia, Italy

³Department of Computer Science, University of Manitoba, Canada

⁴Department of Computer Science, University of Victoria, Canada
lenhart@cs.williams.edu, liotta@diei.unipg.it,
jyoti@cs.umanitoba.ca, rnishat@uvic.ca

Abstract. We prove tight bounds (up to a small multiplicative or additive constant) for the plane and the planar slope numbers of partial 2-trees of bounded degree. As a byproduct of our techniques, we answer a long standing question by Garg and Tamassia about the angular resolution of the planar straight-line drawings of series-parallel graphs of bounded degree.

1 Introduction

A *drawing* of a graph G in \mathbb{R}^2 maps each vertex of G to a point and each edge of G to a Jordan arc such that an edge does not contain a vertex other than its endpoints, no edge crosses itself, edges do not meet tangentially, and edges sharing a common end-vertex do not cross each other. A *planar graph* is a graph that admits a *planar drawing*, i.e. a drawing such that no two edges intersect except at their common end-points. A *plane graph* is a planar graph together with a combinatorial embedding, i.e. a prescribed set of faces including a prescribed outer face. A *plane drawing* of a plane graph G is a planar drawing that realizes the combinatorial embedding of G .

The *slope number* of a straight-line drawing Γ of a planar graph G is the number of distinct slopes of the edges of Γ . Every plane (planar) graph admits a plane (planar) straight-line drawing [1], i.e. a drawing where the edges are mapped to straight line segments. The *planar slope number* of G is the smallest slope number over all planar straight-line drawings of G . If G is a plane graph, the *plane slope number* of G is the smallest slope number over all plane straight-line drawings of G .

The problem of computing drawings of planar graphs with maximum degree four, using only horizontal and vertical slopes, has long been studied in graph drawing through the research on orthogonal and rectilinear graph drawing (see, e.g., [1]). In a seminal paper, Dujmović et al. [2] extend this study to non-orthogonal slopes, and give tight upper and lower bounds (expressed as functions of the number n of vertices) on the plane slope numbers of several graph families including plane 3-trees and plane 3-connected graphs. They also ask whether the plane slope number of a plane graph of maximum degree Δ can be bounded by a function $f(\Delta)$. Keszegh et al. [7] answer the question affirmatively proving that, for a suitable constant c , the plane slope number of a plane

graph of bounded degree Δ is at most $O(c^\Delta)$. In the same paper, Keszegh et al. establish a $3\Delta - 6$ lower bound for the plane slope number of the plane graphs of maximum degree at most Δ , which motivates additional research on reducing the gap between upper and lower bound. The question is studied by Jelínek et al. [5] who prove that the plane slope number of plane partial 3-trees is $O(\Delta^5)$. Also Kant, Dujmović et al., Mondal et al. independently show that the plane slope number of cubic 3-connected plane graphs is six [2,6,9], whereas the slope number (i.e., when the drawings may contain edge crossings) of cubic graphs is four [10].

In this paper we prove tight bounds (up to a small multiplicative or additive constant) for the plane and the planar slope numbers of planar 2-trees of bounded degree. Our results extend previous papers concerning the planar and plane slope numbers of proper subfamilies of the partial 2-trees. Namely, Jelínek et al. [5] prove that the planar slope number of series-parallel graphs with maximum degree three is at most three. Knauer et al. [8] show that the plane slope number of outerplane graphs with maximum degree $\Delta \geq 4$ is at most $\Delta - 1$ and that $\Delta - 1$ slopes are sometimes necessary. As a byproduct of our techniques, we answer a long standing open problem by Garg and Tamassia [3], who ask whether $\Omega(\frac{1}{\Delta^2})$ is a tight lower bound on the *angular resolution* of series-parallel graphs of degree Δ (i.e. they ask whether these graphs admit planar straight-line drawings where minimum angle between any two consecutive edges is $\Omega(\frac{1}{\Delta^2})$). More precisely, our results can be listed as follows.

- We prove that the planar slope number of a partial 2-tree of maximum degree Δ is at most 2Δ and there exist partial 2-trees whose planar slope number is at least Δ if Δ is odd and at least $\Delta + 1$ if Δ is even (Section 3).
- We prove that the plane slope number of a plane partial 2-tree of maximum degree Δ is at most 3Δ and there exist plane 2-trees whose plane slope number is at least $3\Delta - 3$ if Δ is even and at least $3\Delta - 4$ if Δ is odd (Section 4).
- We show that a partial 2-tree G of maximum degree Δ admits a planar straight-line drawing with angular resolution $\frac{\pi}{2\Delta}$. If G is a plane graph, a plane straight-line drawing of G exists whose angular resolution is $\frac{\pi}{3\Delta}$ (Section 5). The previously best known bound was $\frac{1}{48\pi\Delta^2}$, established by varying the input embedding [3].

2 Decomposition Trees and Universal Slope Sets

In this section we recall some known concepts. Throughout the paper “drawing” means “planar straight-line drawing”; “plane drawing” means “plane straight-line drawing”.

SPQ Trees and Block-cut Vertex Trees. Let G be a 2-connected graph. A *separation pair* is a pair of vertices whose removal disconnects G . A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph G_{uv} of G such that $\{u, v\}$ is not a split pair of G_{uv} . We call vertices u and v the *poles* of G_{uv} . Note that a split component of G need not be 2-connected.

A *2-connected series-parallel graph* is recursively defined as follows. A simple cycle with three edges is a 2-connected series-parallel graph. The graph obtained by

replacing an edge of a 2-connected series-parallel graph with a path is a 2-connected series-parallel graph. The graph obtained by adding an edge between the vertices of a non-adjacent separation pair $\{u, v\}$ of a 2-connected series-parallel graph is a 2-connected series-parallel graph. Let G be a 2-connected series-parallel graph. An *SPQ-tree* T of G is a rooted tree describing a recursive decomposition of G into its split components. The nodes of T are of three types: S , P , or Q . Each node μ has an associated graph called the *skeleton* of μ and denoted by $skeleton(\mu)$. Starting from a split pair $\{s, t\}$ of G , T recursively describes the split components of G as follows. The root of T is a P -node corresponding to G ; its skeleton is defined as in the "Parallel case" below.

- *Base case:* The split component H is an edge. Then H corresponds to a Q -node of T whose skeleton is this edge. The Q -nodes are the leaves of T .
- *Series case:* The split component H is a 1-connected graph with split components H_1, \dots, H_k ($k \geq 2$) and cut vertices $c_i = H_i \cap H_{i+1}$. Then H corresponds to an S -node μ of T . The graph $skeleton(\mu)$ is a chain e_1, \dots, e_k of edges such that $e_i = (c_{i-1}, c_i)$, where $c_0 = s$ and $c_k = t$. The children of μ are the roots of the *SPQ-trees* of H_1, \dots, H_k .
- *Parallel case:* Otherwise, the split component H is 2-connected and its split components are H_1, \dots, H_k ($k \geq 2$). Then H has $skeleton(\mu)$ consisting of a set of parallel edges e_1, \dots, e_k between s and t , one for each H_i . The children of μ are the roots of the *SPQ-trees* of H_1, \dots, H_k .

Figure 1(a) and (b) show a 2-connected series-parallel graph and its *SPQ-tree*, which is uniquely determined by the choice of the initial split pair. Note that no P -node (S -node) has a P -node (S -node) as a child. Let T be an *SPQ-tree* of a 2-connected series-parallel graph G and let μ be a node of T . The *pertinent graph* of μ is the subgraph of G whose *SPQ-tree* is the subtree of T rooted at μ , as shown in Figure 1(c). The *virtual edge* of μ is an edge in the skeleton of the parent of μ that represents the pertinent graph of μ . Hence for every internal (i.e., non- Q) node μ in T , each edge in $skeleton(\mu)$ is a virtual edge of some child of μ .

If μ is P -node, then we associate with μ another graph $frame(\mu)$, called the *frame* of μ , which is formed by replacing each edge e in $skeleton(\mu)$ with the skeleton of the child node whose virtual edge is e , as shown in Figure 1(d). Every vertex in a frame corresponds to a unique vertex of G . Given a vertex v of G , the *first frame* of v is the frame that is closest to the root of T and contains v . For any split pair $\{u, v\}$ in a 2-connected series-parallel graph G with n vertices, an *SPQ-tree* having $\{u, v\}$ as reference pair can be computed in $O(n)$ time [4].

A graph G is a *partial 2-tree* (or, has tree-width at most 2) if and only if each 2-connected component of G is either series-parallel or consists of a single edge. Let G be a 1-connected graph. The *block-cut vertex tree* of G , denoted by *BC-tree*, is a graph with vertex set $B \cup C$ such that B consists of one vertex for each *block* (maximal 2-connected subgraph) of G and C consists of one vertex for each cut vertex of G . There is an edge from $b \in B$ to $c \in C$ in the *BC-tree* if and only if the vertex of G represented by c belongs to the block represented by b .

Universal Slope Sets and Free Wedges. Let G be a graph with vertex set V . For a vertex $v \in V$, we denote the degree of v by $\delta(v)$. Hence the *maximum degree* of G is

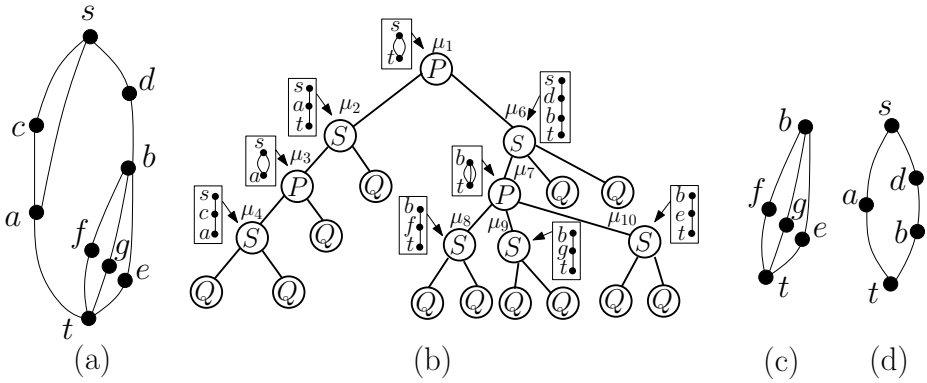


Fig. 1. (a) A 2-connected series-parallel graph G . (b) A SPQ -tree of G , where the internal nodes are labeled with μ_1, \dots, μ_{10} . The skeleton of $\mu_i, 1 \leq i \leq 10$, is drawn in the box associated with μ_i . (c) The pertinent graph of μ_7 . (d) The frame graph of μ_1 .

$\Delta = \max_{v \in V} \delta(v)$. The *excess* of v is $\epsilon(v) = \Delta - \delta(v)$. If H is a subgraph of G , $\delta_H(v)$ and $\epsilon_H(v)$ are the restrictions of $\delta(v)$ and $\epsilon(v)$ to H .

A set S of slopes is *universal* for a family of planar graphs \mathcal{G} if every graph $G \in \mathcal{G}$ admits a planar straight-line drawing such that the slope of every edge in the drawing belongs to S . We consider universal slope sets defined as follows.

Definition 1. Given a positive integer $k \geq 2$, let $\alpha = \frac{\pi}{2k}$. Define S_k to be the set of slopes $i \cdot \alpha$, for $0 \leq i \leq 2k - 1$.

We prove the upper bounds on plane and planar slope numbers showing that there is a value of k depending on Δ such that S_k is universal for the partial 2-trees. In our constructions, we guarantee that some wedge shaped regions of the plane can be used for recursive drawing. In particular, for any $r > 0$, point $p \in \mathbb{R}^2$, and angle ϕ , a ϕ -wedge at p of radius r is a sector of angular measure ϕ in the disk of radius r centered at p . For convenience, we will often omit reference to r , since any suitably small value of r suffices for our purposes. Let v be a vertex in some planar straight-line drawing Γ . A wedge with its apex at v in Γ is a *free wedge* at v if the wedge intersects the drawing Γ only at v . The angular measure of our free wedges will depend on the degree and excess of the corresponding vertices.

3 Slope Number of Partial 2-trees

In this section we present upper and lower bounds on the planar slope number of partial 2-trees of maximum degree Δ . We start by studying 2-connected series-parallel graphs (Section 3.1) and then we extend the study to all partial 2-trees (Section 3.2).

3.1 2-connected Series-parallel Graphs

In this section we show that S_Δ is a universal slope set for the family of 2-connected series-parallel graphs with maximum degree Δ .

Lemma 1. *Let \mathcal{G} be the family of 2-connected series-parallel graphs having maximum degree at most Δ . Then S_Δ is universal for \mathcal{G} .*

Proof. The argument is based on a construction that recursively computes a drawing of G ; the proof is by induction on the number of P -nodes in an SPQ -tree of G . Let T be an SPQ -tree of G having split pair $\{s, t\}$ associated with its root and let $m \in S_\Delta$. Since G is 2-connected, T must have at least one P -node, e.g., the root of T . We show that G admits a drawing Γ using only slopes from S_Δ that satisfies the following properties.

- (1) Graph G is drawn within a triangle $\triangle abc$ having $\angle bac = (\delta(s) - 0.5)\alpha$ and $\angle abc = (\delta(t) - 0.5)\alpha$ (i.e. every edge is either in the interior or on the boundary of $\triangle abc$).
- (2) Vertices s and t are located at a and b , respectively.
- (3) Segment \overline{ab} has slope m .
- (4) The edges incident to s are drawn using consecutive slopes of S_Δ , as are the edges incident to t .
- (5) At each vertex $v \notin \{s, t\}$ in the drawing of G , there is a free $\epsilon(v)\alpha$ -wedge at v contained in $\triangle abc$.

Let a and b be two distinct points on a line of slope $m \in S_\Delta$, and let c be the point of intersection of the lines through a and b having slopes $m + (\delta(s) - 0.5)\alpha$ and $m - (\delta(t) - 0.5)\alpha$, respectively.

Base Case: Assume that T has a single P -node, which must be the root of T since G is 2-connected. The frame of the P -node consists of a set of paths of length at least 1 (and at most one path of length 1). We draw s and t on a and b , respectively. We draw one of these paths between s and t along the segment ab with slope m (if there is a path of length one, then we draw that path along ab ; otherwise, any of the paths can be used). The remaining paths are drawn inside $\triangle abc$ using slopes $m + i\alpha, i = 1, \dots, \delta(s) - 1$ at a and slopes $m - i\alpha, i = 1, \dots, \delta(t) - 1$ at b for the edges incident to s and t , respectively; we use slope m for all other edges. See Figure 2 (a) for an example.

Let Γ be the computed drawing; by construction, Γ is crossing-free and it only uses slopes from S_Δ . Also, the paths from s to t lie within $\triangle abc$ and for each vertex $v \notin \{s, t\}$ there is an empty wedge of angle at least $\epsilon(v)\alpha$ with its apex at v that is completely contained within $\triangle abc$; this wedge is in the “ c side” of the path containing v , i.e., in the half-plane defined by the line through a, b and containing c . Hence, Γ satisfies all invariant Properties (1)-(5).

Induction Step: Suppose now that any 2-connected series-parallel graph having at most j P -nodes in some SPQ -tree admits a drawing that only uses slopes from S_Δ and that satisfies Properties (1)-(5). Let G be a 2-connected series-parallel graph having $j + 1$ P -nodes in some SPQ -tree T . As above, the root of T is a P -node and its frame consists of a set of paths Π_1, \dots, Π_k of length at least 1. We will draw them in a fashion similar to the base case but with one important difference: we do not use consecutive slopes for the edges of the paths incident to s and t , but we leave room for the (recursive) drawings of the pertinent graphs associated with each virtual edge incident to s or t .

To do this, for each $i = 1, \dots, k$, let e_i be the virtual edge incident to s in Π_i and let μ_{e_i} be the node of T corresponding to the virtual edge e_i (note that μ_{e_i} is either a P - or Q -node of T). Further, let $\delta_{e_i}(s)$ be the degree of s in the pertinent graph of μ_{e_i} . Then e_1 is drawn using slope m , and for each $i > 1$, e_i is drawn using

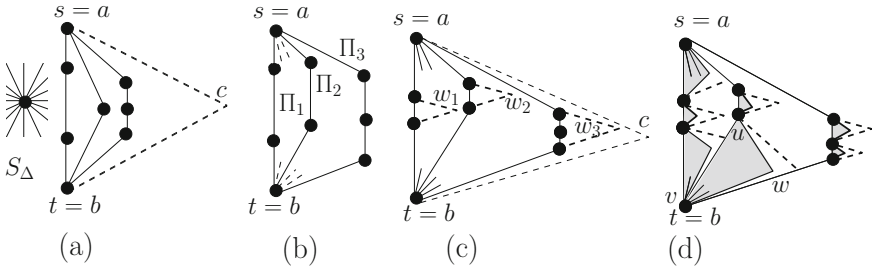


Fig. 2. (a) An example drawing for Base case, where $\delta(s) = \delta(t) = 3$ and $\Delta = 4$. (b) Illustration for Inductive step, where $\delta_{e_1}(s) = 3$, $\delta_{e_2}(s) = 1$, $\delta_{e_1}(t) = 2$, and $\delta_{e_2}(t) = 3$. (c) Illustration for $u_i v_i w_i$ in dashed line, and (d) recursive construction.

slope $slope(e_{i-1}) + \delta_{e_{i-1}}(s)\alpha$. The edges of Π_i incident to t are positioned similarly, beginning with slope m but decreasing the slopes as we move from one path to the next. See Figure 2 (b) for an example.

For every subpath $\Pi_i \setminus \{s, t\}$ with at least two vertices ($1 \leq i \leq k$), we draw the subpath using a sufficiently small line segment as follows. Let u_i, v_i be the endvertices of that subpath $\Pi_i \setminus \{s, t\}$. Let w_i be the point of intersection of the half-lines from u_i and v_i having slopes $slope(u_i v_i) + (\Delta - 0.5)\alpha$ and $slope(u_i v_i) - (\Delta - 0.5)\alpha$, respectively. We draw the paths such that $\Delta u_i v_i w_i$ lies in the region bounded by Π_i and Π_{i+1} . If $i + 1 < k$, then $u_i v_i w_i$ lies within Δabc . An example is illustrated in Figure 2(c) in dashed line.

Now that the frame of the root of T has been drawn, let $e = uv$ be a (drawn) virtual edge of a path Π_i of the frame and let μ_e be its corresponding P - or Q -node in T . Let w be the point of intersection of the lines from u and v having slopes $slope(e) + (\delta_e(u) - 0.5)\alpha$ and $slope(e) - (\delta_e(v) - 0.5)\alpha$, respectively. We recursively draw the pertinent graph of μ_e within Δuvw . If $\{u, v\} \cap \{s, t\} = \emptyset$, then Δuvw is contained in $\Delta u_i v_i w_i$, which by construction does not intersect any part of the already drawn edges. If e is incident to s or t (i.e. either $u = s$ or $v = t$), then Δuvw does not intersect the edge e' of Π_{i+1} incident to s or t , because by construction, the slope of e' is $slope(e) + \delta_e(s)\alpha$. Finally, observe that Δuvw does not intersect any other triangle that contains the drawing of a pertinent graph associated with a node of T which is not in the subtree rooted at μ_e . See Figure 2(d) for an illustration.

The observations above, together with the fact that the drawing of the frame graph of the root is crossing-free and that it satisfies Properties (1)-(4), imply that G admits a drawing that only uses slopes from S_Δ satisfying Properties (1)-(4). To see that Property (5) is also satisfied, note that each path Π_i is drawn as a convex (or linear, for $i = 1$) chain. Thus at each vertex $v \notin \{s, t\}$ has an angle of at least π between its two consecutive (virtual) edges in its first frame. The drawing of G uses two consecutive sets of slopes at v , since the pertinent graph for the two nodes corresponding to each of those two virtual edges is drawn using consecutive slopes. This leaves a free wedge of angular measure $\pi - \delta(v)\alpha = 2\Delta\alpha - \delta(v)\alpha = \Delta\alpha + \epsilon(v)\alpha > \epsilon(v)\alpha$ at v , establishing Property (5). \square

3.2 Partial 2-trees

In this section we extend the result of Lemma 1 to partial 2-trees of maximum degree Δ by proving that S_Δ is universal for these graphs. We shall focus on connected partial 2-trees, since every connected components can be drawn independently of the others.

Lemma 2. *Let \mathcal{G} be the family of partial 2-trees having maximum degree at most Δ . Then S_Δ is universal for \mathcal{G} .*

Proof. We assume that G is connected and has at least one edge. Let T be the block-cutvertex tree of G (see Figures 3(a)–(b)). We build the desired drawing of G by drawing subgraphs of G corresponding to subtrees of T , starting with the leaves of T . The drawing of G produced will have the following properties:

- (a) For some split pair $\{s, t\}$ of G , G is drawn inside a $\delta(s)\alpha$ -wedge with apex s .
- (b) s and t are located on a line of slope $m \in S_\Delta$.
- (c) The edges incident to s are drawn using consecutive slopes of S_Δ , as are the edges incident to t .
- (d) The wedge of Property (a) is bounded by rays from s in directions $m - 0.5\alpha$ and $m + (\delta(s) - 0.5)\alpha$.

If G is 2-connected, then Lemma 1 establishes the existence of a drawing with the desired properties: Property (b) follows obviously from Property (1) of Lemma 1, Property (c) from Property (3) of Lemma 1, and properties (a) and (d) from properties (1), (2) and (4) of Lemma 1.

Otherwise, let s be a cut vertex of G and choose s as the root of T . The parent of each block vertex B of T is a cut vertex s_B of G . For each such block B , choose a vertex t_B in B such that $\{s_B, t_B\}$ is a split pair for B . Each leaf of T is a block vertex B and by Lemma 1, it can be drawn with split pair $\{s_B, t_B\}$ inside a wedge with apex s_B and angular measure $\delta_B(s_B)\alpha$ so that properties (a)–(d) hold.

Assume now that x is a vertex of T for which all subgraphs of T have been drawn respecting properties (a)–(d). Then x represents either a block of G or a cut vertex of G . If x represents a cut vertex v of G , then let T_1, \dots, T_k represent the subtrees of x . Let G_v be the subgraph of G corresponding to the subtree of T with root v . For each of the trees T_i , v is at the apex of the $\delta_{T_i}(v)\alpha$ -wedge in which the subgraph G_i of G corresponding to T_i has been drawn. These wedges can all be rotated about v so that they use consecutive slopes in S_Δ , as shown in Figure 3(c). Thus the subgraph of G corresponding to the union of T_1, \dots, T_k has been drawn in a wedge of angle $\delta_{G_1}(v)\alpha + \dots + \delta_{G_k}(v)\alpha = \delta_{G_v}(v)\alpha$ with apex v such that properties (a)–(d) are satisfied. Note that by Property (a), the drawing of G_v can be made small enough to lie completely inside the free wedge and so does not intersect any other portion of the drawing.

Suppose now that x represents a block B of G . The children of B in T represent the cut vertices of G that belong to B ; let v be one of the child cut vertices of B . Draw B in the manner described by Lemma 1. We consider two cases: $v \neq t_B$ and $v = t_B$ (note that s_B is the parent cut vertex of B , which is handled by the previous case).

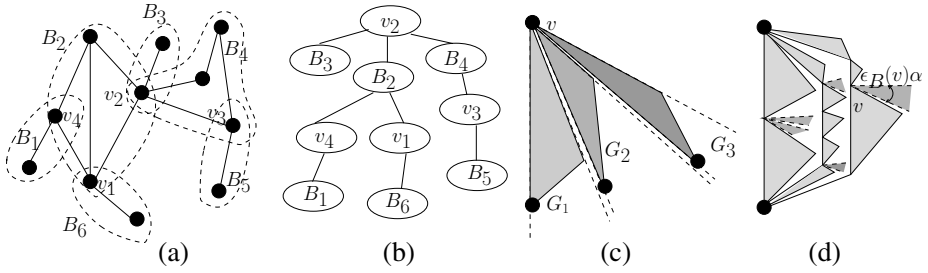


Fig. 3. (a) A 1-connected graph, and (b) corresponding block-cut vertex tree. (c)–(d) Illustration for the proof of Lemma 2.

Assume first that $v \neq t_B$. Then by Lemma 1, there is a free $\epsilon_B(v)\alpha$ -wedge with apex v . Now $\epsilon_B(v)\alpha = (\Delta - \delta_B(v))\alpha > (\delta_G(v) - \delta_B(v))\alpha = \delta_{G_v}(v)\alpha$ and so G_v can be drawn completely inside the free $\epsilon_B(v)\alpha$ -wedge with apex v as shown in Figure 3(d).

If $v = t_B$, then the drawing of G_v already produced can be rotated so that all of the edges in B and of G_v adjacent to t_B use consecutive (clockwise from the line containing s_B and t_B) directions in S_Δ as shown in Figure 3(d). It is an easy observation that the addition of the subgraphs G_v for each cut vertex of B into the free wedge preserves properties (a)–(d) of the drawing of B (and of the drawing of G). \square

Lemma 3. *For any $\Delta > 3$, there exists a 2-connected series-parallel graph G of maximum degree Δ whose planar slope number is at least $\Delta + 1$ if Δ is even and at least Δ if Δ is odd.*

Proof. Consider the graph G obtained from $K_{2,\Delta-1}$ by adding the edge (u, v) connecting the two vertices u and v of degree $\Delta - 1$. Thus G has maximum degree Δ . Now consider any drawing of G . At least half of the remaining $\Delta - 1$ vertices are on one side of the line determined by the segment representing uv in the drawing. Each of these $\lfloor \Delta/2 \rfloor$ vertices forms a triangle with uv and these triangles are nested. Thus no two of the $2\lfloor \Delta/2 \rfloor + 1$ edges in this portion of the drawing of G have the same slope. \square

The following theorem is an immediate consequence of Lemmas 2, and 3.

Theorem 1. *Let G be a partial 2-tree having maximum degree Δ and let $psl(G)$ denote the planar slope number of G . Then $psl(G) \leq 2\Delta$. Also, for every even $\Delta > 3$ there exists a partial 2-tree G such that $psl(G) \geq \Delta + 1$ and for every odd $\Delta \geq 3$ there exists a partial 2-tree G such that $psl(G) \geq \Delta$.*

4 Plane Slope Number of Partial 2-trees

In this section we show that the plane slope number of 2-connected series-parallel graphs, i.e., when the output drawings respect the input embeddings, is at least $3\Delta - 4$ and at most 3Δ . In fact, we show that $S_{1.5\Delta}$ is universal for the family of 2-connected series-parallel graphs with fixed embeddings.

We introduce some additional notation. For an embedded planar 2-connected series-parallel graph G with poles s and t , we call the edge (s, t) (if exists) the *central edge* of G . Since G is 2-connected, the root of its SPQ -tree is a P -node μ . Observe that each of the edges in $skeleton(\mu)$ corresponds to either the edge (s, t) or a *2-connected series-parallel subgraph* of G . If the edge (s, t) exists, then we categorize each of those subgraphs as a *left or right series-parallel subgraph* of G depending on whether it lies to the left or right of the edge (s, t) , while walking from s to t . By G_1^-, \dots, G_l^- , (respectively, G_1^+, \dots, G_r^+) we denote the left (respectively, right) series-parallel subgraphs of G . If (s, t) does not exist, then we assume that all the series-parallel subgraphs are right series-parallel, i.e., G_1^+, \dots, G_r^+ . Furthermore, we assume that the subgraphs are ordered as follows: $G_l^-, \dots, G_1^-, (s, t), G_1^+, \dots, G_r^+$, reflecting their left to right ordering in the embedding.

Lemma 4. *Let \mathcal{G} be the family of plane 2-connected series-parallel graphs of maximum degree at most Δ . Then $S_{1.5\Delta}$ is universal for \mathcal{G} .*

Sketch of Proof: Similar to Lemma 1 we employ an induction on the number of P -nodes in an SPQ -tree T of G . Since the proof follows a similar argument, for reasons of space we sketch here the main idea of the proof. Let $\{s, t\}$ be the split pair associated with the root of T , and let \overline{ab} be a straight line segment with slope m , where $m \in S_{1.5\Delta}$. We show that G admits a drawing Γ using only slopes from $S_{1.5\Delta}$ such that the following properties hold.

- (1) Graph G is drawn within a convex quadrilateral $\square adbc$ having $\angle dac = \delta(s)\alpha$ and $\angle dbc = \delta(t)\alpha$ (i.e. every edge is in the interior of $\square adbc$).
- (2) Vertices s and t are located at a and b , respectively.
- (3) Segment \overline{ab} has slope m .
- (4) The edges incident to s are drawn using consecutive slopes of $S_{1.5\Delta}$, as are the edges incident to t .
- (5) At each vertex $v \notin \{s, t\}$ in the drawing of G , there are two free $\epsilon(v)\alpha$ -wedges at v contained in $\square adbc$, one in the region between the subgraph G_i^+ (or G_i^-) containing v and the previous series-parallel subgraph of G in the ordering, and one in the region between G_i^+ (or G_i^-) and the next series-parallel subgraph of G in the ordering.

Base Case: Assume that T has a single P -node, which must be the root of T since G is 2-connected. We draw s and t on a and b , respectively. The frame of the P -node consists of a set of paths of length at least one (and at most one path of length one). If the central edge exists, then we draw that edge along \overline{ab} (otherwise, we draw the leftmost path along \overline{ab}). We then draw the paths $\Pi_i^+, i = 1, \dots, r$ corresponding to each G_i^+ between a and b using consecutive slopes at s and t , as in the proof of Lemma 1. All the remaining paths $\Pi_i^-, i = 1, \dots, l$ corresponding to G_i^- are drawn symmetrically to the left of segment \overline{ab} , as shown in Figure 4(a). While drawing the paths, we maintain the input embedding. To construct the quadrilateral $\square adbc$, let d be the intersection of the line through a having slope 0.5α plus the slope of the edge of Π_r^+ incident to s with the line through b having slope -0.5α plus the slope of the edge of Π_r^+ incident to t . Similarly, let c be the intersection of the line through a having slope -0.5α plus the

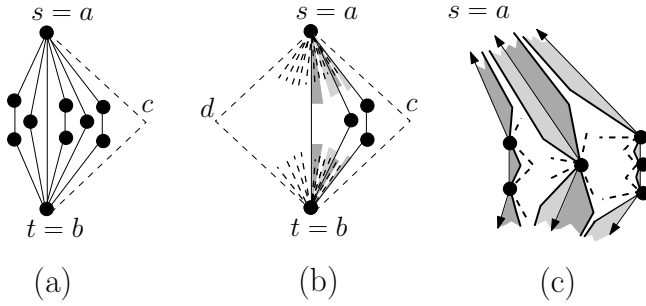


Fig. 4. (a) Base case. (b) Drawing frame, where the light-gray (respectively, dark-gray) regions correspond to the right (respectively, left) series-parallel subgraphs of the corresponding pertinent graph. (c) Recursive construction.

slope of the edge of Π_l^- incident to s with the line through b having slope 0.5α plus the slope of the edge of Π_l^- incident to t . This convex quadrilateral has angle $\delta(s)\alpha$ at a and $\delta(t)\alpha$ at b . See Figure 4 (a) for an example.

It is now straightforward to observe that the resulting drawing is planar and satisfies properties (1)-(4). As for Property (5), consider $v \notin \{s, t\}$ in one of the frame paths and its neighbors u and w on that path. Each of the two angles $\angle uvw$ is either non-acute or at least $\pi - (\delta(u) - 1)\alpha - (\delta(w) - 1)\alpha \geq (\Delta + 2)\alpha \geq \epsilon(v)\alpha$, and so the required empty wedges exist at v .

Induction Step: In a way similar to the proof of Lemma 1, we first draw the frame of the root of the SPQ -tree and then define disjoint convex quadrilaterals for each of the virtual edges. Finally, we recursively compute the drawings of the pertinent graphs inside the corresponding quadrilaterals. The idea is illustrated in Figures 4(b)-(c). \square

By Property (5) of Lemma 4, for every vertex $v \notin \{s, t\}$ in G , there is a free wedge on each side of the drawing of the pertinent subgraph G_i^+ (G_i^-) of G containing v . Thus the arguments used in the proof of Lemma 2 can be directly applied in the fixed embedding case to establish the following lemma.

Lemma 5. *Let \mathcal{G} be the family of plane partial 2-trees of maximum degree at most Δ . Then $S_{1.5\Delta}$ is universal for \mathcal{G} .*

We observe that the $\Delta - 1$ lower bound proved by Knauer et al. [8] for the outerplane slope number implies a lower bound for plane slope number of plane partial 2-trees, because outerplane graphs are plane partial 2-trees. The next lemma shows a better lower bound for partial 2-trees.

Lemma 6. *For every $\Delta \geq 2$, there exists a plane 2-connected series-parallel graph G of maximum degree Δ whose plane slope number is at least $3\Delta - 3$ if Δ is even and at least $3\Delta - 4$ if Δ is odd.*

Proof. Suppose first that $\Delta \geq 2$ is an even number and consider a plane partial 2-tree G defined as follows. The external face of G is a 3-cycle with vertices a, b, c . In its interior there are $\Delta/2 - 1$ paths of length two connecting each pair from $\{a, b, c\}$.

The external face of every plane drawing Γ of G is a triangle $\triangle abc$ that contains the paths of length two in its interior. From elementary geometry, no two edges of Γ can have common slope, and hence the graph, which has $3\Delta - 3$ edges, has plane slope number at least $3\Delta - 3$. Suppose now that Δ is odd and let $\Delta' = \Delta + 1$. Construct a graph G' of maximum degree Δ' as described above. Now remove one of those paths of length 2 connecting a and b and one connecting c and b from G' . This new graph, G , has maximum degree $\Delta = \Delta' - 1$ and it requires a different slope for each edge. Since we deleted four edges from G' , G has $(3\Delta' - 3) - 4 = 3\Delta - 4$ edges. Thus $3\Delta - 4$ slopes are required, and the result is established. \square

Lemmas 4 and 6 imply the following.

Theorem 2. *Let G be a plane partial 2-tree having maximum degree Δ and let $\text{epsl}(G)$ denote the plane slope number of G . Then $\text{epsl}(G) \leq 3\Delta$. Also, for every even $\Delta > 3$ there exists a plane partial 2-tree G such that $\text{epsl}(G) \geq 3\Delta - 3$ and for every odd $\Delta \geq 3$ there exists a plane partial 2-trees G such that $\text{epsl}(G) \geq 3\Delta - 4$.*

5 Angular Resolution

The *angular resolution* of a planar straight-line drawing is the minimum angle between any two edges incident to a common vertex. The angular resolution of a planar graph G is the maximum angular resolution over all possible drawings of G .

Malitz and Papakostas [11] show that the angular resolution of a planar graph of maximum degree Δ is $\Omega(\frac{1}{7\Delta})$. Garg and Tamassia [3] show that there exist planar 3-trees of maximum degree Δ that require angular resolution $O(\sqrt{\frac{\log \Delta}{\Delta^3}})$. They also show that for a subfamily of the partial 2-trees of bounded degree, namely the series-parallel graphs of maximum degree at most Δ , the angular resolution is at least $\frac{1}{48\pi\Delta^2}$. Their drawing technique does not apply to plane graphs since it may vary a given combinatorial embedding. Garg and Tamassia leave as open the problem about whether $\Omega(\frac{1}{\Delta^2})$ is a tight lower bound for the angular resolution of the series-parallel graphs.

An implication of the drawing techniques of the previous sections of this paper is that the angular resolution of partial 2-trees (and thus also of series-parallel graphs) is in fact $\Omega(\frac{1}{\Delta})$. Namely, the constructions of Lemmas 2 and 4 either use the universal set S_Δ or the universal set $S_{1.5\Delta}$ which consist of equally spaced slopes; therefore, the minimum angle between any two edges sharing a common end-vertex is either $\frac{\pi}{2\Delta}$ in the variable embedding setting or it is $\frac{\pi}{3\Delta}$ in the fixed embedding setting. Therefore:

Theorem 3. *A partial 2-tree of maximum degree Δ admits a planar straight-line drawing with angular resolution $\frac{\pi}{2\Delta}$. A plane partial 2-tree of maximum degree Δ admits a planar straight-line drawing with angular resolution $\frac{\pi}{2\Delta}$.*

6 Open Problems

An interesting research direction is to study the trade-off between the slope number and the area requirement of planar graphs. Similar studies have been carried out for the

angular resolution and the area requirements of planar graphs having maximum degree at most Δ (see, e.g., [3]).

Another fascinating open problem is to close the gap between upper and lower bounds on the planar/plane slope number of planar/plane graphs of bounded degree (see [7]). This would be interesting even restricted to partial 3-trees (see [5]).

Acknowledgments. Part of this research took place during the 12th INRIA-McGill-Victoria Workshop on Computational Geometry, held in February 2–8, 2013, at the Bellairs Research Institute of McGill University. We thank the organizers and the participants for the opportunity and the useful discussions. Special thanks go to Zahed Rahmati for early thoughtful insights on the problems of this paper.

References

1. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing. Prentice-Hall, Englewood Cliffs (1999)
2. Dujmović, V., Eppstein, D., Suderman, M., Wood, D.R.: Drawings of planar graphs with few slopes and segments. *Computational Geometry* 38(3), 194–212 (2007)
3. Garg, A., Tamassia, R.: Planar drawings and angular resolution: Algorithms and bounds. In: van Leeuwen, J. (ed.) *ESA 1994*. LNCS, vol. 855, pp. 12–23. Springer, Heidelberg (1994)
4. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) *GD 2000*. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
5. Jelínek, V., Jelínková, E., Kratochvíl, J., Lidický, B., Tesar, M., Vyskocil, T.: The planar slope number of planar partial 3-trees of bounded degree. *Graphs and Combinatorics* 29(4), 981–1005 (2013)
6. Kant, G.: Hexagonal grid drawings. In: Mayr, E.W. (ed.) *WG 1992*. LNCS, vol. 657, pp. 263–276. Springer, Heidelberg (1993)
7. Keszegh, B., Pach, J., Pálvölgyi, D.: Drawing planar graphs of bounded degree with few slopes. *SIAM J. Discrete Math.* 27(2), 1171–1183 (2013)
8. Knauer, K., Micek, P., Walczak, B.: Outerplanar graph drawings with few slopes. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) *COCOON 2012*. LNCS, vol. 7434, pp. 323–334. Springer, Heidelberg (2012)
9. Mondal, D., Nishat, R.I., Biswas, S., Rahman, M.S.: Minimum-segment convex drawings of 3-connected cubic plane graphs. *Journal of Combinatorial Optimization* 25(3), 460–480 (2013)
10. Mukkamala, P., Pálvölgyi, D.: Drawing cubic graphs with the four basic slopes. In: Speckmann, B. (ed.) *GD 2011*. LNCS, vol. 7034, pp. 254–265. Springer, Heidelberg (2011)
11. Papakostas, A., Tollis, I.G.: Improved algorithms and bounds for orthogonal drawings. In: Tamassia, R., Tollis, I.G. (eds.) *GD 1994*. LNCS, vol. 894, pp. 40–51. Springer, Heidelberg (1995)

Slanted Orthogonal Drawings^{*}

Michael A. Bekos¹, Michael Kaufmann¹, Robert Krug¹,
Stefan Näher², and Vincenzo Roselli³

¹ Institute for Informatics, University of Tübingen, Germany
{bekos,mk,krug}@informatik.uni-tuebingen.de

² Institute for Computer Science, University of Trier, Germany
naeher@uni-trier.de

³ Engineering Department, Roma Tre University, Italy
roselli@dia.uniroma3.it

Abstract. We introduce a new model that we call *slanted orthogonal graph drawing*. While in traditional orthogonal drawings each edge is made of axis-aligned line-segments, in slanted orthogonal drawings intermediate diagonal segments on the edges are also permitted, which allows for: (a) smoothening the bends of the produced drawing (as they are replaced by pairs of “half-bends”), and, (b) emphasizing the crossings of the drawing (as they always appear at the intersection of two diagonal segments). We present an approach to compute bend-optimal slanted orthogonal representations, an efficient heuristic to compute close-to-optimal drawings in terms of the total number of bends using quadratic area, and a corresponding LP formulation, when insisting on bend optimality. On the negative side, we show that bend-optimal slanted orthogonal drawings may require exponential area.

1 Introduction

In this paper, we introduce and study a new model in the context of non-planar orthogonal graph drawing: Given a graph G of max-degree 4, determine a drawing Γ of G in which (a) each vertex occupies a point on the integer grid and has four available ports, as in the ordinary orthogonal model, (b) each edge is drawn as a sequence of horizontal, vertical and diagonal segments, (c) a diagonal segment is never incident to a vertex (due to port constraints mentioned above), (d) crossings always involve diagonal segments, and, (e) the minimum of the angles formed by two consecutive segments of any edge always is 135° . We refer to Γ as the *slanted orthogonal drawing* of G , or, shortly, *slog drawing*. Figs.1(a) and 1(b) indicate what we might expect from the new model: crossings on the diagonals are more visible than in the traditional model and the use of area seems to be more effective.

^{*} Part of the research was conducted in the framework of ESF project 10-EuroGIGA-OP-003 GraDR “Graph Drawings and Representations”. The work of M.A. Bekos is implemented within the framework of the Action “Supporting Postdoctoral Researchers” of the Operational Program “Education and Lifelong Learning” (Action’s Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

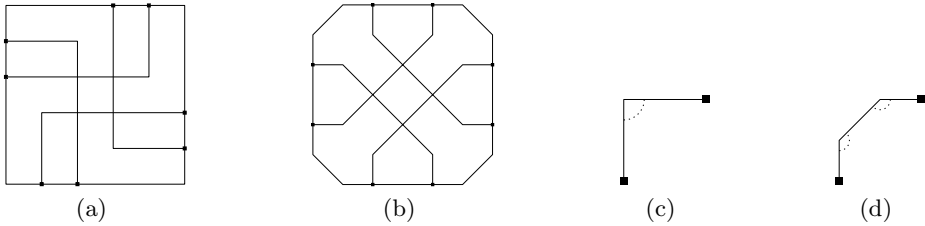


Fig. 1. (a)-(b) Traditional orthogonal and slanted orthogonal drawings of the same graph, assuming fixed ports. (c)-(d) Replacing a 90° bend by two half-bends of 135° .

Orthogonal graph drawing dates back to VLSI layouts and floor-planning applications [6,9,10]. In an *orthogonal drawing* of a graph of max-degree 4, each edge is drawn as a sequence of alternating horizontal and vertical line segments. Typical optimization functions include minimizing the used area [9], the total number of bends [4,8] or the maximum number of bends per edge [1].

Slog drawings are of improved readability and more aesthetic appeal than orthogonal drawings, since bends, which negatively affect the quality of orthogonal drawings are replaced by half-bends that have a smoother shape (see Figs.1(c) and 1(d)). In addition, slog drawings reveal crossings and help distinguishing them from vertices, since crossings are defined by diagonal segments, while vertices are incident to rectilinear segments. Our model resembles an octilinear model as it is heavily used for example in the drawing of Metro Maps [7] but it is closer to the traditional orthogonal style. In particular angles of 45° do not occur at all. So, the complexity results for the octilinear models do not apply.

For minimizing the total number of bends in orthogonal graph drawing Tamassia laid important foundations by the *topology-shape-metrics (TSM)* approach in [8], that works in three phases. In the first *planarization* phase a “planar” embedding is computed for a given (non)planar graph by replacing edge crossings by dummy vertices (referred to as *crossing or c-vertices*). The output is called *planar representation*. In the next *orthogonalization* phase, angles and bends of the drawing are computed, producing an *orthogonal representation*. In the third *compaction* phase the coordinates for vertices and edges are computed. The core is a min-cost flow algorithm to minimize the number of bends in the second phase [2]. We will adopt the TSM approach for our model. Note that the general problem of determining a planar embedding with the minimum number of bends is NP-hard [5], which is also the case for slog drawings. Therefore we assume in the following that a planar representation of the input graph is given.

While constructing the slog drawing we observe the following requirements: (a) all non-dummy vertices (referred to as *real or r-vertices*) use orthogonal ports and, (b) all c-vertices use diagonal ports. This ensures that the computed drawing will be a valid slog drawing that corresponds to the initial planar representation. Edges connecting real (crossing) vertices are referred to as rr-edges (cc-edges), and edges between r- and c-vertices as rc-edges.

This paper is structured as follows: In Section 2 we present an approach to compute bend-optimal slog representations. Afterwards, we present a heuristic to compute close-to-optimal slog drawings, that require polynomial drawing area, based on a given slog representation. To compute the optimal drawing, we give a formulation as a linear program in Section 4. In Section 5 we show that the optimal drawing may require exponential area. We conclude in Section 6.

2 A Flow-Based Approach

In this section, we present a modification of an algorithm by Tamassia [8], which we briefly describe in Section 2.1. Section 2.2 explains our modification and in Section 2.3, we present properties of a bend-minimal slog representation.

2.1 Preliminaries

A central notion to the algorithm of Tamassia [8] is the *orthogonal representation*, that captures the “shape” of the drawing without the exact geometry. An orthogonal representation of a plane graph $G = (V, E)$ is an assignment of four labels to each edge $(u, v) \in E$; two for each direction. Label $\alpha(u, v) \cdot 90^\circ$ corresponds to the angle at vertex u formed by (u, v) and its counterclockwise next incident edge. Label $\beta(u, v)$ corresponds to the number of left turns along (u, v) , when traversing it from u to v . Clearly, $1 \leq \alpha(u, v) \leq 4$ and $\beta(u, v) \geq 0$. The sum of angles around a vertex equals to 360° , so for each vertex $u \in V$, $\sum_{(u,v) \in N(u)} \alpha(u, v) = 4$, where $N(u)$ denotes the neighbors of u . Similarly, since the sum of the angles formed at the vertices and at the bends of a bounded face f equals to $180^\circ(p(f) - 2)$, where $p(f)$ denotes the number of such angles, it follows that $\sum_{(u,v) \in E(f)} \alpha(u, v) + \beta(v, u) - \beta(u, v) = 2a(f) - 4$, where $a(f)$ denotes the number of vertex angles in f , and, $E(f)$ the directed arcs of f in its counterclockwise traversal. If f is unbounded, the sum is increased by 8.

In the flow network one can think of each unit of flow as a 90° angle. The vertices (vertex-nodes; sources) supply 4 units of flow, and each face (face-nodes; sinks) f demands $2a(f) - 4$ units of flow (plus 8 if f is unbounded). To maintain the properties described above each edge from a vertex-node to a face-node in the flow network has a capacity of 4 and a minimum flow of 1, while an edge between adjacent faces has infinite capacity, no lower bound but each unit of flow through it costs one unit. The total cost is actually the number of bends along the corresponding edge. Hence, the min-cost flow solution corresponds to a representation with the minimum number of bends.

2.2 Modifying the Flow Network

We now modify the algorithm of Tamassia, to obtain a slog representation of a planarized graph G with minimum number of half-bends. Recall that G contains two types of vertices, namely real and crossing vertices. Real (crossing) vertices use orthogonal (diagonal) ports. Observe that a pair of half-bends on an rr- or

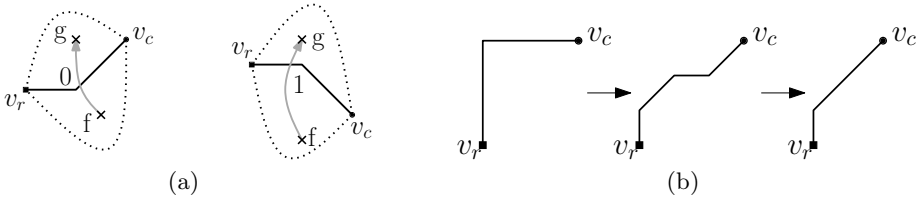


Fig. 2. (a) Two configurations corresponding to zero or one unit of flow over an rc-edge (with f and g being the two adjacent faces) (b) The right rotation of a c-vertex v_c can save a half-bend.

cc-edge of a slog drawing corresponds to a bend of an orthogonal drawing. But a rc-edge changes from an orthogonal port (incident to the r-vertex) to a diagonal port (incident to the c-vertex), requiring at least one half-bend. Consider an rc-edge (v_r, v_c) incident to faces f and g (see Fig.2(a)) and assume that the port of the real vertex v_r is fixed. Depending on the rotation of the crossing vertex v_c (clockwise or counterclockwise) we obtain two different representations with the same number of bends. To model this “free-of-cost” choice, we introduce an edge into the flow network connecting f and g with unit capacity and zero cost. For consistency we assume that, if in the solution of the min-cost flow problem there is no flow over (f, g) , then there exists a left turn from the real to the crossing vertex; otherwise a right turn, as illustrated in Fig.2(a).

2.3 Properties of Optimal Slanted Orthogonal Representations

In this section we give properties of optimal slog representations. We prove that, for a planarized graph G , the computation of a slog representation with minimum number of half-bends respecting the embedding of G is always feasible. Then, we give an upper bound on the number of half-bends in optimal slog representations.

Theorem 1. *For a planarized graph G with max-degree 4, we can efficiently compute a slog representation with minimum number of half-bends respecting the embedding of G .*

Proof. We use a reduction to Tamassia’s network flow algorithm. In particular, since the original flow network computes a (bend-minimal) orthogonal representation for the input plane graph, we will also obtain a slog representation with our modification. We now prove that this representation is also bend-minimal.

Assume that we are given an orthogonal representation F . We can uniquely convert F into a slog representation $S(F)$ by turning all crossing vertices counterclockwise by 45° . More precisely, the last segment of every rc-edge before the crossing vertex will become a left half-bend. Furthermore, every orthogonal bend is converted into two half-bends, bending in the same direction as the orthogonal bend (see Fig.1(c) and 1(d)). Note that the left half-bends at the crossings might neutralize with one of the half-bends originating from an orthogonal bend, if the orthogonal bend is turning to the right (see Fig.2(b)). In this case, only the second one of the right half-bends remains. Note that this is the only possible

saving operation. Therefore, since the number of rc-edges is fixed from the given embedding, a slog representation with minimum number of half-bends should minimize the difference between the number of orthogonal bends of F and the number of first right-bends on rc-edges. However, this is exactly what is done by our min-cost flow network formulation, as the objective is the minimization of the total number of bends in F without the first right-bends on rc-edges. \square

This constructive approach can be reversed so that for each slog representation S , we can get a unique orthogonal representation $F(S)$. Clearly, $F(S(F)) = F$ and $S(F(S)) = S$. Note that this is true only for bend-minimal representations; otherwise we might have staircases of bends which is impossible by min-cost flow computations. From the construction, we can also derive the following.

Corollary 1. *Let $S(F)$ be a slog representation, and F a corresponding orthogonal representation. Let b_S , rb_S and rc_S be the number of half-bends, the number of first right-bends on rc-edges and the number of rc-edges in $S(F)$. Let also b_F be the number of orthogonal bends in F . Then, $b_S = 2 \cdot (b_F - rb_S) + rc_S$.*

The following theorem gives an upper bound for the number of half-bends in optimal slog representations.

Theorem 2. *The number of half-bends of a bend-minimal slog representation is at least twice the number of bends of its related bend-minimal orthogonal representation.*

Proof. Bends of a bend-minimal orthogonal representation correspond to pairs of half-bends on cc and rr edges of a bend-minimal slog representation. In this case, the claim holds with equality. For rc-edges we need a different argument. Let \mathcal{C} be a maximal component spanned by cc-edges. Then, all edges with exactly one endpoint in \mathcal{C} are rc-edges, which can be split into independent cycles around components of crossings. Let C be such a cycle of length k . Clearly, there should be k first half-bends on the rc-edges in the slanted representation. In the corresponding orthogonal representation, the second and third bend of each rc-edge correspond to pairs of half-bends on the same edge in the slog representation. Similarly, in the orthogonal representation the first orthogonal left-bend of each rc-edge corresponds to the second and third left half-bend of the same edge in the slog representation. The only bends that have not been paired (i.e., have no correspondence) are the first right-bends on rc-edges. We claim that in any bend-minimal orthogonal representation, there exist at most $\frac{k}{2}$ first right bends on the edges of C . Assume to the contrary, that in a bend-minimal orthogonal representation, there exist $r > \frac{k}{2}$ first right-bends on the edges of C . If we send the flow along C in reverse direction, we decrease the number of right-bends by r and increase the number of left bends by $k - r$. So, the total number of bends decreases, which shows that the input orthogonal representation was not minimal. From the claim, it follows that the number of first right-bends in the orthogonal representation is at most half of the number of first half-bends of C (in the corresponding slog representation), which concludes the proof since all other half-bends come in pairs and have their correspondences. \square

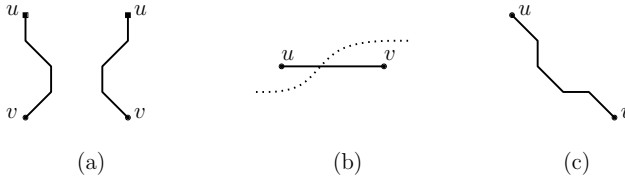


Fig. 3. (a) Spoon gadget for rc-edges. (b) Moving everything above the dotted cut up transforms the orthogonal input to a slanted drawing (c) containing 4 half-bends.

3 A Heuristic to Compute a Close to Optimal Drawing

In this section we present a heuristic which, given an optimal slog representation, computes an actual drawing, which is close to optimal, with respect to the total number of bends, and requires quadratic area. This is quite reasonable, since as we will see in Section 5, insisting in optimal slog drawing may result in exponential area. The basic steps of our approach are given in Algorithm 1. In the following, we describe them in detail.

Algorithm 1. Spoon Based Algorithm

Input: A slanted orthogonal representation F of a given planarized graph G

Output: A slanted orthogonal drawing $\Gamma_s(G)$

- S1: Compute an orthogonal drawing $\Gamma_o(G)$ based on F
 - S2: Replace each orthogonal bend by 2 half-bends {see Figs.1(c) and 1(d)}
 - S3: Fix ports on rc-edges by inserting the spoon gadget {see Fig.3(a)}
 - S4: Apply cuts to fix ports on cc-edges {see Figs.3(b) and 3(c)}
 - S5: Optimize number of rc half-bends {see Fig.4(a)}
 - S6: Optimize number of cc half-bends {see Fig.4(b)}
 - S7: Compact drawing
-

In step 1 of Algorithm 1 we compute a bend-minimal orthogonal drawing $\Gamma_o(G)$ from the slog representation. We use the original algorithm of Tamassia [8], ignoring the flow on the additional edges and the rotation of the crossing vertices. In step 2 of Algorithm 1, we replace all orthogonal bends with pairs of half-bends. In step 3 of Algorithm 1, we connect r-vertices with c-vertices by replacing the segment incident to the c-vertex by a gadget called *spoon*, due to its shape (see Fig.3(a)). It allows switching between orthogonal and diagonal ports on an edge. Note that the slog representation specifies how each c-vertex is rotated, thereby defining the configuration it uses.

In step 4 of Algorithm 1, we employ *cuts* (i.e., is a standard technique to perform layout stretching in orthogonal graph drawing; see [3]) to fix the ports of cc-edges, which still use orthogonal ports. To apply a horizontal (vertical) cut, we have to ensure that each edge crossed by the cut has at least one horizontal

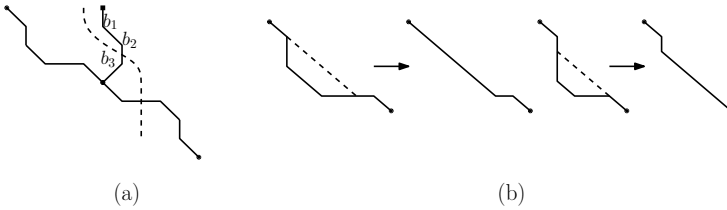


Fig. 4. (a) There is always a cut like the dashed line enabling us to move everything on its left side to the left to save half-bends b_1 and b_2 . (b) Saving bends on cc-edges.

(vertical) segment. This trivially holds before the introduction of the spoons, as $\Gamma_o(G)$ is an orthogonal drawing. It also holds afterwards since a spoon replacing a horizontal (vertical) segment has two horizontal (vertical) segments. To fix a horizontal cc-edge (u, v) with u being to the left of v in the drawing, we use a “horizontal cut” which from left to right and up to vertex u either (a) lies exactly above u , then crosses edge (u, v) and stays exactly below v , or, (b) lies exactly below u , then crosses edge (u, v) and stays exactly above v (see Fig.3(b)). Our choice depends on the slog representation that specifies the rotation of each c-vertex. The result is depicted in Fig.3(c). Observe that the edge has now a horizontal and a vertical segment. Hence, we can fix all remaining cc-edges. To cope with cc-edges with bends we apply the same technique only to the first and last segments of the edge.

The resulting drawing has 2 additional half-bends on rc-edges (the spoon gadget adds 3 half-bends; one is required) and 4 additional half-bends on cc-edges (none is required), with respect to the ones suggested by the representation. By applying cuts again, we can save 2 half-bends for each rc-edge (see Fig.4(a)), by eliminating the diagonal segment of the spoon gadget (step 5 of Algorithm 1). The rectilinear segments of the edge are not affected, to be able to apply future cuts.

It is always possible to remove two of the half-bends on cc-edges (step 6 of Algorithm 1) by a local modification as depicted in Fig.4(b). If the horizontal part of a cc-edge is longer than the vertical one, a shortcut as in the left part of Fig.4(b) can be applied. If the horizontal and the vertical segments of the cc-edge have the same length all four half-bends can be saved.

After applying this operations, the drawing will contain zero additional half-bends on rr-edges and at most two additional half-bends on cc-edges, with respect to the input representation. Note that to apply our technique we need to scale up the initial drawing by a factor of 4 at the beginning, to provide enough space for additional half-bends. In subsequent steps, we increase the drawing area by cuts. However, we can reduce it by contracting along horizontal and vertical cuts at the end (step 7 of Algorithm 1). After the compaction, each horizontal and vertical grid line will be occupied by at least a half-bend, an edge or a vertex, and since all of those are linear in number, the required area of the final slanted drawing is $O(n^2)$. The following theorem summarizes our approach.

Theorem 3. *Given a slog representation of a planarized graph G with max-degree 4, we can efficiently compute a slog drawing requiring $O(n^2)$ area with (i) optimal number of half-bends on rr edges and rc edges without bends and (ii) at most two additional half-bends on cc edges and rc edges with bends.*

4 A Linear Program to Compute Optimal Drawings

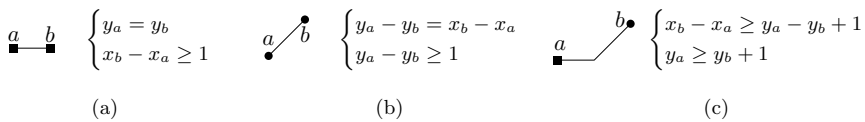
This section describes how to model a given slog representation \mathcal{S} of a plane graph G as a Linear Program (LP). Based on \mathcal{S} , we modify G and obtain a graph G' , that is a subdivision of G and has at most one half-bend on each edge.

Let $\langle b_1, \dots, b_k \rangle$, $k \geq 2$, be the half-bends of edge (u, w) of G in \mathcal{S} , appearing in this order along (u, w) from u to w . Say that u is an r -vertex. We add a new c -vertex v and replace (u, w) by edges (u, v) and (v, w) . The first half-bend b_1 is assigned to (u, v) and $\langle b_2, \dots, b_k \rangle$ to (v, w) . If u is a c -vertex, then v would have been an r -vertex. Observe that the type of v and its ports are defined by the slope of the segments incident to b_1 in \mathcal{S} . By repeating this procedure, we obtain G' , that is a subdivision of G having at most one half-bend on each edge.

Each face f of G has a corresponding face f' in G' such that: (i) the vertices of G' incident to f' are the same as those incident to f in G , plus the ones from the subdivision; and (ii) the sequence of slopes assigned to the segments bounding f' is the same as that of the segments bounding f . So a drawing I' of G' realizing the slog representation is also a drawing of G realizing \mathcal{S} .

For each vertex v of G' we define two variables x_v and y_v , representing its coordinates on the plane. For each edge (a, b) of G' , we define a pair of constraints similar to those in Table 1, depending on the type of vertices of a and b . The table provides an example for each type, the other configurations are analogous.

Table 1. Examples of constraints of the linear program for (a) rr -edges, (b) cc -edges and (c) rc -edges, assuming the y -axis points downwards.



We indirectly minimize the area of the produced drawing by minimizing the total edge length in the objective function. The slopes of the segments allow us to express the Euclidean length of each edge.

Despite the fact that every experiment we made on random and crafted graphs led to a feasible solution, we could not prove the feasibility of the LP. Nevertheless we believe that our LP always admits a feasible solution.

4.1 Addressing Planarity Issues in the LP

The LP models the shape of the edges and the relative positions of all nodes connected by an edge. Since there are no constraints for non-connected nodes

the resulting drawing could be non-planar. An example is given in Fig.5 where the relative position of nodes r_3 and c_2 is not defined by the LP. To solve this problem, we cannot apply the approach used in the original Tamassia algorithm (cutting all faces into rectangles), since our faces are, in general, not rectilinear.

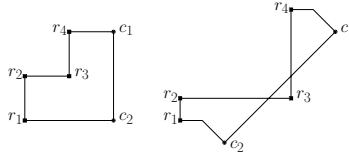


Fig. 5. A configuration that could result in a non-planar solution

In slog drawings we distinguish different corner types. There are *vertex-corners* (or simply vertices) and *bend-corners* (or simply bends). A corner is either *convex* with respect to a face, if the inner angle is $\leq 135^\circ$, or *non-convex* otherwise. An angle of 180° at a vertex is not a corner, since it will be aligned with its neighbors by construction. This gives four possible corners: (non-)/convex vertex(or bend). To ensure planarity, we use *split-edges* and the notion of *almost-convex* faces.

Definition 1. A split-edge is an edge that connects either (a) a non-convex vertex-corner v with a new vertex that subdivides a side parallel to one of the edges incident to v , or, (b) two new vertices that subdivide two parallel edges, when one of them is incident to a non-convex bend-corner (see Figs.6(a) and 6(b)).

Definition 2. A face f is almost-convex if it does not contain any non-convex vertex-corners and no split-edge exists that separates f into two non-convex faces.

First we make all faces almost-convex. To remove non-convex vertex-corners we introduce a new split-edge; see Fig.6(a). If there is no parallel side to one of the segments of a vertex-corner we use a special structure, which we call *nose-gadget* (due to its shape); see Fig.6(c). The dashed line in the figure represents the split-edge we can apply once the gadget (dotted line) is added. The two vertices that are added on the diagonals are c -vertices, while the third one is an r -vertex. By applying the split edge the non-convex vertex corner is removed.

After the previous step no face contains non-convex vertex-corners. If a face contains non-convex bends and is not almost-convex, we search for a split-edge that creates two non-convex faces (as in Fig.6(b)). We apply such split-edges until all faces are almost-convex. Observe that all additional edges can be expressed by using the original set of constraints from the LP. To prove that we can always make all faces almost-convex, we give the following lemmas.

Lemma 1. If a face contains two segments s_1 and s_2 defining a non-convex bend-corner, then it contains a segment s_3 parallel to either s_1 or s_2 .

Sketch of Proof. Assume to the contrary that there is no segment parallel to s_1 and s_2 . Let b be the non-convex bend, and, r and c its adjacent corners;

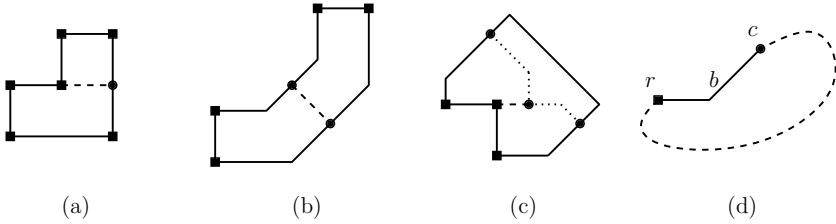


Fig. 6. Split-edges are dashed lines on (a) a vertex and (b) a bend. (c) If no split-edge is possible the nose gadget (dotted line) is used. Circles are additional nodes. (d) This face can not be completed without using at least one segment parallel to (r, b) or (b, c) .

see Fig.6(d). Then, r and c cannot be connected by a polygonal chain of edges without using one of the two slopes defined by (r, b) and (b, c) .

Lemma 2. *An almost-convex face f has at most two consecutive non-convex bend-corners.*

Proof. Assume that f has three consecutive non-convex corners c_1, c_2, c_3 . By Lemma 1, there exists a parallel segment to one of those defining c_2 . Then, there exists a split-edge separating f into two non-convex faces, one containing c_1 and one containing c_3 , which is a contradiction. \square

Lemma 3. *An almost-convex face f has at most two non-convex bend-corners.*

Sketch of Proof. Assuming that f contains at least three non-convex corners, one can lead to a contradiction the fact that f is almost-convex.

Lemma 4. *An almost-convex face f is always drawn planar.*

Sketch of Proof. From Lemma 3, it follows that f has at most two non-convex bend-corners. If f contains no or a single non-convex bend-corner, then it is easy to see that f is always drawn planar. If f contains exactly two non-convex bend-corners, then by Lemma 2 f cannot have more than two consecutive non-convex bend-corners. Hence, there exist in total four cases that one has to consider with respect to the shape of f . In all of them f is drawn planar.

With these lemmas we have shown how to subdivide all faces of a graph $G = (V, E)$ to obtain a graph $G' = (V', E')$ that only contains almost-convex faces. For G' we can compute a planar drawing using our linear program, giving a planar drawing for G . We ensured the planarity of the drawing by adding N new vertices and edges, where N is $O(|E|)$, since there is at most one split-edge for each vertex and for each non-convex bend. All additional edges can be modeled by the original set of constraints. Experiments on random and crafted graphs seem to confirm that our linear program always has a feasible solution.

5 Area Bounds

Slog drawings have aesthetic appeal and improve the readability of non-planar graphs, when compared to traditional orthogonal drawings. However, in this section we show that such drawings might require increased drawing area. Note that most of the orthogonal drawing algorithms require $O(n) \times O(n)$ area. The situation is different if we want to generate slog drawings of optimal number of bends. In particular, we show that the area penalty can be exponential.

Theorem 4. *There exists a graph G whose slanted orthogonal drawing $\Gamma(G)$ of minimum number of bends requires exponential area, assuming that a planarized version $\sigma(G)$ of the resulting drawing is given.*

Proof. The planarized version $\sigma(G)$ of G is given in Fig.7(a) and consists of $n+1$ layers L_0, L_1, \dots, L_n . Layer L_0 is the square grid graph on 9 vertices. Each layer $L_i, i = 1, 2, \dots, n$, is a cycle on 20 vertices. Consecutive layers L_{i-1} and $L_i, i = 1, 2, \dots, n$, are connected by 12 edges which define 12 crossings. Hence, G consists of $20n + 9$ vertices and $32n + 13$ edges that define $12n$ crossings.

A slog drawing $\Gamma(G)$ of G with minimum number of bends derived from $\sigma(G)$ ideally introduces (a) no bends on crossing-free edges of $\sigma(G)$, and, (b) two half-bends in total for each rc-edge. Now observe that at each layer there exist four vertices, that have two ports pointing to the next layer (gray-colored in Fig.7(a)). This together with requirements (a) and (b) suggests that the vertices of each layer L_i should reside along the edges of a rectangle, say R_i , such that the vertices of L_i whose ports point to the next layer coincide with the corners of $R_i, i = 0, 1, 2, \dots, n$ (with the only exception of the “innermost” vertex of L_0 ; in Fig.7(b), R_i is identified with cycle L_i). Hence, the routing of the edges that connect consecutive layers should be done as illustrated in Fig.7(b). Since L_0 is always drawable in a 3×3 box meeting all requirements mentioned above, and, $\sigma(G)$ is highly symmetric, we can assume that each R_i is a square of side length $w_i, i = 0, 1, 2, \dots, n$. Then, it is not difficult to see that $w_0 = 3$ and $w_{i+1} = 2w_i + 8, i = 1, 2, \dots, n$. This implies that the area of $\Gamma(G)$ is exponential in the number of layers of G and therefore exponential in the number of vertices of G (recall that G has $n + 1$ layers and $20n + 9$ vertices). □

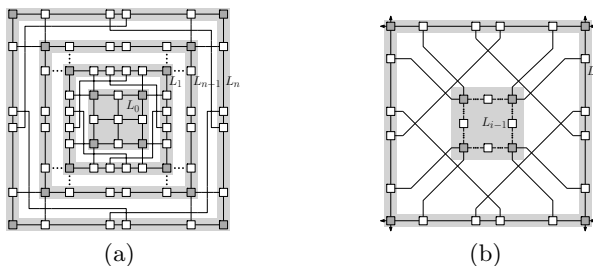


Fig. 7. (a) A planarized version $\sigma(G)$ of a graph G . (b) Edges involved in crossings in $\sigma(G)$ contribute two half-bends.

6 Conclusion and Open Problems

We introduced a new model for drawing graphs of max-degree four, in which orthogonal bends are replaced by pairs of “slanted” bends and crossings occur on diagonal segments only. The main advantage of this model is that, even in drawings of large graphs (where vertices might not be clearly visible), it is immediately clear which pair of edges induce a crossing and where such a crossing is located in the drawing. We presented an algorithm to construct slog drawings with almost-optimal number of bends and quadratic area, for general max-degree four graphs. By a modification of Tamassia’s min-cost flow approach, we showed that a bend-optimal representation of the graph can efficiently be computed in polynomial time and we presented an LP-approach to compute a corresponding drawing. A natural problem is whether every max-degree four graph admits such a drawing. Our experiments on randomly generated and crafted inputs led us to believe that it is possible, although we could not prove it.

References

1. Biedl, T.C., Kant, G.: A better heuristic for orthogonal graph drawings. In: van Leeuwen, J. (ed.) *ESA 1994*. LNCS, vol. 855, pp. 24–35. Springer, Heidelberg (1994)
2. Cornelsen, S., Karrenbauer, A.: Accelerated bend minimization. *Journal of Graph Algorithms Applications* 16(3), 635–650 (2012)
3. Fößmeier, U., Heß, C., Kaufmann, M.: On improving orthogonal drawings: The 4M-algorithm. In: Whitesides, S.H. (ed.) *GD 1998*. LNCS, vol. 1547, pp. 125–137. Springer, Heidelberg (1999)
4. Fößmeier, U., Kaufmann, M.: Drawing high degree graphs with low bend numbers. In: Brandenburg, F.J. (ed.) *GD 1995*. LNCS, vol. 1027, pp. 254–266. Springer, Heidelberg (1996)
5. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal of Computing* 31(2), 601–625 (2001)
6. Leiserson, C.E.: Area-efficient graph layouts (for VLSI). In: *21st Symposium on Foundations of Computer Science*, vol. 1547, pp. 270–281. IEEE (1980)
7. Nöllenburg, M., Wolff, A.: Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Trans. Vis. Comput. Graph.* 17(5), 626–641 (2011)
8. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. *SIAM Journal of Computing* 16(3), 421–444 (1987)
9. Tamassia, R., Tollis, I.G.: Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems* 36(9), 1230–1234 (1989)
10. Valiant, L.G.: Universality considerations in VLSI circuits. *IEEE Transaction on Computers* 30(2), 135–140 (1981)

Drawing Arrangement Graphs in Small Grids, or How to Play Planarity

David Eppstein*

Department of Computer Science, University of California, Irvine, USA

Abstract. We describe a linear-time algorithm that finds a planar drawing of every graph of a simple line or pseudoline arrangement within a grid of area $O(n^{7/6})$. No known input causes our algorithm to use area $\Omega(n^{1+\epsilon})$ for any $\epsilon > 0$; finding such an input would represent significant progress on the famous k -set problem from discrete geometry. Drawing line arrangement graphs is the main task in the *Planarity* puzzle.

1 Introduction

Planarity (<http://planarity.net/>) is a puzzle developed by John Tantalo and Mary Radcliffe in which the user moves the vertices of a planar graph, starting from a tangled circular layout (Figure 1), into a position where its edges (drawn as line segments) do not cross. The game is played in a sequence of levels of increasing difficulty. To construct the graph for the i th level, the game applet chooses $\ell = i + 3$ random lines in general position in the plane. It creates a vertex for each of the $\ell(\ell - 1)/2$ crossings of two lines, and an edge for each of the $\ell(\ell - 2)$ consecutive pairs of crossings on the same line.

One strategy for solving *Planarity* would be to reconstruct a set of lines forming the given graph (Figure 2, left). However, this is tedious to do by hand, and has high computational complexity: testing whether an arrangement of curves is combinatorially equivalent to a line arrangement is NP-hard [1], from which it follows that recognizing line arrangement graphs is also NP-hard [2]. More precisely, both problems are complete for the existential theory of the reals [3]. And although drawings constructed in this way accurately convey the underlying construction of the graph, they have low angular resolution (at most π/ℓ) and close vertex spacing, making them hard to read and hard to place by hand. In practice these puzzles may be solved more easily by an incremental strategy

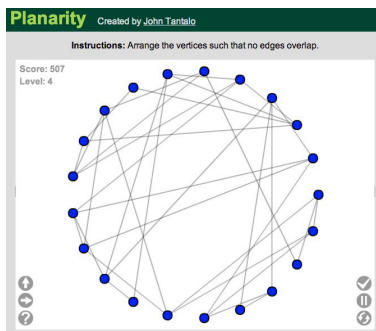


Fig. 1. Initial state of *Planarity*

* This research was supported in part by NSF grants 0830403 and 1217322 and by the Office of Naval Research under grant N00014-08-1-1015.

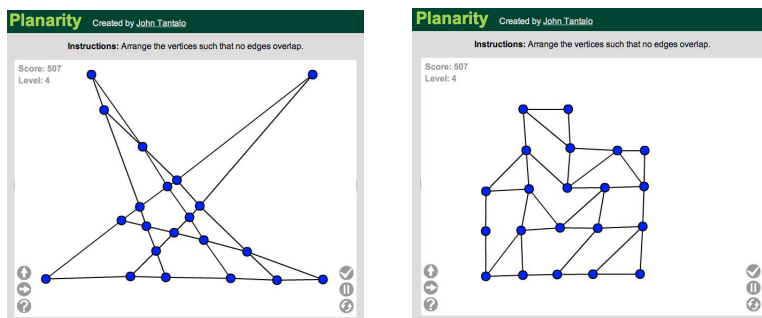


Fig. 2. Two manually constructed solutions to the puzzle from Figure 1. Left: a set of lines with this graph as its arrangement. Right: an (approximate) grid layout.

that maintains a planar embedding of a subgraph of the input, starting from a single short cycle (such as a triangle or quadrilateral), and that at each step extends the embedding by a single face, bounded by a short path connecting two vertices on the boundary of the previous embedding. When using this strategy to solve a Planarity puzzle, the embedding may be kept tidy by placing each vertex into an approximate grid (Figure 2, right). Curiously, the grid drawings found by this incremental grid-placement heuristic appear to have near-linear area; in contrast, there exist planar graphs such as the *nested triangles graph* that cannot be drawn planarly in a grid of less than $\Theta(n^2)$ area [4, 5].

In this paper we explain this empirical finding of small grid area by developing an efficient algorithm for constructing compact grid drawings of the arrangement graphs arising in Planarity. Because recognizing line arrangement graphs is NP-hard, we identify a larger family of planar graphs (the graphs of simple pseudoline arrangements) that may be recognized and decomposed into pseudolines in linear time. We show that every n -vertex simple pseudoline arrangement graph may be drawn in linear time in a grid of size $\kappa_{\max}(O(\sqrt{n})) \times O(\sqrt{n})$; here $\kappa_{\max}(\ell)$ is the maximum complexity of a k -level of a pseudoline arrangement with ℓ pseudolines [6–8], a topological variant of the famous k -set problem from discrete geometry (see Section 3 for a formal definition). The best proven upper bounds of $O(\ell^{4/3})$ on the complexity of k -levels [7–9] imply that the grid in which our algorithm draws these graphs has size $O(n^{2/3}) \times O(\sqrt{n})$ and area $O(n^{7/6})$. However, all known lower bounds on k -level complexity are of the form $\Omega(\ell^{1+o(1)})$ [6, 10], suggesting that our algorithm is likely to perform even better in practice than our worst-case bound. If we could find a constant $\epsilon > 0$ and a family of inputs that would cause our algorithm to use area $\Omega(\ell^{1+\epsilon})$, such a result would represent significant progress on the k -set problem.

We also investigate the construction of *universal point sets* for arrangement graphs, sets of points that can be used as the vertices for a straight-line planar drawing of every n -vertex arrangement graph. Our construction directly provides a universal point set consisting of $O(n^{7/6})$ grid points; we show how to sparsify this structure, leading to the construction of a universal set of $O(n \log n)$ points within a grid whose dimensions are again $O(n^{2/3}) \times O(\sqrt{n})$.

Finally, we formalize and justify an algorithm for manual solution of these puzzles that greedily finds short cycles and adds them as faces to a partial planar embedding. Although this algorithm may fail for general planar graphs, we show that for arrangement graphs it always finds a planar embedding that is combinatorially equivalent to the original arrangement.

2 Preliminaries

Following Shor [1], we define a *pseudoline* to be the image of a line under a homeomorphism of the Euclidean plane. Pseudolines include lines, non-self-crossing polygonal chains starting and ending in infinite rays, and the graphs of continuous real functions. Two pseudolines *cross* at a point x if a neighborhood of x is homeomorphic to a neighborhood of the crossing point of two lines, with the homeomorphism taking the pseudolines to the lines. An *arrangement* of pseudolines is a finite set of pseudolines, the intersection of every two of which is a single crossing point. An arrangement is *simple* if all pairs of pseudolines have distinct crossing points. A *pseudoline arrangement graph* is a planar graph whose vertices are the crossings in a simple pseudoline arrangement, and whose edges connect consecutive crossings on a pseudoline.

Most of the ideas in the following result are from Bose et al. [2], but we elaborate on that paper to show that linear time recognition of arrangement graphs is possible. (See [12] for a more complicated linear time algorithm that recognizes the dual graphs of a wider class of arrangement graphs, the graphs of *weak* pseudoline arrangements in which pairs of pseudolines need not cross)

Lemma 1. *If we are given as input a graph G , then in linear time we can determine whether it is a pseudoline arrangement graph, determine its (unique) embedding as an arrangement graph, and find a pseudoline arrangement for which it is the arrangement graph.*

Proof. Let G^* be formed from a pseudoline arrangement graph G by adding a new vertex v_∞ adjacent to all vertices in G of degree less than four. As Bose et al. [2] show, G^* is 3-connected and planar, and its unique planar embedding is compatible with the embedding of G as an arrangement graph. For convenience we include two edges in G^* from v_∞ to each degree two vertex in G , so that, in G^* , all vertices except v_∞ have degree four. With this modification, the pseudolines of the arrangement for G are represented in G^* by paths starting and ending at v_∞ that, at each other vertex, connect two opposite edges in the embedding.

For any given graph G of maximum degree four we may, in linear time, add a new vertex v_∞ , test planarity of the augmented graph G^* , and embed G^*

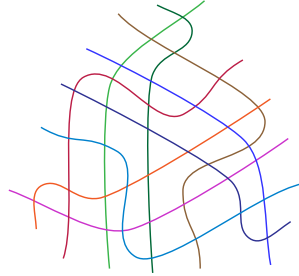


Fig. 3. A simple pseudoline arrangement that cannot be transformed into a line arrangement. Redrawn from Figure 5.3.2 of [11], who attribute this arrangement to Ringel.

in the plane. The edge partition of G^* into paths through opposite edges at each degree four vertex may be found in linear time by connected component analysis. By labeling each edge with the identity of its path, we may verify that this partition does not include cycles disjoint from v_∞ and that no path crosses itself. We additionally check that G has $\ell(\ell-1)/2$ vertices, where ℓ is the number of paths. Finally, by listing the pairs of paths passing through each vertex and bucket sorting this list, we may verify in linear time that no two paths cross more than once. If G passes all of these checks, its decomposition into paths gives a valid pseudoline arrangement, which may be constructed by viewing the embedding of G^* as being on a sphere, puncturing the sphere at point v_∞ , and homeomorphically mapping the punctured sphere to the plane. \square

3 Small Grids

To describe our grid drawing algorithm for pseudoline arrangement graphs, we need to introduce the concept of a *wiring diagram*. A wiring diagram is a particular kind of pseudoline arrangement, in which the ℓ pseudolines largely lie on ℓ horizontal lines (with coordinates $y = 1, y = 2, \dots, y = \ell$). The pseudolines on two adjacent tracks may cross each other, swapping which track they lie on, near points with coordinates $x = 1, x = 2, \dots, x = \ell(\ell - 1)/2$; each crossing is formed by removing two short segments of track and replacing them by two crossing line segments between the tracks. It is convenient to require different crossings to have different x coordinates, following Goodman [13], although some later sources omit this requirement. Figure 4 depicts an example. Wiring diagrams already provide reasonably nice grid drawings of arrangement graphs [14], but are unsuitable for our purposes, for two reasons: they draw the edges connecting pairs of adjacent crossings as polygonal chains with two bends, and for some arrangements, even allowing crossings to share x -coordinates, drawing the wiring diagram of ℓ lines in a grid may require width $\Omega(\ell^2)$ (Figure 5), much larger than our bounds. Instead, we will use these diagrams as a tool for constructing a different and more compact straight-line drawing.

For an arrangement of non-vertical lines in general position, an equivalent wiring diagram may be constructed by a *plane sweep* algorithm [15], which simulates the left-to-right motion of a vertical line across the arrangement. At most

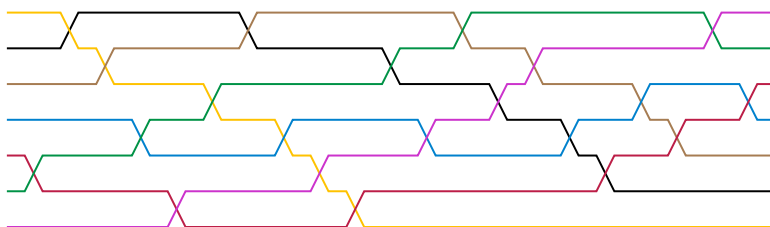


Fig. 4. A wiring diagram formed by a plane sweep of the arrangement from Figure 2

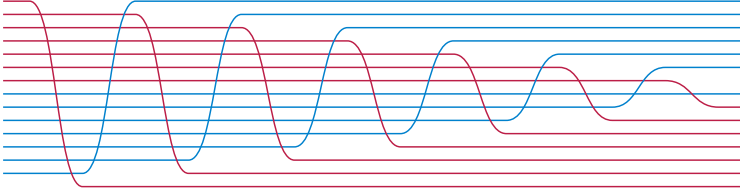


Fig. 5. Cocktail shaker sort corresponds to an arrangement of ℓ pseudolines for which drawing the wiring diagram in a grid requires width $\Omega(\ell^2)$

points in the sweep, the intersection points of the arrangement lines with the sweep line maintain a fixed top-to-bottom order with each other, with their positions in this order reflected in the assignment of the corresponding pseudolines to tracks. When the sweep line crosses a vertex of the arrangement, two intersection points swap positions in the top-to-bottom order, corresponding to a crossing in the wiring diagram. The left-to-right order of crossings in the wiring diagram is thus exactly the sorted order of the crossing points of the arrangement, as sorted by their x coordinates. The wiring diagram in Figure 4 was constructed in this way from the approximate line arrangement depicted in Figure 1.

Every simple pseudoline arrangement, also, has an equivalent wiring diagram, that may be constructed in time linear in its number of crossings. The proof of this fact uses *topological sweeping*, a variant of plane sweeping originally developed to speed up sweeping of straight line arrangements by relaxing the strict left-to-right ordering of the crossing points [16], that can also be extended to apply to pseudoline arrangements [17]. The steps of the topological sweeping algorithm require only determining the relative ordering of crossings along each of the input pseudolines, something that may easily be determined from our path decomposition of a pseudoline arrangement graph by precomputing the position of each crossing on each of the two pseudolines it belongs to.

We define the i th level $L_{\mathcal{D}}(i)$ in a wiring diagram \mathcal{D} to be the set of crossings that occur between tracks i and $i + 1$. A crossing belongs to $L_{\mathcal{D}}(i)$ if and only if $i - 1$ lines pass between it and the bottom face of the arrangement (the face below all of the tracks in the wiring diagram); therefore, once this bottom face is determined, the levels are fixed by this choice regardless of how the crossings are ordered to form a wiring diagram. If we define the size $|\mathcal{D}|$ of a diagram to be its number of pseudolines, and the level complexity $\kappa(\mathcal{D})$ to be $\max_i |L_{\mathcal{D}}(i)|$, then it is a longstanding open problem in discrete geometry (a variant of the k -set problem) to determine the maximum level complexity of an arrangement of ℓ pseudolines, $\kappa_{\max}(\ell) = \max_{|\mathcal{D}|=\ell} \kappa(\mathcal{D})$. (Often this problem is stated in terms of the middle level of an arrangement, rather than the maximum-complexity level, but this variation makes no difference to the asymptotic behavior of the level complexity.) The known bounds on this quantity are $\kappa_{\max}(\ell) = O(\ell^{4/3})$ [7–9], and $\kappa_{\max}(\ell) = \Omega(\ell c^{\sqrt{\log \ell}})$ for some constant $c > 1$ [6, 10], where the last bound is $O(n^{1+\epsilon})$ for all constants $\epsilon > 0$.

Theorem 1. *Let G be a pseudoline arrangement graph with n vertices, determined by $\ell = \Theta(\sqrt{n})$ pseudolines. Then in time $O(n)$ we may construct a planar straight-line drawing of G , in a grid of size $(\ell - 1) \times \kappa_{\max}(\ell) = O(n^{1/2}) \times O(n^{2/3})$.*

Proof. We find a decomposition of G into pseudoline paths, by the algorithm of Lemma 1, and use topological sweeping to convert this decomposition into a wiring diagram. We place each vertex v of G at the coordinates (i, j) , where i is the position of v within its level of the wiring diagram and j is the number of tracks below its level of the wiring diagram.

With this layout, every edge of G either connects consecutive vertices within the same level as each other, or it connects vertices on two consecutive levels. In the latter case, each edge between two consecutive levels corresponds to a horizontal segment of the wiring diagram that lies on the track between the two levels; the left-to-right ordering of these horizontal segments is the same as the left-to-right ordering of both the lower endpoints and the upper endpoints of these edges. Because of this consistent ordering of endpoints, no two edges between the same two consecutive levels can cross. There can also not be any crossings between edges that do not both lie in the same level or connect the same two consecutive levels. Therefore, the drawing we have constructed is planar. By construction, it has the dimensions given in the theorem. \square

Figure 6 depicts the output of our algorithm, using the wiring diagram of Figure 4, for the graph of Figure 1. The arrangement has six levels, with at most five vertices per level, giving a 6×5 grid. Although not as compact as the manually-found 5×5 grid of Figure 2, it is much smaller than standard grid drawings that do not take advantage of the arrangement structure of this graph. A more careful placement of vertices within each row would improve the angular resolution and edge length of the drawing but we have omitted this step in order to make the construction more clear.

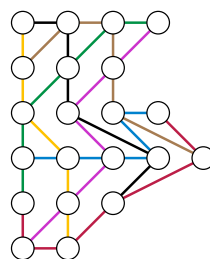


Fig. 6. Output of the drawing algorithm of Theorem 1, based on the wiring diagram of Figure 4

4 Universal Point Sets

A *universal point set* for the n -vertex graphs in a class \mathcal{C} of graphs is a set U_n of points in the plane such that every n -vertex graph in \mathcal{C} can be drawn with its vertices in U_n and with its edges drawn as non-crossing straight line segments [18]. Grids of $O(n) \times O(n)$ points form universal sets of quadratic size for the planar graphs [19, 20], and despite very recent improvements the best upper bound known remains quadratic [21]. A rectangular grid that is universal must have $\Omega(n^2)$ points [4, 5]; the best known lower bounds for universal point sets that are not required to be grids are only linear [18].

Subquadratic bounds are known on the size of universal point sets for subclasses of the planar graphs including the outerplanar graphs [22], simply-nested

planar graphs [21,23], planar 3-trees [24], and graphs of bounded pathwidth [21]; however, these results do not apply to arrangement graphs. The grid drawing technique of Theorem 1 immediately provides a universal point set for arrangement graphs of size $O(n^{7/6})$; in this section we significantly improve this bound, while only increasing the area of our drawings by a constant factor.

Following Bannister et al. [21], define a sequence of positive integers ξ_i for $i = 1, 2, 3, \dots$ by the equation $\xi_i = i \oplus (i - 1)$ where \oplus denotes the bitwise binary exclusive or operation. The sequence of these values begins

$$1, 3, 1, 7, 1, 3, 1, 15, 1, 3, 1, 7, 1, 3, 1, \dots$$

Lemma 2 (Bannister et al. [21]). *Let the finite sequence $\alpha_1, \alpha_2, \dots, \alpha_k$ have sum s . Then there is a subsequence $\beta_1, \beta_2, \dots, \beta_k$ of the first s terms of ξ such that, for all i , $\alpha_i \leq \beta_i$. The sum of the first s terms of ξ is between $s \log_2 s - 2s$ and $s \log_2 s + s$.*

Recall that the grid drawing technique of Theorem 1 produces a drawing in which the vertices are organized into $\ell - 1$ rows of at most $\kappa_{\max}(\ell) = O(\ell^{4/3})$ vertices per row, where $\ell = O(\sqrt{n})$ is the number of lines in the underlying n -vertex arrangement. In this drawing, suppose that there are n_i vertices on the i th row of the drawing, and define a sequence $\alpha_i = \lceil n_i/\ell \rceil$.

Lemma 3. $\sum \alpha_i \leq 3(\ell - 1)/2$.

Proof. We may partition the n_i vertices in the i th row n_i into $\lfloor n_i/\ell \rfloor$ groups of exactly ℓ vertices, together with at most one smaller group; then α_i is the number of groups. The contribution to $\sum \alpha_i$ from the groups of exactly ℓ vertices is at most $n/\ell = (\ell - 1)/2$. There is at most one smaller group per row so the contribution from the smaller groups is at most $\ell - 1$. Thus the total value of the sum is at most $3(\ell - 1)/2$. □

Theorem 2. *There is a universal point set of $O(n \log n)$ points for the n -vertex arrangement graphs, forming a subset of a grid of dimensions $O(\ell) \times \kappa_{\max}(\ell)$.*

Proof. Let $s = 3(\ell - 1)/2$. We form our universal point set as a subset of an $s \times \kappa_{\max}(\ell)$ grid; the area of the grid from which the points are drawn is exactly $3/2$ times the area of the $(\ell - 1)$ -row grid drawing technique of Theorem 1. In the i th row of this grid, we include in our universal point set $\min(\ell \xi_i, \kappa_{\max}(\ell))$ of the grid vertices in that row. It does not matter for our construction exactly which points of the row are chosen to make this number of points.

By Lemma 2, there is a subsequence β_i of the first s rows of sequence ξ , such that the β is termwise greater than or equal to α . This subsequence corresponds to a subsequence $(r_1, r_2, \dots, r_{\ell-1})$ of the rows of our universal point set, such that row r_i has at least $\min(\ell \beta_i, \kappa_{\max}(\ell)) \geq n_i$ points in it. Mapping the i th row of the drawing of Theorem 1 to row r_i of this point set will not create any crossings, because the mapping is monotonic within each row and because all edges of the drawing connect pairs of vertices that are either in the same row or in rows that are consecutive in the selected subsequence.

The number of points in the point set is $O(\ell s \log s)$ where $s = O(\ell)$. Therefore, this number of points is $O(\ell^2 \log \ell) = O(n \log n)$. □

5 Greedy Embedding Algorithm

The algorithm of Lemma 1 uses as a subroutine a linear-time planarity testing algorithm. Although such algorithms may be efficiently implemented on computers, they are not really suitable for hand solution of Planarity puzzles. Instead, it is more effective in practice to build up a planar embedding one face at a time, by repeatedly finding a short cycle in the input graph and attaching it to the previously constructed partial embedding. Here “short” means as short as can be found; it is not possible to limit attention to cycles of length three, four, or any fixed bound. For instance in Figure 3 the central triangle is separated from the rest of the graph by faces with five sides, and by modifying this example it is possible to separate part of an arrangement graph from the rest of the graph by faces with arbitrarily many sides. Thus, this hand-solution heuristic may be formalized by the following steps.

1. Choose an arbitrary starting vertex v .
2. Find a cycle C_1 of minimum possible length containing v .
3. Embed C_1 as a simple cycle in the plane.
4. While some of the edges of the input graph have not yet been embedded:
 - (a) Let C_i be the cycle bounding the current partial embedding. Define an *attachment vertex* of C_i to be a vertex that is incident with edges not already part of the current embedding.
 - (b) Choose two attachment vertices u and v , and a path P_i in C_i from u to v , such that there are no attachment vertices interior to P_i .
 - (c) Find a shortest path S_i from u to v , using only edges that are not already part of the current partial embedding.
 - (d) If necessary, adjust the positions of the embedded vertices (without changing the combinatorial structure of the embedding) so that S_i may be drawn with straight line edges.
 - (e) Add S_i to the embedding, outside C_i , so that the new face between P_i and S_i does not contain C_i . After this change, the new bounding cycle C_{i+1} of the partial embedding is formed from C_i by replacing P_i by S_i .

When it is successful, this algorithm decomposes the input graph into the cycle C_1 and a sequence of edge-disjoint paths S_1, S_2 , etc. Such a decomposition is known as an *open ear decomposition* [25].

This greedy ear decomposition algorithm does not always work for arbitrary planar graphs: even the initial cycle that is found by the algorithm may not be a face of an embedding of the given graph, causing the algorithm to make incorrect assumptions about the structure of the embedding. However (ignoring the possible difficulty of performing step d) the algorithm does always correctly embed the arrangement graphs used by Planarity. These graphs may have multiple embeddings; to distinguish among them, define the *canonical embedding* of an arrangement graph to be the one given by the arrangement from which it was constructed. By Lemma 1, the canonical embedding is unique. As we prove below, the cycles of an arrangement graph that the algorithm assumes to be faces really are faces of the canonical embedding.

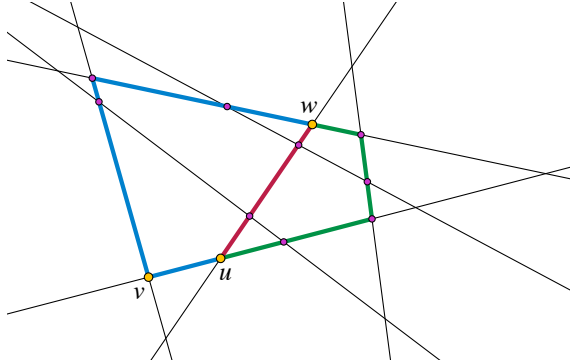


Fig. 7. Illustration for the proof of Lemma 4. Every non-facial cycle C through vertex v (blue and green edges) is crossed by at least one line $\ell = uw$ (red edges), forming a theta-graph. All the vertices on the red path of the theta are matched by an equal number of vertices on each of the other two paths, caused by crossings with the same lines, and the other two paths have additional vertices at their bends, so the red-blue cycle is shorter than the blue-green cycle.

Lemma 4. *Let v be an arbitrary vertex of arrangement graph G , and C be a shortest cycle containing v . Then C is a face of the canonical embedding of G .*

Proof. Let C be an arbitrary simple cycle through v . Then if C is not a face of the arrangement forming G , there is a line ℓ that crosses it; let u and w be two vertices on the boundary of C connected through the interior of C by ℓ (Figure 7). Then C together with the path along ℓ from u to w form a theta-graph, a graph with two degree three vertices (u and w) connected by three paths. Every vertex of ℓ between u and w is caused by a crossing of ℓ with another line that also must cross the other two paths of the theta-graph; in addition, each of these two paths must bend at least once at a vertex that does not correspond to a line that crosses ℓ . Therefore, the path through ℓ is strictly shorter than the other two paths in the theta-graph. Replacing one of the two paths of C from u to w by the path through ℓ produces a shorter cycle that still contains v . Since an arbitrary cycle C that is not a face can be replaced by a shorter cycle through v , it follows that every shortest cycle through v is a face. \square

Lemma 5. *Let D be a drawing of a subset of the faces of the canonical embedding of an arrangement graph G whose union is a topological disk, let u and v be two attachment vertices on the boundary of D with no attachment vertices interior to the boundary path P from u to v , and let S be a shortest path from u to v using only edges not already part of D . Then the cycle formed by the union of P and S is a face of the canonical embedding of G .*

Proof. Assume for a contradiction that $P \cup S$ is not a face; then as in the proof of Lemma 4, this cycle must be crossed by a line ℓ , a path L of which forms a

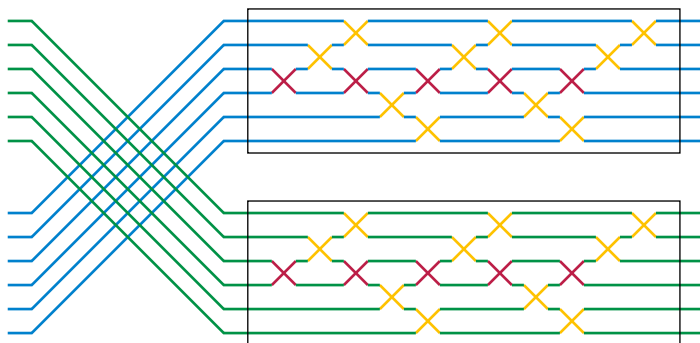


Fig. 8. Two stacked arrangements of $\ell/2$ pseudolines, each with high level complexity, cause our algorithm to create wide drawings no matter how it chooses a wiring diagram

theta-graph together with $P \cup S$. Additionally, because P is assumed to be part of a drawing of a subset of the faces of G , it cannot be crossed by ℓ , for any crossing would cause it to have an attachment vertex between u and v . Therefore, the two degree-three vertices of the theta-graph both belong to S . By the same reasoning as in the proof of Lemma 4, L must be shorter than the other two paths of the theta-graph, so replacing the path that is entirely within S by L would produce a shorter path from u to v , contradicting the construction of S as a shortest path. This contradiction shows that $P \cup S$ must be a face. \square

Theorem 3. *When the greedy ear decomposition embedding algorithm described above is applied to an arrangement graph G , it correctly constructs the canonical embedding of G .*

Proof. We prove by induction on the number of steps of the algorithm that after each step the partial embedding consists of faces of the canonical embedding whose union is a disk. Lemma 4 shows as a base case that the induction hypothesis is true after the first step. In each subsequent step, the ability to find two attachment vertices follows from the fact that arrangement graphs are 2-vertex-connected, which in turn follows from the fact that they can be augmented by a single vertex to be 3-vertex-connected [2]. Lemma 5 shows that, if the induction hypothesis is true after i steps then it remains true after $i + 1$ steps. \square

6 Conclusions

We have found a grid drawing algorithm for pseudoline arrangement graphs that uses area within a small factor of linear, much smaller than the known quadratic grid area lower bounds for arbitrary planar graphs. We have also shown that these graphs have near-linear universal point sets within a constant factor of the same area, and that a simple greedy embedding heuristic suitable for hand solution of Planarity puzzles is guaranteed to find a correct embedding.

The precise area used by our grid drawing algorithm depends on the worst-case behavior of the function $\kappa(\mathcal{D})$ counting the number of crossings in a k -level of an arrangement; closing the gap between the upper and lower bounds for this function remains an important and difficult open problem in combinatorial geometry. However, closing this gap is not the only possible method for improving our drawing algorithm.

A tempting avenue for improvement is to observe that a single pseudoline arrangement may be represented by many different wiring diagrams; therefore, we can select the wiring diagram \mathcal{D} that represent the same pseudoline arrangement and that minimizes $\kappa(\mathcal{D})$. However, this would not improve our worst case width by more than a constant factor. For, if the input forms a pseudoline arrangement constructed by stacking two arrangements of $\ell/2$ lines with maximal k -level complexity, one above the other (Figure 8), then one of these two instances will survive intact in any wiring diagram for the arrangement, forcing our algorithm to produce a drawing with width at least $\kappa_{\max}(\ell/2)$. Further improvements in our algorithm will likely come by finding an alternative layout that avoids the complexity of k -levels, by proving that k -levels are small in the average case if not the worst case, or by reducing the known combinatorial bounds on k -levels.

It is also tempting to consider other drawing styles for arrangement graphs, such as orthogonal drawing (in which each edge is an axis-aligned polyline). Because arrangement graphs contain triangles, such a drawing may be forced to have at least one bend per edge. However, the need either to align neighboring vertices on adjacent rows of the drawing, or to provide space between rows for parallel edge tracks, may cause these drawings to be significantly larger than the straight-line drawings we study, so we have not found an area bound for orthogonal drawing that is as tight as our bound for straight-line drawing.

References

- [1] Shor, P.W.: Stretchability of pseudolines is NP-hard. In: Gritzmann, P., Sturmfels, B. (eds.) *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 4, pp. 531–554. Amer. Math. Soc., Providence (1991)
- [2] Bose, P., Everett, H., Wismath, S.: Properties of arrangement graphs. *International Journal of Computational Geometry & Applications* 13(6), 447–462 (2003)
- [3] Schaefer, M.: Complexity of some geometric and topological problems. In: Eppstein, D., Gansner, E.R. (eds.) *GD 2009*. LNCS, vol. 5849, pp. 334–344. Springer, Heidelberg (2010)
- [4] Dolev, D., Leighton, T., Trickey, H.: Planar embedding of planar graphs. *Advances in Computing Research* 2, 147–161 (1984)
- [5] Valiant, L.G.: Universality considerations in VLSI circuits. *IEEE Transactions on Computers* C-30(2), 135–140 (1981)
- [6] Klawe, M., Paterson, M., Pippenger, N.: Inversions with $n^{1+\Omega(1/\sqrt{\log n})}$ transpositions at the median (September 21, 1982) (unpublished manuscript)
- [7] Tamaki, H., Tokuyama, T.: A characterization of planar graphs by pseudo-line arrangements. *Algorithmica* 35(3), 269–285 (2003)

- [8] Sharir, M., Smorodinsky, S.: Extremal configurations and levels in pseudoline arrangements. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 127–139. Springer, Heidelberg (2003)
- [9] Dey, T.L.: Improved bounds for planar k -sets and related problems. *Discrete and Computational Geometry* 19(3), 373–382 (1998)
- [10] Tóth, G.: Point sets with many k -sets. *Discrete and Computational Geometry* 26(2), 187–194 (2001)
- [11] Goodman, J.E.: Pseudoline arrangements. In: Goodman, J.E., O’Rourke, J. (eds.) *Handbook of Discrete and Computational Geometry*, pp. 83–109. CRC Press (1997)
- [12] Eppstein, D.: Algorithms for drawing media. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 173–183. Springer, Heidelberg (2005)
- [13] Goodman, J.E.: Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics* 32(1), 27–35 (1980)
- [14] Muthukrishnan, S., Sahinalp, S.C., Paterson, M.S.: Grid layout of switching and sorting networks. US Patent 6185220 (2001)
- [15] Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers* C-28(9), 643–647 (1979)
- [16] Edelsbrunner, H., Guibas, L.J.: Topologically sweeping an arrangement. *Journal of Computer and System Sciences* 38(1), 165–194 (1989); Corrigendum, *JCSS* 42(2), 249–251 (1991)
- [17] Snoeyink, J., Hershberger, J.: Sweeping arrangements of curves. In: Proc. 5th ACM Symp. on Computational Geometry (SCG 1989), pp. 354–363 (1989)
- [18] Chrobak, M., Karloff, H.: A lower bound on the size of universal sets for planar graphs. *SIGACT News* 20, 83–86 (1989)
- [19] de Fraysseix, H., Pach, J., Pollack, R.: Small sets supporting Fáry embeddings of planar graphs. In: Proc. 20th ACM Symp. Theory of Computing (STOC 1988), pp. 426–433 (1988)
- [20] Schnyder, W.: Embedding planar graphs on the grid. In: Proc. 1st ACM/SIAM Symp. Discrete Algorithms (SODA 1990), pp. 138–148 (1990)
- [21] Bannister, M.J., Cheng, Z., Devanny, W.E., Eppstein, D.: Superpatterns and universal point sets. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 208–219. Springer, Heidelberg (2013)
- [22] Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified positions. *Amer. Math. Monthly* 98(2), 165–166 (1991)
- [23] Angelini, P., Di Battista, G., Kaufmann, M., Mchedlidze, T., Roselli, V., Squarcella, C.: Small point sets for simply-nested planar graphs. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 75–85. Springer, Heidelberg (2011)
- [24] Fulek, R., Tóth, C.D.: Universal point sets for planar three-trees. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 341–352. Springer, Heidelberg (2013)
- [25] Khuller, S.: Ear decompositions. *SIGACT News* 20(1), 128 (1989)

Incremental Grid-Like Layout Using Soft and Hard Constraints

Steve Kieffer, Tim Dwyer, Kim Marriott, and Michael Wybrow

Caulfield School of Information Technology,
Monash University, Caulfield, Victoria 3145, Australia,
National ICT Australia, Victoria Laboratory,
{Steve.Kieffer,Tim.Dwyer,Kim.Marriott,Michael.Wybrow}@monash.edu

Abstract. We explore various techniques to incorporate grid-like layout conventions into a force-directed, constraint-based graph layout framework. In doing so we are able to provide high-quality layout—with predominantly axis-aligned edges—that is more flexible than previous grid-like layout methods and which can capture layout conventions in notations such as SBGN (Systems Biology Graphical Notation). Furthermore, the layout is easily able to respect user-defined constraints and adapt to interaction in online systems and diagram editors such as Dunnart.

Keywords: constraint-based layout, grid layout, interaction, diagram editors.

1 Introduction

Force-directed layout remains the most popular approach to automatic layout of undirected graphs. By and large these methods untangle the graph to show underlying structure and symmetries with a layout style that is organic in appearance [4]. Constrained graph layout methods extend force-directed layout to take into account user-specified constraints on node positions such as alignment, hierarchical containment and non-overlap [5]. These methods have proven a good basis for semi-automated graph layout in tools such as Dunnart [7] that allow the user to interactively guide the layout by moving nodes or adding constraints.

However, when undirected graphs (and other kinds of diagrams) are drawn by hand it is common for a more grid-like layout style to be used. Grid-based layout is widely used by graphic designers and it is common in hand-drawn biological networks and metro-map layouts. Previous research has shown that grid-based layouts are more memorable than unaligned placements [14]. Virtually all diagram creation tools provide some kind of snap-to-grid feature.

In this paper we investigate how to modify constrained force-directed graph layout methods [5] to create more orthogonal and grid-like layouts with a particular focus on interactive applications such as Dunnart. In Figure 1 we show undirected graphs arranged with our various layout approaches compared with traditional force-directed layout.

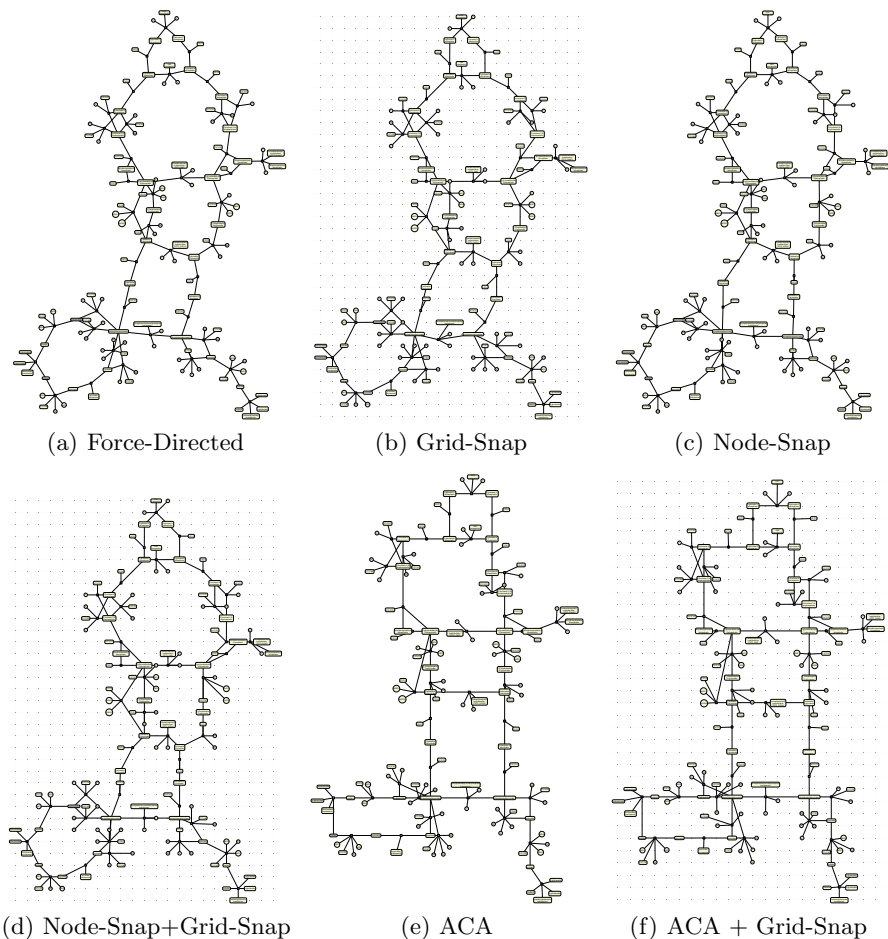


Fig. 1. Different combinations of our automatic layout techniques for grid-like layout compared with standard force-directed layout. The layout is for an SBGN (Systems Biology Graphical Notation) diagram of the Glycolysis-Gluconeogenesis pathway obtained from MetaCROP [15]. In SBGN diagrams, *process nodes* represent individual chemical reactions which typically form links in long metabolic pathways, and are often connected to several degree-1 nodes representing “currency molecules” like ATP and ADP, while precisely two of their neighbours are degree-2 nodes representing principal metabolites. It is conventional that the edges connecting main chemicals and process nodes be axis-aligned in long chains, but not the leaf edges. We achieve this by tailoring the cost functions discussed in Sect. 4.

Before proceeding, it is worth defining what we mean by a *grid-like layout*. It is commonly used to mean some combination of the following properties:

1. nodes are positioned at points on a fairly coarse grid;
2. edges are simple horizontal or vertical lines or in some cases 45° diagonals;
3. nodes of the same kind are horizontally or vertically aligned;
4. edges are orthogonal, i.e., any bends are 90° .

and thus is different from the notion of a *grid layout*, which is simply property (1). In this paper we are primarily interested in producing layouts with properties (1) and (2), though our methods could also achieve (3). We do not consider edges with orthogonal bends, though this could be an extension or achieved through a routing post-process (a simple example of this is provided in [10]).

The standard approach to extending force-directed methods to handle new aesthetic criteria is to add extra “forces” which push nodes in order to satisfy particular aesthetics. One of the most commonly used functions is *stress* [9]. Our first contribution (Sect. 3) is to develop penalty terms that can be added to the stress function to reward placement on points in a grid (Property 1) and to reward horizontal or vertical node alignment and/or horizontal or vertical edges (Property 2 or 3). We call these the *Grid-Snap* and *Node-Snap* methods respectively.

However, additional terms can make the goal function rich in local minima that impede convergence to a more aesthetically pleasing global minimum. Also, such “soft” constraints cannot guarantee satisfaction and so layouts in which nodes are *nearly-but-not-quite aligned* can occur. For this reason we investigate a second approach based on constrained graph layout in which *hard* alignment constraints are automatically added to the layout so as to ensure horizontal or vertical node alignment and thus horizontal or vertical edges (Property 2 or 3). This *adaptive constrained alignment (ACA)* method (Sect. 4) is the most innovative contribution of our paper.

In Sect. 6 we provide an empirical investigation of the speed of these approaches and the quality of layout with respect to various features encoding what we feel are the aesthetic criteria important in grid-like network layout.

While the above approaches can be used in once-off network layout, our original motivation was for interactive-layout applications. In Sect. 5 we discuss an interaction model based on the above for the use of grid-like layout in interactive semi-automatic layout tools such as Dunnart.

Related Work: Our research is related to proposals for automatic grid-like layout of biological networks [1,13,11]. These arrange biological networks with grid coordinates for nodes in addition to various layout constraints. In particular Barsky *et al.* [1] consider alignment constraints between biologically similar nodes and Kojima *et al.* [11] perform layout subject to rectangular containers around functionally significant groups of nodes (e.g., metabolites inside the nucleus of a cell). In general they use fairly straight-forward simulated annealing or simple incremental local-search strategies. Such methods work to a degree but are slow and may never reach a particularly aesthetically appealing minimum.

Another application where grid-like layout is an important aesthetic is automatic metro-map layout. Stott *et al.* [18] use a simple local-search (“hill-climbing”) technique to obtain layout on grid points subject to a number of constraints, such as octilinear edge orientation. Wang and Chi [20] seek similar layout aesthetics but using continuous non-linear optimization subject to octilinearity and planarity constraints. This work, like ours, is based on a quasi-Newton optimization method, but it is very specific to metro-map layout and it is not at all clear how these techniques could be adapted to general-purpose interactive diagramming applications. Metro-map layout algorithms such as [16] are designed to run for many hours before finding a solution.

Another family of algorithms that compute grid-like layout are so-called *orthogonal* graph drawing methods. There have been some efforts to make these incremental, for example Brandes *et al.* [2] can produce an orthogonal drawing of a graph that respects the topology for a given set of initial node positions. Being based on the “Kandinski” orthogonal layout pipeline, extending such a method with user-defined constraints such as alignment or hierarchical containment would require non-trivial engineering of each stage in the pipeline. There is also a body of theoretical work considering the computability and geometric properties of layout with grid-constraints for various classes of graphs, e.g. [3]. Though interesting in its own right, such work is usually not intended for practical application, which is the primary concern of this paper.

There are several examples of the application of soft-constraints to layout. Sugiyama and Misue [19] augment the standard force-model with “magnetic” edge-alignment forces. Ryall *et al.* [17] explored the use of various force-based constraints in the context of an interactive diagramming editor. It is the limitations of such soft constraints (discussed below) which prompt the development of the techniques described in Sect. 4.

2 Aesthetic Criteria

Throughout this paper we assume that we have a graph $G = (V, E, w, h)$ consisting of a set of nodes V , a set of edges $E \subseteq V \times V$ and w_v, h_v are the width and height of node $v \in V$. We wish to find a straight-line 2D drawing for G . This is specified by a pair (x, y) where (x_v, y_v) is the centre point of each $v \in V$.

We quantify grid-like layout quality through the following metrics. In subsequent sections we use these to develop soft and hard constraints that directly or indirectly aim to optimise them. We use these metrics in our evaluation Sect. 6. **Embedding quality** We measure this using the *P-stress* function [8], a variant of *stress* [9] that does not penalise unconnected nodes being more than their desired distance apart. It measures the separation between each pair of nodes $u, v \in V$ in the drawing and their *ideal distance* d_{uv} proportional to the graph theoretic path between them:

$$\sum_{u < v \in V} w_{uv} ((d_{uv} - d(u, v))^+)^2 + \sum_{(u, v) \in E} wp ((d(u, v) - d_L)^+)^2$$

where $d(u, v)$ is the Euclidean distance between u and v , $(z)^+ = z$ if $z \geq 0$ otherwise 0, d_L is an ideal edge length, $wp = \frac{1}{d_L}$, and $w_{uv} = \frac{1}{d_{uv}^2}$.

Edge crossings The number of edge crossings in the drawing.

Edge/node overlap The number of edges intersecting a node box. With straight-line edges this also penalises coincident edges.¹

Angular resolution Edges incident on the same node have a uniform angular separation. Stott *et al.* [18] give a useful formulation:

$$\sum_{v \in V} \sum_{\{e_1, e_2\} \in E} |2\pi / \text{degree}(v) - \theta(e_1, e_2)|$$

Edge obliqueness We prefer horizontal or vertical edges and then—with weaker preference—edges at a 45° orientation. Our precise metric is $M \left| \tan^{-1} \frac{y_u - y_v}{x_u - x_v} \right|$ where $M(\theta)$ is an “M-shaped function” over $[0, \pi/2]$ that highly penalizes edges which are almost but not quite axis-aligned and gives a lower penalty for edges midway between horizontal and vertical.² Other functions like those of [18,11] could be used instead.

Grid placement Average of distances of nodes from their closest grid point.

3 Soft-Constraint Approaches

In this section we describe two new terms that can be combined with the *P-stress* function to achieve more grid-like layout: *NS-stress* for “node-snap stress” and *GS-stress* for “grid-snap stress.” An additional term *EN-sep* gives good separation between nodes and edges. Layout is then achieved by minimizing

$$P\text{-stress} + k_{ns} \cdot \text{NS-stress} + k_{gs} \cdot \text{GS-stress} + k_{en} \cdot \text{EN-sep}$$

where $k_{ns,gs,en}$ control the “strength” of the various components. These extra terms, as defined below, tend to make nodes lie on top of one another. It is essential to avoid this by solving subject to node-overlap prevention constraints, as described in [6]. To obtain an initial “untangled” layout we run with $k_{ns} = k_{gs} = k_{en} = 0$ and without non-overlap constraints (Fig. 1(a)), and then run again with the extra terms and constraints to perform “grid beautification”.

Minimization of the *NS-stress* term favours horizontal or vertical alignment of pairs of connected nodes (Figs. 1(c) and 5). Specifically, taking σ as the distance at which nodes should snap into alignment with one another, we define:

$$\text{NS-stress} = \sum_{(u,v) \in E} q_\sigma(x_u - x_v) + q_\sigma(y_u - y_v) \quad \text{where } q_\sigma(z) = \begin{cases} z^2/\sigma^2 & |z| \leq \sigma \\ 0 & \text{otherwise.} \end{cases}$$

¹ Node/node overlaps are also undesirable. We avoid them completely by using hard non-overlap constraints [6] in all our tests and examples.

² Note that $[0, \pi/2]$ is the range of $|\tan^{-1}|$. The “M” function is zero at 0 and $\pi/2$, a small value $p \geq 0$ at $\pi/4$, a large value $P > 0$ at δ and $\pi/2 - \delta$ for some small $\delta > 0$, and linear in-between.

We originally tried several other penalty functions which turned out not to have good convergence. In particular any smooth function with local maxima at $\pm\sigma$ must be concave-down somewhere over the interval $[-\sigma, \sigma]$, and while differentiability may seem intuitively desirable for quadratic optimization it is in fact trumped by downward concavity, which plays havoc with standard step-size calculations on which our gradient-projection algorithm is based. Thus, obvious choices like an inverted quartic $(1 + (z^2 - \sigma^2)^2)^{-1}$ or a sum of inverted quadratics $(1 + (z + \sigma)^2)^{-1} + (1 + (z - \sigma)^2)^{-1}$ proved unsuitable in place of $q_\sigma(z)$. We review the step size, gradient, and Hessian formulae for our snap-stress functions in [10].

We designed our *GS-stress* function likewise to make the lines of a virtual grid exert a similar attractive force on nodes once within the snap distance σ :

$$\text{GS-stress} = \sum_{u \in V} q_\sigma(x_u - a_u) + q_\sigma(y_u - b_u)$$

where (a_u, b_u) is the closest grid point to (x_u, y_u) (with ties broken by favouring the point closer to the origin), see Fig. 1(b). The grid is defined to be the set of all points $(n\tau, m\tau)$, where n and m are integers, and τ is the “grid size”. With *GS-stress* active it is important to set some other parameters proportional to τ . First, we take $\sigma = \tau/2$. Next, we modify the non-overlap constraints to allow no more than one node centre to be in the vicinity of any one grid point by increasing the minimum separation distance allowed between adjacent nodes to τ . Finally, we found that setting the ideal edge length equal to τ for initial force-directed layout, before activating *GS-stress*, helped to put nodes in positions compatible with the grid.

Our third term *EN-sep* is also a quadratic function based on $q_\sigma(z)$ that separates nodes and nearby axis-aligned edges to avoid node/edge overlaps and coincident edges:

$$\text{EN-sep} = \sum_{e \in E_V \cup E_H} \sum_{u \in V} q_\sigma((\sigma - d(u, e))^+),$$

where E_V and E_H are the sets of vertically and horizontally aligned edges, respectively, and the distance $d(u, e)$ between a node u and an edge e is defined as the length of the normal from u to e if that exists, or $+\infty$ if it does not. Here again we took $\sigma = \tau/2$.

In our experiments we refer to various combinations of these terms and constraints:
Node-Snap: *NS-stress*, *EN-sep*, non-overlap constraints, $k_{gs} = 0$
Grid-Snap: *GS-stress*, *EN-sep*, ideal edge lengths equal to grid size, non-overlap, constraints with separations tailored to grid size, $k_{ns} = 0$.
Node-Snap+Grid-Snap: achieves extra alignment by adding *NS-stress* to the above **Grid-Snap** recipe (i.e. $k_{ns} \neq 0$)

4 Adaptive Constrained Alignment

Another way to customize constrained force-directed layout is by adding *hard* constraints, and in this section we describe how to make force-directed layouts more grid-like simply by adding alignment and separation constraints (Fig. 1(e)).

The algorithm, which we call *Adaptive Constrained Alignment* or *ACA*, is a greedy algorithm which repeatedly chooses an edge in G and aligns it horizontally or vertically (see *adapt_const_align* procedure of Figure 2). It *adapts* to user specified constraints by not adding alignments that violate these. The algorithm halts when the heuristic can no longer apply alignments without creating edge overlaps. Since each edge is aligned at most once, there are at most $|E|$ iterations.

We tried the algorithm with three different heuristics for choosing potential alignments, which we discuss below.

Node overlaps and edge/node overlaps can be prevented with hard non-overlap constraints and the *EN-sep* soft constraint discussed in Section 3, applied either before or after the ACA process. However, coincident edges can be accidentally created and then enforced as we apply alignments if we do not take care to maintain the orthogonal ordering of nodes. If for example two edges (u, v) and (v, w) sharing a common endpoint v are both horizontally aligned, then we must maintain either the ordering $x_u < x_v < x_w$ or the opposite ordering $x_w < x_v < x_u$.

Therefore we define the notion of a *separated alignment*, written $\text{SA}(u, v, D)$ where $u, v \in V$ and $D \in \{\mathbb{N}, \mathbb{S}, \mathbb{W}, \mathbb{E}\}$ is a compass direction. Applying a separated alignment means applying two constraints to the force-directed layout—one alignment and one separation—as follows:

$$\begin{aligned} \text{SA}(u, v, \mathbb{N}) &\equiv x_u = x_v \text{ and } y_v + \beta(u, v) \leq y_u, & \text{SA}(u, v, \mathbb{S}) &\equiv \text{SA}(v, u, \mathbb{N}), \\ \text{SA}(u, v, \mathbb{W}) &\equiv y_u = y_v \text{ and } x_v + \alpha(u, v) \leq x_u, & \text{SA}(u, v, \mathbb{E}) &\equiv \text{SA}(v, u, \mathbb{W}), \end{aligned}$$

where $\alpha(u, v) = (w_u + w_v)/2$ and $\beta(u, v) = (h_u + h_v)/2$. (Thus for example $\text{SA}(u, v, \mathbb{N})$ can be read as, “the ray from u through v points north,” where we think of v as lying north of u when its y -coordinate is smaller.)

```

proc adapt_const_align( $G, C, H$ )
  ( $x, y$ )  $\leftarrow$  cfdl( $G, C$ )
   $SA \leftarrow H(G, C, x, y)$ 
  while  $SA \neq \text{NULL}$ 
     $C.append(SA)$ 
    ( $x, y$ )  $\leftarrow$  cfdl( $G, C$ )
     $SA \leftarrow H(G, C, x, y)$ 
  return ( $x, y, C$ )

proc chooseSA( $G, C, x, y, K$ )
   $S \leftarrow \text{NULL}$ 
   $cost \leftarrow \infty$ 
  for each  $(u, v) \in E$  and dir.  $D$ 
    if not creates_coincidence( $C, x, y, u, v, D$ )
      if  $K(u, v, D) < cost$ 
         $S \leftarrow SA(u, v, D)$ 
         $cost \leftarrow K(u, v, D)$ 
  return  $S$ 

```

Fig. 2. Adaptive constrained alignment algorithm. G is the given graph, C the set of user-defined constraints, H the alignment choice heuristic, and *cfdl* the constrained force-directed layout procedure.

Alignment Choice Heuristics. We describe two kinds of alignment choice heuristics: *generic*, which can be applied to any graph, and *convention-based*, which are intended for use with layouts that conform to special conventions, e.g. SBGN diagrams [12]. Our heuristics are designed according to two principles:

- (1) try to retain the overall shape of the initial force-directed layout;
- (2) do not obscure the graph structure by creating undesirable overlaps

and differ only in the choice of a *cost function* K which is plugged into the procedure `chooseSA` in Figure 2. This relies on procedure `creates_coincidence` which implements the edge coincidence test described by Theorem 1. Among separated alignments which would not lead to an edge coincidence, `chooseSA` selects one of lowest cost. Cost functions may return a special value of ∞ to mark an alignment as never to be chosen.

The `creates_coincidence` procedure works by maintaining a $|V|$ -by- $|V|$ array of flags which indicate for each pair of nodes u, v whether they are aligned in either dimension and whether there is an edge between them. The cost of initializing the array is $\mathcal{O}(|V|^2 + |E| + |C|)$, but this is done only once in ACA. Each time a new alignment constraint is added the flags are updated in $\mathcal{O}(|V|)$ time, due to transitivity of the alignment relation. Checking whether a proposed separated alignment would create an edge coincidence also takes $\mathcal{O}(|V|)$ time, and works according to Theorem 1. (Proof is provided in [10].) Note that the validity of Theorem 1 relies on the fact that we apply separated alignments $\text{SA}(u, v, D)$ only when (u, v) is an edge in the graph.

Theorem 1. *Let G be a graph with separated alignments. Let u, v be nodes in G which are not yet constrained to one another. Then the separated alignment $\text{SA}(u, v, \mathbb{E})$ creates an edge coincidence in G if and only if there is a node w which is horizontally aligned with either u or v and satisfies either of the following two conditions: (i) $(u, w) \in E$ while $x_u < x_w$ or $x_v < x_w$; or (ii) $(w, v) \in E$ while $x_w < x_v$ or $x_w < x_u$. The case of vertical alignments is similar.*

We tried various cost functions, which addressed the aesthetic criteria of Section 2 in different ways. We began with a *basic cost*, which was either an estimate $K_{dS}(u, v, D)$ of the change in the stress function after applying the proposed alignment $\text{SA}(u, v, D)$, or else the negation of the obliqueness of the edge, $K_{ob}(u, v, D) = -\text{obliqueness}((u, v))$, as measured by the function of Section 2. In this way we could choose to address the aesthetic criteria of *embedding quality* or *edge obliqueness*, and we found that the results were similar. Both rules favour placing the first alignments on edges which are almost axis-aligned, and this satisfies our first principle of being guided as much as possible by the shape of the initial force-directed layout. See for example Figure 1.

On top of this basic cost we considered *angular resolution* of degree-2 nodes by adding a large but finite cost that would postpone certain alignments until after others had been attempted; namely, we added a fixed cost of 1000 (ten times larger than average values of K_{dS} and K_{ob}) for any alignment that would make a degree-2 node into a “bend point,” i.e., would make one of its edges horizontally aligned while the other was vertically aligned. This allows long chains of degree-2 nodes to form straight lines, and cycles of degree-2 nodes to form perfect rectangles. For SBGN diagrams we used a modified rule based on *non-leaf degree*, or number of neighbouring nodes which are not leaves (Figs. 1(e) and (f)).

Respecting User-Defined Constraints. Layout constraints can easily wind up in conflict with one another if not chosen carefully. In Dunnart such conflicts are detected during the projection operation described in [5], an *active set*

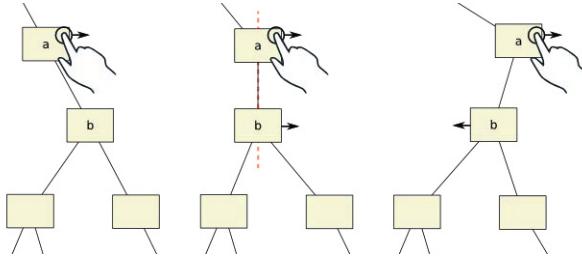


Fig. 3. Interacting with Node-Snap. The user is dragging node a steadily to the right. When the horizontal distance between a and b is less than the average width of these two nodes, the NS -stress function causes b to align with a . As the user continues dragging, the now aligned node b will follow until either a quick jerk of node a breaks the alignment, or else edges attached to b pull it back to the left, overcoming its attraction to a . To the user, the impression is that the alignment persisted until it was “torn” by the underlying forces in the system.

method which iteratively determines the most violated constraint c and satisfies it by minimal disturbance of the node positions. When it is impossible to satisfy c without violating one of the constraints that is already in the active set, c is simply marked *unsatisfiable*, and the operation carries on without it.

For ACA it is important that user-defined constraints are never marked unsatisfiable in deference to an alignment imposed by the process; therefore we term the former *definite* constraints and the latter *tentative* constraints. We employ a modified projection operation which always chooses to mark one or more tentative constraint as unsatisfiable if they are involved in a conflict.

For conflicts involving more than one tentative constraint, we use Lagrange multipliers to choose which one to reject. These are computed as a part of the projection process. Since alignment constraints are equalities (not inequalities) the sign of their Lagrange multiplier does not matter, and a constraint whose Lagrange multiplier is maximal in absolute value is one whose rejection should permit the greatest decrease in the stress function. Therefore we choose this one.

ACA does not snap nodes to grid-points: if desired this can be achieved once ACA has added the alignment constraints by activating Grid-Snap.

5 Interaction

One benefit of the approaches described above is that they are immediately applicable for use in interactive tools where the underlying graph, the prerequisite constraint system, or ideal positions for nodes can all change dynamically. We implemented Node-Snap, Grid-Snap and Adaptive Constrained Alignment for interactive use in the Dunnart diagram editor.³ In Dunnart, automatic layout

³ A video demonstrating interactive use of the approaches described in this paper is available at <http://www.dunnart.org>.

runs continuously in a background worker thread, allowing the layout to adapt immediately to user-specified changes to positions or constraints.

For example, Figure 3 illustrates user interaction with Node-Snap. As the user drags a node around the canvas, it may snap into alignment with an adjacent node. Slowly dragging a node aligned with other nodes will move them together and keep them in alignment, while quickly dragging a node will instead cause it to be torn from any alignments.

When we tried Node-Snap interactively in Dunnart we found that nodes tended to stick together in clumps if the σ parameter of *NS-stress* was larger than their average size in either dimension. We solved this problem by replacing the snap-stress term by

$$\sum_{(u,v) \in E} q_{\alpha(u,v)}(x_u - x_v) + q_{\beta(u,v)}(y_u - y_v)$$

where α, β are as defined in Sect. 4.

In Dunnart, a dragged object is always pinned to the mouse cursor. In the case of Grid-Snap, the dragged node is unpinned and will immediately snap to a grid point on mouse-up. Other nodes, however, will snap-to or tear-away from grid points in response to changing dynamics in the layout system. During dragging we also turn off non-overlap constraints and reapply them on mouse-up. This prevents nodes being unexpectedly pushed out of place as a result of the expanded non-overlap region (Sect. 3). Additionally, since *GS-stress* holds nodes in place, we allow the user to quickly drag a node to temporarily overcome the grid forces and allow the layout to untangle with standard force-directed layout. Once it converges we automatically reapply *GS-stress*.

6 Evaluation

To evaluate the various techniques we applied each to 252 graphs from the “AT&T Graphs” corpus (<ftp://ftp.research.att.com/dist/drawdag/ug.gz>) with between 10 and 244 nodes. We excluded graphs with fewer than 10 nodes and two outlier graphs: one with 1103 nodes and one with 0 edges. We recorded running times of each stage in the automated batch process and the various aesthetic metrics described in Sect. 2, using a MacBook Pro with a 2.3GHz Intel Core I7 CPU. Details of collected data etc. are given in [10].

We found that ACA was the slowest, often taking up to 10 times as long as the other methods, on average around 5 seconds for graphs with around 100 nodes, while the other approaches took around a second. ACA was also sensitive to the density of edges. Of the soft constraint approaches, Grid-Snap (being very local) added very little time over the unconstrained force-directed approach.

The Edge Obliqueness results are shown in Fig. 4 as this is arguably the metric that is most indicative of grid-like layout. Another desirable property of grid-like layout, as noted in Sect. 4 is that longer paths in the graph also be aligned. ACA does a good job of aligning such paths, as is visible in Fig. 1 and 5.

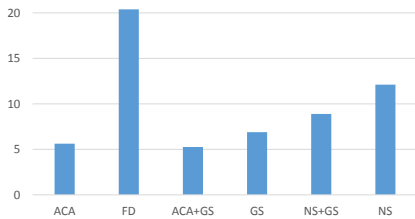


Fig. 4. Edge obliqueness (see Sect. 2) results. The hard-constraint approach ACA is better than either of the soft constraint approaches Grid-Snap (GS) and Node-Snap (NS). The combination of ACA and GS gives the best result.

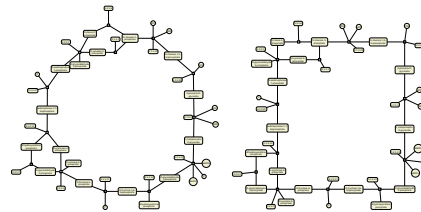


Fig. 5. Layout of a SBGN diagram of Calvin Cycle pathway shows how ACA (right) gives a more pleasing rectangular layout than Node-Snap (left).

7 Conclusion

We explored incorporating grid-like layout conventions into constraint-based graph layout. We give two *soft* approaches (Node-Snap, Grid-Snap) based on adding terms to the goal function, and an adaptive constraint based approach (ACA) in which *hard* alignment constraints are added greedily. ACA is slower but gives more grid-like layout and so is preferred for medium sized graphs.

We have also discussed how the approaches can be integrated into interactive diagramming tools like Dunnart. Here ACA and Grid-Snap provide good initial layouts, while Node-Snap helps the user create further alignments by hand.

Future work is to improve the speed of ACA by adding more than one alignment constraint at a time and also to use Lagrange multipliers (LM) to improve the adaptivity of ACA. One idea is to reject any alignment whose LM exceeds a threshold on each iteration of ACA. Then, running ACA continuously during interaction would allow us to achieve the behaviour illustrated in Fig. 3 through hard rather than soft constraints. Another issue with all the techniques described is the many fiddly parameters, weights and thresholds. We intend to further investigate principled ways to automatically set these.

References

1. Barsky, A., Gardy, J.L., Hancock, R.E., Munzner, T.: Cerebral: a cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics* 23(8), 1040–1042 (2007)
2. Brandes, U., Eiglsperger, M., Kaufmann, M., Wagner, D.: Sketch-driven orthogonal graph drawing. In: Goodrich, M.T., Kobourov, S.G. (eds.) *GD 2002*. LNCS, vol. 2528, pp. 1–11. Springer, Heidelberg (2002)
3. Chrobak, M., Payne, T.H.: A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters* 54(4), 241–246 (1995)
4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Inc. (1999)

5. 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(5), 821–828 (2006)
6. Dwyer, T., Marriott, K., Stuckey, P.J.: Fast node overlap removal. In: Healy, P., Nikolov, N.S. (eds.) *GD 2005*. LNCS, vol. 3843, pp. 153–164. Springer, Heidelberg (2006)
7. Dwyer, T., Marriott, K., Wybrow, M.: Dunnart: A constraint-based network diagram authoring tool. In: Tollis, I.G., Patrignani, M. (eds.) *GD 2008*. LNCS, vol. 5417, pp. 420–431. Springer, Heidelberg (2009)
8. Dwyer, T., Marriott, K., Wybrow, M.: Topology preserving constrained graph layout. In: Tollis, I.G., Patrignani, M. (eds.) *GD 2008*. LNCS, vol. 5417, pp. 230–241. Springer, Heidelberg (2009)
9. Gansner, E.R., Koren, Y., North, S.C.: Graph drawing by stress majorization. In: Pach, J. (ed.) *GD 2004*. LNCS, vol. 3383, pp. 239–250. Springer, Heidelberg (2005)
10. Kieffer, S., Dwyer, T., Marriott, K., Wybrow, M.: Incremental grid-like layout using soft and hard constraints. Tech. Rep. 2013/275, Monash University (2013), <http://www.csse.monash.edu.au/publications/2013/tr-2013-275-full.pdf>
11. Kojima, K., Nagasaki, M., Jeong, E., Kato, M., Miyano, S.: An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC Bioinformatics* 8(1), 76 (2007)
12. Le Novère, N., et al.: The Systems Biology Graphical Notation. *Nature Biotechnology* 27, 735–741 (2009)
13. Li, W., Kurata, H.: A grid layout algorithm for automatic drawing of biochemical networks. *Bioinformatics* 21(9), 2036–2042 (2005)
14. Marriott, K., Purchase, H., Wybrow, M., Goncu, C.: Memorability of visual features in network diagrams. *IEEE Transactions on Visualization and Computer Graphics* 18(12), 2477–2485 (2012)
15. MetaCrop, <http://metacrop.ipk-gatersleben.de>
16. Nöllenburg, M., Wolff, A.: Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE Transactions on Visualization and Computer Graphics* 17(5), 626–641 (2011)
17. Ryall, K., Marks, J., Shieber, S.: An interactive constraint-based system for drawing graphs. In: Robertson, G.G., Schmandt, C. (eds.) *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pp. 97–104. ACM Press (1997)
18. Stott, J., Rodgers, P., Martinez-Ovando, J.C., Walker, S.G.: Automatic metro map layout using multicriteria optimization. *IEEE Transactions on Visualization and Computer Graphics* 17(1), 101–114 (2011)
19. Sugiyama, K., Misue, K.: Graph drawing by the magnetic spring model. *Journal of Visual Languages and Computing* 6(3), 217–231 (1995)
20. Wang, Y.S., Chi, M.T.: Focus+context metro maps. *IEEE Transactions on Visualization and Computer Graphics* 17(12), 2528–2535 (2011)

Using ILP/SAT to Determine Pathwidth, Visibility Representations, and other Grid-Based Graph Drawings^{*}

Therese Biedl¹, Thomas Bläsius², Benjamin Niedermann², Martin Nöllenburg², Roman Prutkin², and Ignaz Rutter²

¹ David R. Cheriton School of Computer Science, University of Waterloo, Canada

² Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

Abstract. We present a simple and versatile formulation of grid-based graph representation problems as an integer linear program (ILP) and a corresponding SAT instance. In a grid-based representation vertices and edges correspond to axis-parallel boxes on an underlying integer grid; boxes can be further constrained in their shapes and interactions by additional problem-specific constraints. We describe a general d -dimensional model for grid representation problems. This model can be used to solve a variety of NP-hard graph problems, including pathwidth, bandwidth, optimum st -orientation, area-minimal (bar- k) visibility representation, boxicity- k graphs and others. We implemented SAT-models for all of the above problems and evaluated them on the Rome graphs collection. The experiments show that our model successfully solves NP-hard problems within few minutes on small to medium-size Rome graphs.

1 Introduction

Integer linear programming (ILP) and Boolean satisfiability testing (SAT) are indispensable and widely used tools in solving many hard combinatorial optimization and decision problems in practical applications [3,9]. In graph drawing, especially for planar graphs, these methods are not frequently applied. A few notable exceptions are crossing minimization [8, 10, 19, 22], orthogonal graph drawing with vertex and edge labels [4] and metro-map layout [26]. Recent work by Chimani et al. [11] uses SAT formulations for testing upward planarity. All these approaches have in common that they exploit problem-specific properties to derive small and efficiently solvable models, but they do not generalize to larger classes of problems.

In this paper we propose a generic ILP model that is flexible enough to capture a large variety of different grid-based graph layout problems, both polynomially-solvable and NP-complete. We demonstrate this broad applicability by adapting the base model to six different NP-complete example problems: pathwidth, bandwidth, optimum st -orientation, minimum area bar- and bar k -visibility representation, and boxicity- k testing. For minimum-area visibility representations and boxicity this is, to the best of our knowledge, the first implementation of an exact solution method. Of course this flexibility comes at the cost of losing some of the efficiency of more specific approaches.

* T. Biedl is supported by NSERC, M. Nöllenburg is supported by the Concept for the Future of KIT under grant YIG 10-209.

Our goal, however, is not to achieve maximal performance for a specific problem, but to provide an easy-to-adapt solution method for a larger class of problems, which allows quick and simple prototyping for instances that are not too large. Our ILP models can be translated into equivalent SAT formulations, which exhibit better performance in the implementation than the ILP models themselves. We illustrate the usefulness of our approach by an experimental evaluation that applies our generic model to the above six NP-complete problems using the well-known Rome graphs [1] as a benchmark set. Our evaluation shows that, depending on the problem, our model can solve small to medium-size instances (sizes varying from about 25 vertices and edges for bar-1 visibility testing up to more than 250 vertices and edges, i.e., all Rome graphs, for optimum *st*-orientation) to optimality within a few minutes. In Section 2, we introduce generic grid-based graph representations and formulate an ILP model for d -dimensional integer grids. We show how this model can be adapted to six concrete one-, two- and d -dimensional grid-based layout problems in Sections 3 and 4. In Section 5 we evaluate our implementations and report experimental results. The implementation is available from `illwww.itl.kit.edu/gdsat`. Omitted proofs are in the full version [2].

2 Generic Model for Grid-Based Graph Representations

In this section we explain how to express d -dimensional boxes in a d -dimensional integer grid as constraints of an ILP or a SAT instance. In the subsequent sections we use these boxes as basic elements for representing vertices and edges in problem-specific ILP and SAT models. Observe that we can restrict ourselves to boxes in integer grids.

Lemma 1. *Any set I of n boxes in \mathbb{R}^d can be transformed into another set I' of n closed boxes on the integer grid $\{1, \dots, n\}^d$ such that two boxes intersect in I if and only if they intersect in I' .*

2.1 Integer Linear Programming Model

We will describe our model in the general case for a d -dimensional integer grid, where $d \geq 1$. Let $\mathcal{R}^d = [1, U_1] \times \dots \times [1, U_d]$ be a bounded d -dimensional integer grid, where $[A, B]$ denotes the set of integers $\{A, A + 1, \dots, B - 1, B\}$. In a *grid-based graph representation* vertices and/or edges are represented as d -dimensional boxes in \mathcal{R}^d . A *grid box* R in \mathcal{R}^d is a subset $[s_1, t_1] \times \dots \times [s_d, t_d]$ of \mathcal{R}^d , where $1 \leq s_k \leq t_k \leq U_k$ for all $1 \leq k \leq d$. In the following we describe a set of ILP constraints that together create a non-empty box for some object v . We denote this ILP as $\mathcal{B}(d)$.

We first extend \mathcal{R}^d by a margin of dummy points to $\bar{\mathcal{R}}^d = [0, U_1 + 1] \times \dots \times [0, U_d + 1]$. We use three sets of binary variables:

$$x_{\mathbf{i}}(v) \in \{0, 1\} \qquad \forall \mathbf{i} \in \bar{\mathcal{R}}^d \qquad (1)$$

$$b_i^k(v) \in \{0, 1\} \qquad \forall 1 \leq k \leq d \text{ and } 1 \leq i \leq U_k \qquad (2)$$

$$e_i^k(v) \in \{0, 1\} \qquad \forall 1 \leq k \leq d \text{ and } 1 \leq i \leq U_k \qquad (3)$$

Variables $x_{\mathbf{i}}(v)$ indicate whether grid point \mathbf{i} belongs to the box representing v ($x_{\mathbf{i}}(v) = 1$) or not ($x_{\mathbf{i}}(v) = 0$). Variables $b_i^k(v)$ and $e_i^k(v)$ indicate whether the box of v may start

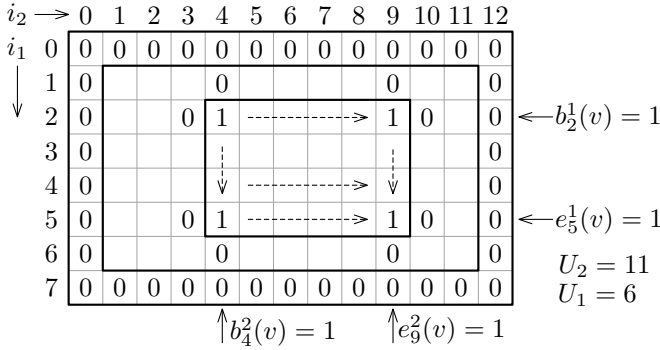


Fig. 1. Example of a 2-dimensional 8×13 grid $\bar{\mathcal{R}}^d$ with a 4×6 grid box and the corresponding variable assignments

or end at position i in dimension k . We use $\mathbf{i}[k]$ to denote the k -th coordinate of grid point $\mathbf{i} \in \mathcal{R}^d$ and $\mathbf{1}_k = (0, \dots, 0, 1, 0, \dots, 0)$ to denote the k -th d -dimensional unit vector. If $d = 1$ we will drop the dimension index of the variables to simplify the notation. The following constraints model a box in \mathcal{R}^d (see Fig. 1 for an example):

$$x_{\mathbf{i}}(v) = 0 \quad \forall \mathbf{i} \in \bar{\mathcal{R}}^d \setminus \mathcal{R}^d \tag{4}$$

$$\sum_{\mathbf{i} \in \mathcal{R}^d} x_{\mathbf{i}}(v) \geq 1 \tag{5}$$

$$\sum_{i \in [1, U_k]} b_i^k(v) = 1 \quad \forall 1 \leq k \leq d \tag{6}$$

$$\sum_{i \in [1, U_k]} e_i^k(v) = 1 \quad \forall 1 \leq k \leq d \tag{7}$$

$$x_{\mathbf{i} - \mathbf{1}_k}(v) + b_{\mathbf{i}[k]}^k(v) \geq x_{\mathbf{i}}(v) \quad \forall \mathbf{i} \in \mathcal{R}^d \text{ and } 1 \leq k \leq d \tag{8}$$

$$x_{\mathbf{i}}(v) \leq x_{\mathbf{i} + \mathbf{1}_k}(v) + e_{\mathbf{i}[k]}^k(v) \quad \forall \mathbf{i} \in \mathcal{R}^d \text{ and } 1 \leq k \leq d \tag{9}$$

Constraint (4) creates a margin of zeroes around \mathcal{R}^d . Constraint (5) ensures that the shape representing v is non-empty, and constraints (6) and (7) provide exactly one start and end position in each dimension. Finally, due to constraints (8) and (9) each grid point inside the specified bounds belongs to v and all other points don't.

Lemma 2. *The ILP $\mathcal{B}(d)$ defined by constraints (1)–(9) correctly models all non-empty grid boxes in \mathcal{R}^d .*

Our example ILP models in Sections 3 and 4 extend ILP $\mathcal{B}(d)$ by constraints controlling additional properties of vertex and edge boxes. For instance, boxes can be easily constrained to be single points, to be horizontal or vertical line segments, to intersect if and only if they are incident or adjacent in G , to meet in endpoints etc. The definition of an objective function for the ILP depends on the specific problem at hand and will be discussed in the problem sections. In the full version [2] we explain how to translate the ILP $\mathcal{B}(d)$ into an equivalent SAT formulation with better practical performance.

3 One-Dimensional Problems

In the following, let $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$. One-dimensional grid-based graph representations can be used to model vertices as intersecting intervals (one-dimensional boxes) or as disjoint points that induce a certain vertex order. We present ILP models for three such problems.

3.1 Pathwidth

The pathwidth of a graph G is a well-known graph parameter with many equivalent definitions. We use the definition via the smallest clique size of an interval supergraph. More precisely, a graph is an *interval graph* if it can be represented as intersection graph of 1-dimensional intervals. A graph G has *pathwidth* $pw(G) \leq p$ if there exists an interval graph H that contains G as a subgraph and for which all cliques have size at most $p+1$. It is NP-hard to compute the pathwidth of an arbitrary graph and even hard to approximate it [5]. There are fixed-parameter algorithms for computing the pathwidth, e.g. [6], however, we are not aware of any implementations of these algorithms. The only available implementations are exponential-time algorithms, e.g., in sage¹.

Problem 1 (Pathwidth). Given a graph $G = (V, E)$, determine the pathwidth of G , i.e., the smallest integer p so that $pw(G) \leq p$.

There is an interesting connection between pathwidth and planar graph drawings of small height. Any planar graph that has a planar drawing of height h has pathwidth at most h [18]. Also, pathwidth is a crucial ingredient in testing in polynomial time whether a graph has a planar drawing of height h [15].

We create a one-dimensional grid representation of G , in which every vertex is an interval and every edge forces the two vertex intervals to intersect. The objective is to minimize the maximum number of intervals that intersect in any given point. We use the ILP $\mathcal{B}(1)$ for a grid $\mathcal{R} = [1, n]$, which already assigns a non-empty interval to each vertex $v \in V$. We add binary variables for the edges of G , a variable $p \in \mathbb{N}$ representing the pathwidth of G , and a set of additional constraints as follows.

$$x_i(e) \in \{0, 1\} \qquad \forall i \in \mathcal{R} \text{ and } e \in E \qquad (10)$$

$$\sum_{i \in \mathcal{R}} x_i(e) \geq 1 \qquad \forall e \in E \qquad (11)$$

$$x_i(uv) \leq x_i(u) \qquad x_i(uv) \leq x_i(v) \qquad \forall uv \in E \qquad (12)$$

$$\sum_{v \in V} x_i(v) \leq p + 1 \qquad \forall i \in \mathcal{R} \qquad (13)$$

Our objective function is to minimize the value of p subject to the above constraints.

It is easy to see that every edge must be represented by some grid point (constraint (11)), and can only use those grid points, where the two end vertices intersect (constraint (12)). Hence the intervals of vertices define some interval graph H that is

¹ www.sagemath.org

a supergraph of G . Constraint (13) enforces that at most $p + 1$ intervals meet in any point, which by Helly's property means that H has clique-size at most $p + 1$. So G has pathwidth at most p . By minimizing p we obtain the desired result. In our implementation we translate the ILP into a SAT instance. We test satisfiability for fixed values of p , starting with $p = 1$ and increasing it incrementally until a solution is found.

Theorem 1. *There exists an ILP/SAT formulation with $O(n(n + m))$ variables and $O(n(n + m))$ constraints / $O(n^3 + n\binom{n}{p+2})$ clauses of maximum size n that has a solution of value $\leq p$ if and only if G has pathwidth $\leq p$.*

With some easy modifications, the above ILP can be used for testing whether a graph is a (proper) interval graph. Section 4.2 shows that boxicity- d graphs, the d -dimensional generalization of interval graphs, can also be recognized by our ILP.

3.2 Bandwidth

The bandwidth of a graph G with n vertices is another classic graph parameter, which is NP-hard to compute [12]; due to the practical importance of the problem there are also a few approaches to find exact solutions to the bandwidth minimization problem. For example, [14] and [25] use the branch-and-bound technique combined with various heuristics. We present a solution that can be easily described using our general framework. However, regarding the running time, it cannot be expected to compete with techniques specially tuned for solving the bandwidth minimization problem.

Let $f : V \rightarrow \{1, \dots, n\}$ be a bijection that defines a linear vertex order. The *bandwidth* of G is defined as $bw(G) = \min_f \max\{f(v) - f(u) \mid uv \in E \text{ and } f(u) < f(v)\}$, i.e., the minimum length of the longest edge in G over all possible vertex orders.

In the full version [2] we describe an ILP that assigns the vertices of G to disjoint grid points and requires for an integer k that any pair of adjacent vertices is at most k grid points apart, i.e., we can test if $bw(G) \leq k$.

Theorem 2. *There exists an ILP/SAT formulation with $O(n^2)$ variables and $O(n \cdot m)$ constraints / $O(n^3)$ clauses of maximum size n that has a solution if and only if G has bandwidth $\leq k$.*

3.3 Optimum st -Orientation

Let G be an undirected graph and let s and t be two vertices of G with $st \in E$. An *st -orientation* of G is an orientation of the edges such that s is the unique source and t is the unique sink [17]. Such an orientation can exist only if $G \cup (s, t)$ is biconnected. Computing an st -orientation can be done in linear time [7, 17], but it is NP-complete to find an st -orientation that minimizes the length of the longest path from s to t , even for planar graphs [28]. It has many applications in graph drawing [27] and beyond.

Problem 2 (Optimum st -orientation). Given a graph $G = (V, E)$ and two vertices $s, t \in V$ with $st \in E$, find an orientation of E such that s is the only source, t is the only sink, and the length of the longest directed path from s to t is minimum.

In the full version [2] we formulate an ILP using points for vertices and non-degenerate intervals for edges that computes a *height- k st-orientation* of G , i.e., an *st-orientation* such that the longest path has length at most k (if one exists).

Theorem 3. *There exists an ILP with $O(n(n + m))$ variables and constraints that computes an optimum st-orientation. Alternatively, there exists an ILP/SAT formulation with $O(k(n + m))$ variables and $O(k(n + m))$ constraints / $O(k^2(n + m))$ clauses of maximum size n that has a solution if and only if a height- k st-orientation of G exists.*

4 Higher-Dimensional Problems

In this section we give examples of two-dimensional visibility graph representations and a d -dimensional grid-based graph representation problem. Let again $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$.

4.1 Visibility Representations

A visibility representation (also: *bar visibility representation* or *weak visibility representation*) of a graph $G = (V, E)$ maps all vertices to disjoint horizontal line segments, called *bars*, and all edges to disjoint vertical bars, such that for each edge $uv \in E$ the bar of uv has its endpoints on the bars for u and v and does not intersect any other vertex bar. Visibility representations are an important visualization concept in graph drawing, e.g., it is well known that a graph is planar if and only if it has a visibility representation [29, 30]. An interesting recent extension are bar k -visibility representations [13], which additionally allow edges to intersect at most k non-incident vertex bars. We use our ILP to compute compact visibility and bar k -visibility representations. Minimizing the area of a visibility representation is NP-hard [24] and we are not aware of any implemented exact algorithms to solve the problem for any $k \geq 0$. By Lemma 1 we know that all bars can be described with integer coordinates of size $O(m + n)$.

Problem 3 (Bar k -Visibility Representation). Given a graph G and an integer $k \geq 0$, find a bar k -visibility representation on an integer grid of size $H \times W$ (if one exists).

Bar visibility representations. Our goal is to test whether G has a visibility representation in a grid with H columns and W rows (and thus minimize H or W). We set $\mathcal{R}^2 = [1, H] \times [1, W]$ and use ILP $\mathcal{B}(2)$ to create grid boxes for all edges and vertices in G . We add one more set of binary variables for vertex-edge incidences and the following constraints.

$$x_i(e, v) \in \{0, 1\} \qquad \forall i \in \mathcal{R}^2 \forall e \in E \forall v \in e \qquad (14)$$

$$b_i^1(v) = e_i^1(v) \qquad \forall i \in [1, U_1] \forall v \in V \qquad (15)$$

$$b_i^2(e) = e_i^2(e) \qquad \forall i \in [1, U_2] \forall e \in E \qquad (16)$$

$$\sum_{v \in V} x_i(v) \leq 1 \qquad \forall i \in \mathcal{R}^2 \qquad (17)$$

$$\sum_{v \in V \setminus e} x_i(v) \leq (1 - x_i(e)) \qquad \forall i \in \mathcal{R}^2 \forall e \in E \qquad (18)$$

$$x_i(e, v) \leq x_i(e) \quad x_i(e, v) \leq x_i(v) \quad \forall i \in \mathcal{R}^2 \forall e \in E \forall v \in e \quad (19)$$

$$\sum_{i \in \mathcal{R}^2} x_i(e, v) \geq 1 \quad \forall e \in E \forall v \in e \quad (20)$$

$$x_i(e, v) \leq b_{i[1]}^1(e) + e_{i[1]}^1(e) \quad \forall i \in \mathcal{R}^2 \forall e \in E \forall v \in e \quad (21)$$

Constraints (15) and (16) ensure that all vertex boxes are horizontal bars of height 1 and all edge boxes are vertical bars of width 1. Constraint (17) forces the vertex boxes to be disjoint; edge boxes will be implicitly disjoint (for a simple graph) due to the remaining constraints. No edge is allowed to intersect a non-incident vertex due to constraint (18). Finally, we need to set the new incidence variables $x_i(e, v)$ for an edge e and an incident vertex v so that $x_i(e, v) = 1$ if and only if e and v share the grid point i . Constraints (19) and (20) ensure that each incidence in G is realized in at least one grid point, but it must be one that is used by the boxes of e and v . Finally, constraint (21) requires edge e to start and end at its two intersection points with the incident vertex boxes. This constraint is optional, but yields a tighter formulation.

Since every graph with a visibility representation is planar (and vice versa) we have $m \in O(n)$. Moreover, our ILP and SAT models can also be used to test planarity of a given graph by setting $H = n$ and $W = 2n - 4$, which is sufficient due to Tamassia and Tollis [29]. This might not look interesting at first sight since planarity testing can be done in linear time [21]. However, we think that this is still useful as one can add other constraints to the ILP model, e.g., to create simultaneous planar embeddings, and use it as a subroutine for ILP formulations of applied graph drawing problems such as metro maps [26] and cartograms.

Theorem 4. *There is an ILP/SAT formulation with $O(HWn)$ variables and $O(HWn)$ constraints / $O(HWn^2)$ clauses of maximum size HW that solves Problem 3 for $k = 0$.*

Bar k -visibility representations. It is easy to extend our previous model for $k = 0$ to test bar k -visibility representations for $k \geq 1$. See [2] for a detailed description.

Theorem 5. *There exists an ILP/SAT formulation with $O(HW(n + m))$ variables and $O(HW(m^2 + n))$ constraints / $O(\binom{HW}{k+1}m + HWm^2)$ clauses of maximum size HW that solves Problem 3 for $k \geq 1$.*

4.2 Boxicity- d Graphs

A graph is said to have *boxicity* d if it can be represented as intersection graph of d -dimensional axis-aligned boxes. Testing whether a graph has boxicity d is NP-hard, even for $d = 2$ [23]. We are not aware of any implemented algorithms to determine the boxicity of a graph. By Lemma 1 we can restrict ourselves to a grid of side length n . In the full version [2], we give an ILP model for testing whether a graph has boxicity d .

Theorem 6. *There exists an ILP with $O(n^d(n + m))$ variables and $O(n^{d+2})$ constraints as well as a SAT instance with $O(n^d(n + m))$ variables and $O(n^{d+2})$ clauses of maximum size $O(n^d)$ to test whether a graph G has boxicity d .*

5 Experiments

We implemented and tested our formulation for minimizing pathwidth, bandwidth, length of longest path in an *st*-orientation, and width of bar-visibility and bar 1-visibility representations, as well as deciding whether a graph has boxicity 2.

We performed the experiments on a single core of an AMD Opteron 6172 processor running Linux 3.4.11. The machine is clocked at 2.1 Ghz, and has 256 GiB RAM. Our implementation (available from <http://illwww.itl.kit.edu/gdsat>) is written in C++ and was compiled with GCC 4.7.1 using optimization `-O3`. As test sample we used the *Rome graphs* dataset [1] which consists of 11533 graphs with vertex number between 10 and 100. 18% of the Rome graphs are planar. The size distribution of the Rome graphs can be found in the full version [2].

We initially used the *Gurobi* solver [20] to test the implementation of the ILP formulations, however it turned out that even for very small graphs ($n < 10$) solving a single instance can take minutes. We therefore focused on the equivalent SAT formulations gaining a significant speed-up. As SAT solver we used *MiniSat* [16] in version 2.2.0. For each of the five minimization problems we determined obvious lower and upper bounds in $O(n)$ for the respective graph parameter. Starting with the lower bound we iteratively increased the parameter to the next integer until a solution was found (or a predefined timeout was exceeded). Each iteration consists of constructing the SAT formulation and executing the SAT solver. We measured the total time spent in all iterations. For boxicity 2 we decided to consider square grids and minimize their side lengths. Thus the same iterative procedure applies to boxicity 2.

Note that for all considered problems a binary search-like procedure for the parameter value did not prove to be efficient, since the solver usually takes more time with increasing parameter value, which is mainly due to the increasing number of variables and clauses. For the one-dimensional problems we used a timeout of 300 seconds, for the two-dimensional problems of 600 seconds.

We ran the instances sorted by size $n + m$ starting with the smallest graphs. If more than 400 consecutive graphs in this order produced timeouts, we ended the experiment prematurely and evaluated only the so far obtained results. Figures 2 and 3 summarize our experimental results and show the percentage of Rome graphs solved within the given time limit, as well as scatter plots with each solved instance represented as a point depending on its graph size and the required computation time.

Pathwidth. As Fig. 2a shows, we were able to compute the pathwidth for 17.0% of all Rome graphs, from which 82% were solved within the first minute and only 3% within the last. Therefore, we expect that a significant increase of the timeout value would be necessary for a noticeable increase of the percentage of solved instances. We note that almost all small graphs ($n + m < 45$) could be solved within the given timeout, however, for larger graphs, the percentage of solved instances rapidly drops, as the red curve in Fig. 2b shows. Almost no graphs with $n + m > 70$ were solved.

Bandwidth. We were able to compute the bandwidth for 22.3% of all Rome graphs (see Fig. 2a), from which 90% were solved within the first minute and only 1.3% within the last. Similarly to the previous case, the procedure terminated successfully within 300 seconds for almost all small graphs ($n + m < 55$ in this case), while almost none of the larger graphs ($n + m > 80$) were solved; see the red curve in Fig. 2c.

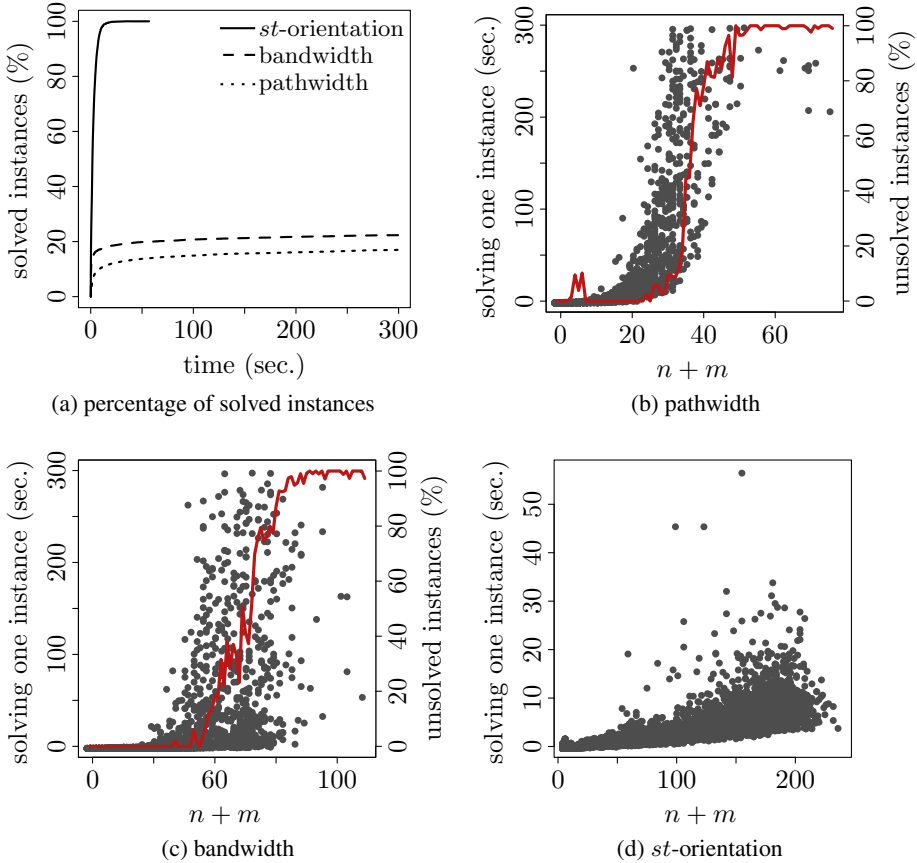


Fig. 2. Experimental results for the one-dimensional problems. (a) Percentage of solved instances. (b)–(d): Time in seconds for solving an instance (dots) and percentage of instances not solved within 300 seconds (red curves), both in relation to $n + m$

Optimum *st*-Orientation. Note that very few of the Rome graphs are biconnected. Therefore, to test our SAT implementation for computing the minimum number of levels in an *st*-orientation, we subdivided each graph into biconnected blocks and removed those with $n \leq 2$, which produced 13606 blocks in total ($3 \leq n + m \leq 230$). Then, for each such block, we randomly selected one pair of vertices s, t , $s \neq t$, connected them by an edge if it did not already exist and ran the iterative procedure. In this way, for the respective choice of s, t we were able to compute the minimum number of levels in an *st*-orientation for all biconnected blocks; see Fig. 2a. Moreover, no graph took longer than 57 seconds, for 97% of the graphs it took less than 10 seconds and for 68% less than 3 seconds. Even for the biggest blocks with $m + n > 200$, the procedure successfully terminated within 15 seconds in 93% of the cases; see Fig. 2d.

Bar Visibility. To compute bar-visibility representations of minimum width, we iteratively tested for each graph all widths W between 1 and $2n - 4$. We used the trivial

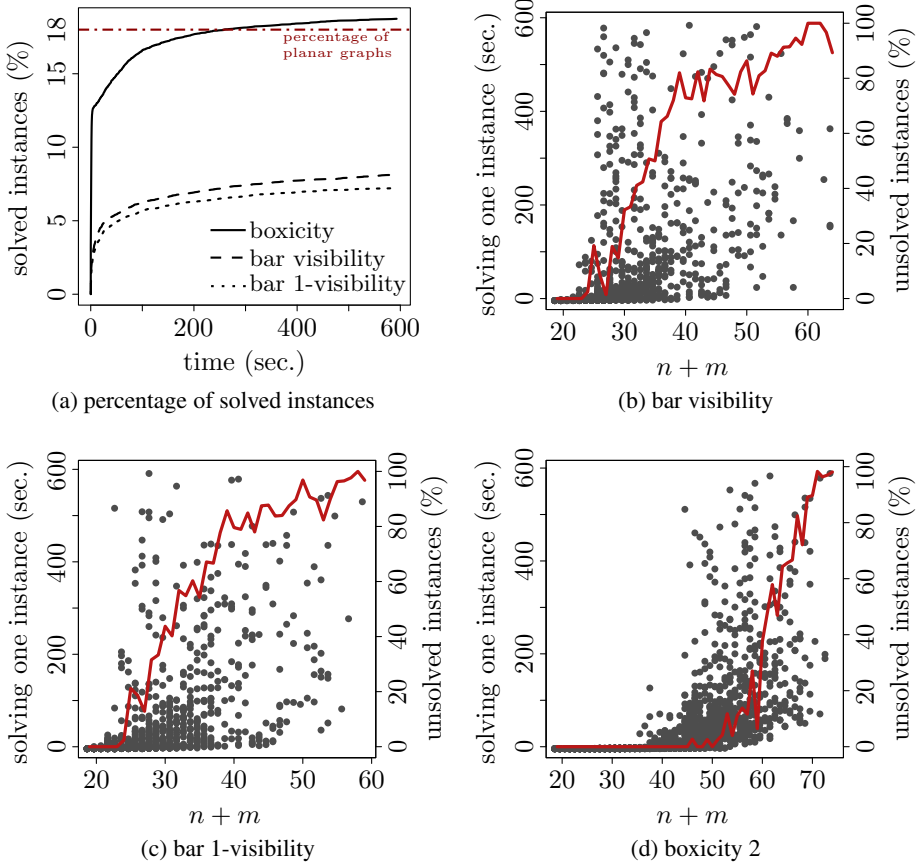


Fig. 3. Experimental results for the two-dimensional problems. (a) Percentage of solved instances. The red horizontal line shows the percentage of planar graphs over all Rome graphs. (b)–(d): Time in seconds for solving an instance (dots) and percentage of instances not solved within 600 seconds (red curves), both in relation to $n + m$.

upper bound $H = n$ for the height. We were able to compute solutions for 28.5% of all 3281 planar Rome graphs (see Fig. 3a), 69% of which were solved within the first minute and less than 0.1% within the last. We were able to solve all small instances with $n + m \leq 23$ and almost none for $n + m > 55$; see the red curve in Fig. 3b.

Bar 1-Visibility. We also ran width minimization for bar 1-visibility representations on all Rome graphs. The procedure terminated successfully within the given time for 833 graphs (7.2% of all Rome graphs), which is close to the corresponding number for bar-visibility; see Fig. 3a. For bar 1-visibility, eight graphs were solved which were not solved for bar-visibility. Interestingly, they were all planar. All but 113 graphs successfully processed in the previous experiment were also successfully processed in this one. A possible explanation for those 113 graphs is that the SAT formulation for bar

1-visibility requires more clauses. All small graphs with $n + m \leq 23$ were processed successfully. Interestingly, for none of the processed graphs the minimum width actually decreased in comparison to their minimum-width bar-visibility representation.

Boxicity-2. For testing boxicity 2, we started with a 3×3 grid for each graph and then increased height and width simultaneously after each iteration. Within the specified timeout of 600 seconds, we were able to decide whether a graph has boxicity 2 for 18.7% of all Rome graphs (see Fig. 3b), 82% of which were processed within the first minute and 0.3% within the last. All of the successfully processed graphs actually had boxicity 2. Small graphs with $n + m \leq 50$ were processed almost completely, while almost none of the graphs with $n + m > 70$ finished; see Fig. 3d.

6 Conclusion

We presented a versatile ILP formulation for determining placement of grid boxes according to problem-specific constraints. We gave six examples of how to extend this formulation for solving numerous NP-hard graph drawing and representation problems. Our experimental evaluation showed that while solving the original ILP is rather slow, the derived SAT formulations perform well for smaller graphs. While our approach is not suitable to replace faster specialized exact or heuristic algorithms, it does provide a simple-to-use tool for solving problems that can be modeled by grid-based graph representations with little implementation effort. This can be useful, e.g., for verifying counterexamples, NP-hardness gadgets, or for solving certain instances in practice.

Many other problems can easily be formulated as ILPs by assigning grid-boxes to vertices or edges. Among those are, e.g., testing whether a planar graph has a straight-line drawing of height h , whether a planar graph has a rectangular dual with integer coordinates and prescribed integral areas, whether a graph is a t -interval graph, or whether a bipartite graph can be represented as a planar bus graph. Important open problems are to reduce the complexity of our formulations and whether approximation algorithms for graph drawing can be derived from our model via fractional relaxation.

References

1. Rome graphs, www.graphdrawing.org/download/rome-graphml.tgz
2. Biedl, T., Bläsius, T., Niedermann, B., Nöllenburg, M., Prutkin, R., Rutter, I.: A versatile ILP/SAT formulation for pathwidth, optimum st-orientation, visibility representation, and other grid-based graph drawing problems. CoRR abs/1308.6778 (2013)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press (2009)
4. Binucci, C., Didimo, W., Liotta, G., Nonato, M.: Orthogonal drawings of graphs with vertex and edge labels. Comput. Geom. Theory Appl. 32(2), 71–114 (2005)
5. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms 18(2), 238–255 (1995)
6. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms 21(2), 358–402 (1996)
7. Brandes, U.: Eager *st*-ordering. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 247–256. Springer, Heidelberg (2002)

8. 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. *Discrete Optimization* 5(2), 373–388 (2008)
9. Chen, D.S., Batson, R.G., Dang, Y.: *Applied Integer Programming*. Wiley (2010)
10. Chimani, M., Mutzel, P., Bomze, I.: A new approach to exact crossing minimization. In: Halperin, D., Mehlhorn, K. (eds.) *ESA 2008*. LNCS, vol. 5193, pp. 284–296. Springer, Heidelberg (2008)
11. Chimani, M., Zeranski, R.: Upward planarity testing via SAT. In: Didimo, W., Patrignani, M. (eds.) *GD 2012*. LNCS, vol. 7704, pp. 248–259. Springer, Heidelberg (2013)
12. Chinn, P.Z., Chvátalova, J., Dewdney, A.K., Gibbs, N.E.: The bandwidth problem for graphs and matrices—a survey. *J. Graph Theory* 6(3), 223–254 (1982)
13. Dean, A.M., Evans, W., Gethner, E., Laison, J.D., Safari, M.A., Trotter, W.T.: Bar k -visibility graphs. *J. Graph Algorithms Appl.* 11(1), 45–59 (2007)
14. Del Corso, G.M., Manzini, G.: Finding exact solutions to the bandwidth minimization problem. *Computing* 62(3), 189–203 (1999)
15. Dujmovic, V., Fellows, M.R., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F.A., Whitesides, S., Wood, D.R.: On the parameterized complexity of layered graph drawing. *Algorithmica* 52(2), 267–292 (2008)
16. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) *SAT 2003*. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
17. Even, S., Tarjan, R.E.: Computing an st -numbering. *Theoret. Comput. Sci.* 2(3), 339–344 (1976)
18. Felsner, S., Liotta, G., Wismath, S.: Straight-line drawings on restricted integer grids in two and three dimensions. *J. Graph Algorithms Appl.* 7(4), 363–398 (2003)
19. Gange, G., Stuckey, P.J., Marriott, K.: Optimal k -level planarization and crossing minimization. In: Brandes, U., Cornelsen, S. (eds.) *GD 2010*. LNCS, vol. 6502, pp. 238–249. Springer, Heidelberg (2011)
20. Gurobi Optimization, Inc.: *Gurobi optimizer reference manual* (2013)
21. Hopcroft, J., Tarjan, R.: Efficient planarity testing. *J. ACM* 21(4), 549–568 (1974)
22. Jünger, M., Mutzel, P.: 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.* 1(1), 1–25 (1997)
23. Kratochvíl, J.: A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Appl. Math.* 52(3), 233–252 (1994)
24. Lin, X., Eades, P.: Towards area requirements for drawing hierarchically planar graphs. *Theoret. Comput. Sci.* 292(3), 679–695 (2003)
25. Martí, R., Campos, V., Piñana, E.: A branch and bound algorithm for the matrix bandwidth minimization. *Europ. J. of Operational Research* 186, 513–528 (2008)
26. Nöllenburg, M., Wolff, A.: Drawing and labeling high-quality metro maps by mixed-integer programming. *IEEE TVCG* 17(5), 626–641 (2011)
27. Papamanthou, C., G. Tollis, I.: Applications of parameterized st -orientations. *J. Graph Algorithms Appl.* 14(2), 337–365 (2010)
28. Sadasivam, S., Zhang, H.: NP-completeness of st -orientations for plane graphs. *Theoret. Comput. Sci.* 411(7-9), 995–1003 (2010)
29. Tamassia, R., Tollis, I.: A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.* 1(1), 321–341 (1986)
30. Wismath, S.K.: Characterizing bar line-of-sight graphs. In: *Proc. First Ann. Symp. Comput. Geom.*, SCG 1985, pp. 147–152. ACM, New York (1985)

Untangling Two Systems of Noncrossing Curves*

Jiří Matoušek^{1,2}, Eric Sedgwick³, Martin Tancer^{1,4}, and Uli Wagner⁵

¹ Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic

² Institute of Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland

³ School of CTI, DePaul University, 243 S. Wabash Ave, Chicago, IL 60604, USA

⁴ Institutionen för matematik, KTH, 100 44 Stockholm, Sweden

⁵ IST Austria, Am Campus 1, 3400 Klosterneuburg, Austria

Abstract. We consider two systems $(\alpha_1, \dots, \alpha_m)$ and $(\beta_1, \dots, \beta_n)$ of curves drawn on a compact two-dimensional surface \mathcal{M} with boundary. Each α_i and each β_j is either an arc meeting the boundary of \mathcal{M} at its two endpoints, or a closed curve. The α_i are pairwise disjoint except for possibly sharing endpoints, and similarly for the β_j . We want to “untangle” the β_j from the α_i by a self-homeomorphism of \mathcal{M} ; more precisely, we seek an homeomorphism $\varphi: \mathcal{M} \rightarrow \mathcal{M}$ fixing the boundary of \mathcal{M} pointwise such that the total number of crossings of the α_i with the $\varphi(\beta_j)$ is as small as possible. This problem is motivated by an application in the algorithmic theory of embeddings and 3-manifolds.

We prove that if \mathcal{M} is planar, i.e., a sphere with $h \geq 0$ boundary components (“holes”), then $O(mn)$ crossings can be achieved (independently of h), which is asymptotically tight, as an easy lower bound shows. In general, for an arbitrary (orientable or nonorientable) surface \mathcal{M} with h holes and of (orientable or nonorientable) genus $g \geq 0$, we obtain an $O((m+n)^4)$ upper bound, again independent of h and g .

Keywords: Curves on 2-manifolds, simultaneous planar drawings, Lickorish’s theorem.

1 Introduction

Let \mathcal{M} be a surface, by which we mean a two-dimensional compact manifold with (possibly empty) boundary $\partial\mathcal{M}$.

By the classification theorem for surfaces, if \mathcal{M} is orientable, then \mathcal{M} is homeomorphic to a sphere with $h \geq 0$ holes and $g \geq 0$ attached handles; the number g is also called the *orientable genus* of \mathcal{M} . If \mathcal{M} is nonorientable, then it is homeomorphic to a sphere with $h \geq 0$ holes and with $g \geq 0$ *cross-caps*; in this case, the integer g is known as the *nonorientable genus* of \mathcal{M} . In the sequel, the word

* Research supported by the ERC Advanced Grant No. 267165. Moreover, the research of J.M. was also partially supported by Grant GRADR Eurogiga GIG/11/E023, the research of M.T. was supported by a Göran Gustafsson postdoctoral fellowship, and the research of U.W. was supported by the Swiss National Science Foundation (Grant SNSF-PP00P2-138948)

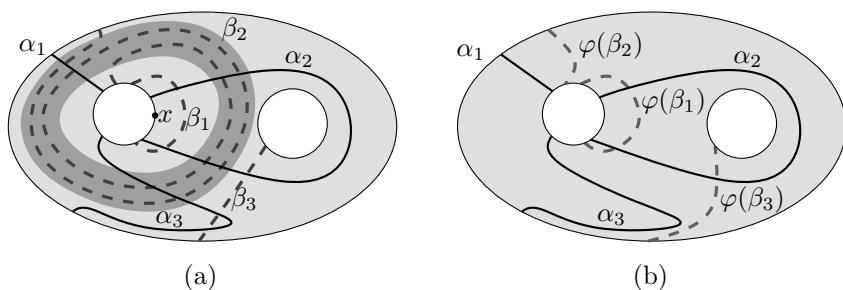


Fig. 1. Systems A and B of curves on a surface \mathcal{M} , with $g = 0$ and $h = 3$ (a), and a re-drawing of B via a ∂ -automorphism φ reducing the number of intersections (b).

“genus” will mean orientable genus for orientable surfaces and nonorientable genus for nonorientable surfaces.

We will consider curves in \mathcal{M} that are *properly embedded*, i.e., every curve is either a simple arc meeting the boundary $\partial\mathcal{M}$ exactly at its two endpoints, or a simple closed curve avoiding $\partial\mathcal{M}$. An *almost-disjoint system of curves* in \mathcal{M} is a collection $A = (\alpha_1, \dots, \alpha_m)$ of curves that are pairwise disjoint except for possibly sharing endpoints.

In this paper we consider the following problem: We are given two almost-disjoint systems $A = (\alpha_1, \dots, \alpha_m)$ and $B = (\beta_1, \dots, \beta_n)$ of curves in \mathcal{M} , where the curves of B intersect those of A possibly very many times, as in Fig. 1(a). We would like to “redraw” the curves of B in such a way that they intersect those of A as little as possible.

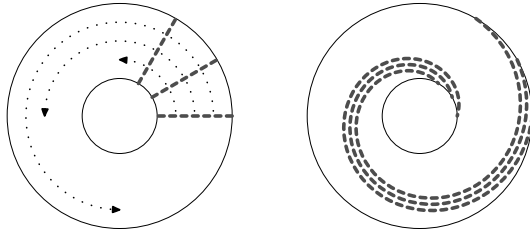
We consider re-drawings only in a restricted sense, namely, induced by ∂ -automorphisms of \mathcal{M} , where a ∂ -automorphism is a homeomorphism $\varphi: \mathcal{M} \rightarrow \mathcal{M}$ that fixes the boundary $\partial\mathcal{M}$ pointwise. Thus, given the α_i and the β_j , we are looking for a ∂ -automorphism φ such that the number of intersections (crossings) between $\alpha_1, \dots, \alpha_m$ and $\varphi(\beta_1), \dots, \varphi(\beta_n)$ is as small as possible (where sharing endpoints does not count). We call this minimum number of crossings achievable through any choice of φ the *entanglement number* of the two systems A and B .

In the orientable case, let $f_{g,h}(m, n)$ denote the maximum entanglement number of any two systems $A = (\alpha_1, \dots, \alpha_m)$ and $B = (\beta_1, \dots, \beta_n)$ of almost-disjoint curves on an orientable surface of genus g with h holes. Analogously, we define $\hat{f}_{g,h}(m, n)$ as the maximum entanglement number of any two systems A and B of m and n curves, respectively, on a nonorientable surface of genus g with h holes. It is easy to see that f and \hat{f} are nondecreasing in m and n , which we will often use in the sequel.

To give the reader some intuition about the problem, let us illustrate which re-drawings are possible with a ∂ -automorphism and which are not. In the example of Fig. 1, it is clear that the two crossings of β_3 with α_3 can be avoided by sliding β_3 aside.¹ It is perhaps less obvious that the crossings of β_2 can also be

¹ This corresponds to an *isotopy* of the surface that fixes the boundary pointwise.

eliminated: To picture a suitable ∂ -automorphism, one can think of an annular region in the interior of \mathcal{M} , shaded darkly in Fig. 1 (a), that surrounds the left hole and β_1 and contains most of the spiral formed by β_2 . Then we cut \mathcal{M} along the outer boundary of that annular region, twist the region two times (so that the spiral is unwound), and then we glue the outer boundary back. Here is an example of a single twist of an annulus; straight-line curves on the left are transformed to spirals on the right .



On the other hand, it is impossible to eliminate the crossings of β_1 or β_3 with α_2 by a ∂ -automorphism. For example, we cannot re-route β_1 to go around the right hole and thus avoid α_2 , since this re-drawing is not induced by any ∂ -automorphism φ : indeed, β_1 separates the point x on the boundary of left hole from the right hole, whereas α_2 does not separate them; therefore, the curve α_2 has to intersect $\varphi(\beta_1)$ at least twice, once when it leaves the component containing x and once when it returns to this component.

A rather special case of our problem, with $m = n = 1$ and only closed curves, was already considered by Lickorish [Lic62], who showed that the intersection of a pair of simple closed curves can be simplified via Dehn twists (and thus a ∂ -automorphism) so that they meet at most twice (also see Stillwell [Sti80]). The case with $m = 1$, n arbitrary, only closed curves, and \mathcal{M} possibly nonorientable was proposed in 2010 as a Mathoverflow question [Huy10] by T. Huynh. In an answer A. Putman proposes an approach via the “change of coordinates principle” (see, e.g., [FM11, Sec. 1.3]), which relies on the classification of 2-dimensional surfaces—we will also use it at some points in our argument.

The results. A natural idea for bounding $f_{g,h}(m, n)$ and $\hat{f}_{g,h}(m, n)$ is to proceed by induction, employing the change of coordinates principle mentioned above. This does indeed lead to finite bounds, but the various induction schemes we have tried always led to bounds at least exponential in one of m, n . Independently of our work, Geelen, Huynh, and Richter [GHR13] also recently proved bounds of this kind; see the discussion below. Partially influenced by the results on exponentially many intersections in representations of string graphs and similar objects (see [KM91,SSŠ03]), we first suspected that an exponential behavior might be unavoidable. Then, however, we found, using a very different approach, that polynomial bounds actually do hold.

For planar \mathcal{M} , i.e., $g = 0$, we obtain an asymptotically tight bound:

Theorem 1. *For planar \mathcal{M} , we have $f_{0,h}(m, n) = O(mn)$, independent of h .*

Here and in the sequel, the constants implicit in the O -notation are absolute, independent of g and h .

A simple example providing a lower bound of $2mn$ is obtained, e.g., by replicating α_2 in Fig. 1 m times and β_1 n times.

In general, we obtain the following bounds:

- Theorem 2.** (i) *For the orientable case, $f_{g,h}(m, n) = O((m + n)^4)$.*
 (ii) *For the nonorientable case, $\hat{f}_{g,h}(m, n) = O((m + n)^4)$.*

Both parts of Theorems 2 are derived from the planar case, Theorem 1. In the orientable case, we use the following results on genus reduction. For a more convenient notation, let us set $L = \max(m, n)$.

Proposition 1 (Orientable genus reductions).

- (i) *For $g > L$, we have $f_{g,h}(m, n) \leq f_{L,g+h-L}(m, n)$.*
 (ii) *$f_{g,h}(m, n) \leq f_{0,h+1}(cg(m + g), cg(n + g))$ for a suitable constant $c > 0$.*

Theorem 2 (i), the orientable case, follows immediately from Proposition 1 and the planar bound.

In the nonorientable case, Theorem 2 (ii) is derived in two steps. First, analogous to Proposition 1 (i), we have the following reduction:

Proposition 2 (Nonorientable genus reduction). *For $g > 4L + 2$, we have $\hat{f}_{g,h}(m, n) \leq \hat{f}_{g',h'}(m, n)$, where $g' = 4L + 2 - (g \bmod 2)$ and $h' = h + \lceil g/2 \rceil - 2L - 1$.*

The second step is a reduction to the orientable case.

Proposition 3 (Orientability reduction). *There is a constant c such that $\hat{f}_{g,h}(m, n) \leq f_{\lfloor (g-1)/2 \rfloor, h+1+(g \bmod 2)}(c(g + m), c(g + n))$.*

Table 1 summarizes the proof of Theorem 2.

Motivation. We were led to the question concerning untangling curves on surfaces while working on a project on 3-manifolds and embeddings. Specifically, we are interested in an algorithm for the following problem: given a 3-manifold M with boundary, does M embed in the 3-sphere? A special case of this problem, with the boundary of M a torus, was solved in [JS03]. The problem is motivated, in turn, by the question of algorithmically testing the embeddability of a 2-dimensional simplicial complex in \mathbb{R}^3 ; see [MTW11].

In our current approach, which has not yet been completely worked out, we need just a finite bound on $f_{g,h}(m, n)$. However, we consider the problem investigated in this paper interesting in itself and contributing to a better understanding of combinatorial properties of curves on surfaces.

As mentioned above, the question studied in the present paper has also been investigated independently by Geelen, Huynh, and Richter [GHR13], with a rather different and very strong motivation stemming from the theory of graph minors, namely the question of obtaining explicit upper bounds for the graph minor algorithms of Robertson and Seymour. Phrased in the language of the present paper, Geelen et al. [GHR13, Theorem 2.1] show that $f_{g,h}(m, n)$ and $\hat{f}_{g,h}(m, n)$ are both bounded by $n3^m$.

Table 1. A summary of the proof

1. For a planar surface, temporarily remove the holes not incident to any α_i or β_j , and contract the remaining “active” holes, augment the resulting planar graphs to make them 3-connected. Make a simultaneous plane drawing of the resulting planar graphs G_1 and G_2 with every edge of G_1 intersecting every edge of G_2 at most $O(1)$ times. Decontract the active holes and put the remaining holes back into appropriate faces (Theorem 1; Section 2).
2. If the genus is larger than $c(m+n)$, find handles or cross-caps avoided by the α_i and β_j , temporarily remove them, untangle the α_i and β_j , and put the handles or cross-caps back (Propositions 1 (i) and 2; the proofs are omitted from this extended abstract).
3. If the surface is nonorientable, make it orientable by cutting along a suitable curve that intersects the α_i and β_j at most $O(m+n)$ times, untangle the resulting pieces of the α_i and β_j , and glue back (Proposition 3).
4. Make the surface planar by cutting along a suitable system of curves (canonical system of loops), untangle the resulting pieces of the α_i and β_j , and glue back (Proposition 1 (ii)).

2 Planar Surfaces

In this section we prove Theorem 1. In the proof we use the following basic fact (see, e.g., [MT01]).

Lemma 1. *If G is a maximal planar simple graph (a triangulation), then for every two planar drawings of G in S^2 there is an automorphism ψ of S^2 converting one of the drawings into the other (and preserving the labeling of the vertices and edges). Moreover, if an edge e is drawn by the same arc in both of the drawings, w.l.o.g. we may assume that ψ fixes it pointwise.*

Let us introduce the following piece of terminology. Let G be as in the lemma, and let D_G, D'_G be two planar drawings of G . We say that D_G, D'_G are *directly equivalent* if there is an orientation-preserving automorphism of S^2 mapping D_G to D'_G , and we call D_G, D'_G *mirror-equivalent* if there is an orientation-reversing automorphism of S^2 converting D_G into D'_G .

We will also rely on a result concerning simultaneous planar embeddings; see [BKR12]. Let V be a vertex set and let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two planar graphs on V . A planar drawing D_{G_1} of G_1 and a planar drawing D_{G_2} of G_2 are said to form a *simultaneous embedding* of G_1 and G_2 if each vertex $v \in V$ is represented by the same point in the plane in both D_{G_1} and D_{G_2} .

We note that G_1 and G_2 may have common edges, but they are not required to be drawn in the same way in D_{G_1} and in D_{G_2} . If this requirement is added, one speaks of a *simultaneous embedding with fixed edges*. There are pairs of planar graphs known that do not admit any simultaneous embedding with fixed edges

(and consequently, no simultaneous straight-line embedding). An important step in our approach is very similar to the proof of the following result.

Theorem 3 (Erten and Kobourov [EK05]). *Every two planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ admit a simultaneous embedding in which every edge is drawn as a polygonal line with at most 3 bends.*

We will need the following result, which follows easily from the proof given in [EK05]. For the reader's convenience, instead of just pointing out the necessary modifications, we present a full proof.

Theorem 4. *Every two planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ admit a simultaneous, piecewise linear embedding in which every two edges e_1 of G_1 and e_2 of G_2 intersect at least once and at most C times, for a suitable constant C .²*

In addition, if both G_1 and G_2 are maximal planar graphs, let us fix a planar drawing D'_{G_1} of G_1 and a planar drawing D'_{G_2} of G_2 . The planar drawing of G_1 in the simultaneous embedding can be required to be either directly equivalent to D'_{G_1} , or mirror-equivalent to it, and similarly for the drawing of G_2 (each of the four combinations can be prescribed).

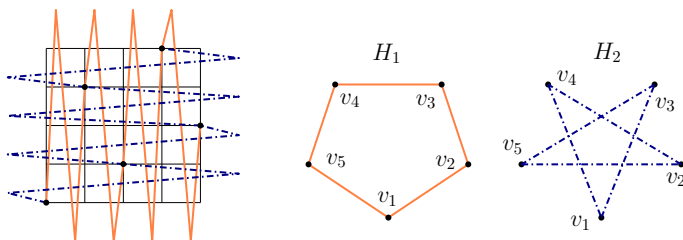
Proof. For the beginning, we assume that both graphs are Hamiltonian. Later on, we will drop this assumption.

Let v_1, v_2, \dots, v_n be the order of the vertices as they appear on (some) Hamiltonian cycle H_1 of G_1 . Since the vertex set V is common for G_1 and G_2 , there is a permutation $\pi \in S(n)$ such that $v_{\pi(1)}, \dots, v_{\pi(n)}$ is the order of the vertices as they appear on some Hamiltonian cycle H_2 of G_2 .

We draw the vertex v_i in the grid point $p_i = (i, \pi(i))$, $i = 1, 2, \dots, n$. Let S be the square $[1, n] \times [1, n]$. A *bispiked* curve is an x -monotone polygonal curve with two bends such that it starts inside S ; the first bend is above S , the second bend is below S and it finishes in S again.

The $n - 1$ edges $v_i v_{i+1}$, of H_1 , $i = 1, 2, \dots, n - 1$, are drawn as bispiked curves starting in p_i and finishing in p_{i+1} . In order to distinguish edges and their drawings, we denote these bispiked curves by $c(i, i + 1)$.

Similarly, we draw the edges $v_{\pi(i)} v_{\pi(i+1)}$ of H_2 , $i = 1, 2, \dots, n - 1$, as y -monotone analogs of bispiked curves, where the first bend is on the left of S and the second is on the right of S ; here is an example:



² An obvious bound from the proof is $C \leq 36$, since every edge in this embedding is drawn using at most 5 bends. By a more careful inspection, one can easily get $C \leq 25$, and a further improvement is probably possible.

We continue only with description of how to draw G_1 ; G_2 is drawn analogously with the grid rotated by 90 degrees.

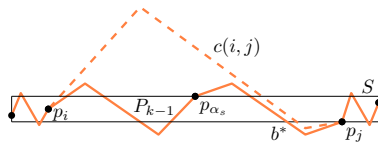
Let D'_{G_1} be a planar drawing of G_1 . Every edge from E_1 that is not contained in H_1 is drawn either inside D'_{H_1} or outside. Thus, we split $E_1 \setminus E(H_1)$ into two sets E'_1 and E''_1 .

Let P_0 be the polygonal path obtained by concatenation of the curves $c(1, 2), c(2, 3), \dots, c(n - 1, n)$. Now our task is to draw the edges of $E'_1 \cup \{v_1v_n\}$ as bispiked curves, all above P_0 , and then the edges of E''_1 below P_0 .

We start with E'_1 and we draw edges from it one by one, in a suitably chosen order, while keeping the following properties.

- (P1) Every edge v_iv_j , where $i < j$, is drawn as a bispiked curve $c(i, j)$ starting in p_i and ending in p_j .
- (P2) The x -coordinate of the second bend of $c(i, j)$ belongs to the interval $[j - 1, j]$.
- (P3) The polygonal curve P_k that we see from above after drawing the k th edge is obtained as a concatenation of some curves $c(1, i_1), c(i_1, i_2), \dots, c(i_\ell, n)$.

Here is an illustration; the square S is deformed for the purposes of the drawing:



Initially, before drawing the first edge, the properties are obviously satisfied.

Let us assume that we have already drawn $k - 1$ edges of E'_1 , and let us focus on drawing the k th edge. Let $e = v_iv_j \in E'_1$ be an edge that is not yet drawn and such that all edges below e are already drawn, where “below e ” means all edges $v_{i'}v_{j'} \in E'_1$ with $i \leq i' < j' \leq j$, $(i, j) \neq (i', j')$. (This choice ensures that we will draw all edges of E'_1 .)

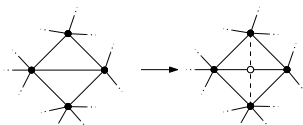
Since D'_{G_1} is a planar drawing, we know that there is no edge $v_{i'}v_{j'} \in E'_1$ with $i < i' < j < j'$ or $i' < i < j' < j$, and so the points p_i and p_j have to belong to P_{k-1} . The subpath P' of P_{k-1} between p_i and p_j is the concatenation of curves $c(i, \alpha_1), c(\alpha_1, \alpha_2), \dots, c(\alpha_s, j)$ as in the inductive assumptions. In particular, the x -coordinate of the second bend b^* of $c(\alpha_s, j)$ belongs to the interval $[j - 1, j]$. We draw $c(i, j)$ as follows: The second bend of $c(i, j)$ is slightly above b^* but still below the square S . The first bend of S is sufficiently high above S (with the x -coordinate somewhere between i and $j - 1$) so that the resulting bispiked curve $c(i, j)$ does not intersect P_{k-1} . The properties (P1) and (P2) are obviously satisfied by the construction. For (P3), the path P_k is obtained from P_{k-1} by replacing P' with $c(i, j)$.

After drawing the edges of E'_1 , we draw v_1v_n in the same way. Then we draw the edges of E''_1 in a similar manner as those of E'_1 , this time as bispiked curves below P_0 . This finishes the construction for Hamiltonian graphs.

Now we describe how to adjust this construction for non-Hamiltonian graphs, in the spirit of [EK05].

First we add edges to G_1 and G_2 so that they become planar triangulations. This step does not affect the construction at all, except that we remove these edges in the final drawing.

Next, we subdivide some of the edges of G_i with *dummy* vertices. Moreover, we attach two new *extra* edges to each dummy vertex, as in the following illustration:



By choosing the subdivided edges suitably, one can obtain a 4-connected, and thus Hamiltonian, graph; see [EK05, Proof of Theorem 2] for details (this idea previously comes from [KW02]). An important property of this construction is that each edge of G_i is subdivided at most once.

In this way, we obtain new Hamiltonian graphs G'_1 and G'_2 , for which we want to construct a simultaneous drawing as in the first part of the proof. A little catch is that G'_1 and G'_2 do not have same vertex sets, but this is easy to fix. Let d_i be the number of dummy vertices of G'_i , $i = 1, 2$, and say that $d_1 \geq d_2$. We pair the d_2 dummy vertices of G'_2 with some of the dummy vertices of G'_1 . Then we iteratively add $d_1 - d_2$ new triangles to G'_2 , attaching each of them to an edge of a Hamiltonian cycle. This operation keeps Hamiltonicity and introduces $d_1 - d_2$ new vertices, which can be matched with the remaining $d_1 - d_2$ dummy vertices in G'_1 .

After drawing resulting graphs, we remove all extra dummy vertices and extra edges added while introducing dummy vertices. An original edge e that was subdivided by a dummy vertex is now drawn as a concatenation of two bispiked curves. Therefore, each edge is drawn with at most 5 bends.

Two edges with 5 bends each may in general have at most 36 intersections, but in our case, there can be at most 25 intersections, since the union of the two segments before and after a dummy vertex is both x -monotone and y -monotone.

Because of the bispiked drawing of all edges, it is also clear that every edge of G_1 crosses every edge of G_2 at least once.

Finally, the requirements on directly equivalent or mirror-equivalent drawings can easily be fulfilled by interchanging the role of top and bottom in the drawing of G_1 or left and right in the drawing of G_2 . Theorem 4 is proved. \square

Proof of Theorem 1. Let a planar surface \mathcal{M} and the curves $\alpha_1, \dots, \alpha_m, \beta_1, \dots, \beta_n$ be given; we assume that \mathcal{M} is a subset of S^2 . From this we construct a set V of $O(m+n)$ vertices in S^2 and planar drawings D_{G_1} and D_{G_2} of two simple graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ in S^2 , as follows.

1. We put all endpoints of the α_i and of the β_j into V .

2. We choose a new vertex in the interior of each α_i and each β_j , or two distinct vertices if α_i or β_j is a loop with a single endpoint, or three vertices if α_i or β_j is a closed curve, and we add all of these vertices to V . These new vertices are all distinct and do not lie on any curves other than where they were placed.
3. If the boundary of a hole in \mathcal{M} already contains a vertex introduced so far, we add more vertices so that it contains at least 3 vertices of V . This finishes the construction of V .
4. To define the edge set $E_1 = E(G_1)$ and the planar drawing D_{G_1} , we take the portions of the curves $\alpha_1, \dots, \alpha_m$ between consecutive vertices of V as edges of E_1 . Similarly, we make the arcs of the boundaries of the holes into edges in E_1 ; these will be called the *hole edges*. By the choice of the vertex set V above, this yields a simple plane graph.
5. Then we add new edges to E_1 so that we obtain a drawing D_{G_1} in S^2 of a maximal planar simple graph G_1 (i.e., a triangulation) on the vertex set V . While choosing these edges, we make sure that all holes containing no vertices of G lie in faces of D_{G_1} adjacent to some of the α_i . New edges drawn in the interior of a hole are also called *hole edges*.
6. We construct $G_2 = (V, E_2)$ and D_{G_2} analogously, using the curves β_1, \dots, β_m . We make sure that all hole edges are common to G_1 and G_2 .

After this construction, each hole of \mathcal{M} contains either no vertex of V on its boundary or at least three vertices. In the former case, we speak of an *inner hole*, and in the latter case, of a *subdivided hole*. A face f of D_{G_1} or D_{G_2} is a *non-hole face* if it is not contained in a subdivided hole. An inner hole H has its *signature*, which is a pair (f_1, f_2) , where f_1 is the unique non-hole face of D_{G_1} containing H , and f_2 is the unique non-hole face of D_{G_2} containing H .³ By the construction, each f_1 appearing in a signature is adjacent to some α_i , and each f_2 is adjacent to some β_j .

In the following claim, we will consider different drawings D'_{G_1} and D'_{G_2} for G_1 and G_2 . By Lemma 1, the faces of D_{G_1} are in one-to-one correspondence with the faces of D'_{G_1} . For a face f_1 of D_{G_1} , we denote the corresponding face by f'_1 , and similarly for a face f_2 of D_{G_2} and f'_2 .

Claim 5. *The graphs G_1 and G_2 as above have planar drawings D'_{G_1} and D'_{G_2} , respectively, that form a simultaneous embedding in which each edge of G_1 crosses each edge of G_2 at most C times, for a suitable constant C ; moreover, D'_{G_1} is directly equivalent to D_{G_1} ; D'_{G_2} is directly equivalent to D_{G_2} ; all hole edges are drawn in the same way in D'_{G_1} and D'_{G_2} ; and whenever (f_1, f_2) is a signature of an inner hole, the interior of the intersection $f'_1 \cap f'_2$ is nonempty.*

We postpone the proof of Claim 5, and we first finish the proof of Theorem 1 assuming this claim.

³ Classifying inner holes according to the signature helps us to obtain a bound independent on the number of holes. Inner holes with same signature are all treated in the same way, independent of their number.

For each inner hole H with signature (f_1, f_2) , we introduce a closed disk B_H in the interior of $f'_1 \cap f'_2$. We require that these disks are pairwise disjoint. In sequel, we consider holes as subsets of S^2 homeomorphic to closed disks (in particular, a hole H intersects \mathcal{M} in ∂H).

Claim 6. *There is an orientation-preserving automorphism φ_1 of S^2 transforming every inner hole H to B_H and D_{G_1} to D'_{G_1} .*

Proof. Using Lemma 1 again, there is an orientation-preserving automorphism ψ_1 transforming D_{G_1} into D'_{G_1} (since D_{G_1} and D'_{G_1} are directly equivalent).

Let f_1 be a face of D_{G_1} . The interior of f'_1 contains images $\psi_1(H)$ of all holes H with signature (f_1, \cdot) , and it also contains the disks B_H for these holes. Therefore, there is a boundary- and orientation-preserving automorphism of f'_1 that maps each $\psi_1(H)$ to B_H .

By composing these automorphisms on every f'_1 separately, we have an orientation-preserving automorphism ψ_2 fixing D'_{G_1} and transforming each $\psi_1(H)$ to B_H . The required automorphism is $\varphi_1 = \psi_2\psi_1$. □

Claim 7. *There is an orientation-preserving automorphism φ_2 of S^2 that fixes hole edges (of subdivided holes), fixes B_H for every inner hole H , and transforms $\varphi_1(D_{G_2})$ to D'_{G_2} .*

Proof. By Lemma 1 there is an orientation-preserving automorphism ψ_3 of S^2 that fixes hole edges and transforms $\varphi_1(D_{G_2})$ to D'_{G_2} .

If an inner hole H has a signature (\cdot, f_2) , then both $\psi_3(B_H)$ and B_H belong to the interior of f'_2 . Therefore, as in the proof of the previous claim, there is an orientation-preserving homeomorphism ψ_4 that fixes D'_{G_2} and transforms $\psi_3(B_H)$ to B_H . We can even require that $\psi_4\psi_3$ is identical on B_H . We set $\varphi_2 := \psi_4\psi_3$. □

To finish the proof of Theorem 1, we set $\varphi = \varphi_1^{-1}\varphi_2\varphi_1$. We need that φ fixes the holes (inner or subdivided) and that $\alpha_1, \dots, \alpha_m$ and $\varphi(\beta_1), \dots, \varphi(\beta_m)$ have $O(mn)$ intersections. It is routine to check all the properties:

If H is a hole (inner or subdivided), then φ_2 fixes $\partial\varphi_1(H)$. Therefore, φ also restricts to a ∂ -automorphism of \mathcal{M} .

The collections of curves $\alpha_1, \dots, \alpha_m$ and $\varphi(\beta_1), \dots, (\beta_m)$ have same intersection properties as the collections $\varphi_1(\alpha_1), \dots, \varphi_1(\alpha_m)$ and $\varphi_2(\varphi_1(\beta_1)), \dots, \varphi_2(\varphi_1(\beta_m))$. Since each α_i and each β_j was subdivided at most three times in the construction, by Claims 5, 6, and 7, these collections have at most $O(mn)$ intersections. The proof of the theorem is finished, except for Claim 5.

Proof of Claim 5. Given G_1 and G_2 , we form auxiliary planar graphs \tilde{G}_1 and \tilde{G}_2 on a vertex set \tilde{V} by contracting all hole edges and removing the resulting loops and multiple edges. We note that a loop cannot arise from an edge that was a part of some α_i or β_j .

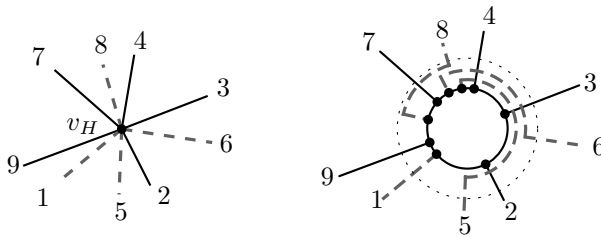
Then we consider planar drawings $D_{\tilde{G}_1}$ and $D_{\tilde{G}_2}$ forming a simultaneous embedding as in Theorem 4, with each edge of \tilde{G}_1 crossing each edge of \tilde{G}_2 at least once and most a constant number of times.

Let $v_H \in \tilde{V}$ be the vertex obtained by contracting the hole edges on the boundary of a hole H . Since the drawings $D_{\tilde{G}_1}$ and $D_{\tilde{G}_2}$ are piecewise linear, in a sufficiently small neighborhood of v_H the edges are drawn as radial segments.

We would like to replace v_H by a small circle and thus turn the drawings $D_{\tilde{G}_1}, D_{\tilde{G}_2}$ into the required drawings D'_{G_1}, D'_{G_2} . But a potential problem is that the edges in $D_{\tilde{G}_1}, D_{\tilde{G}_2}$ may enter v_H in a wrong cyclic order.

We claim that the edges in $D_{\tilde{G}_1}$ entering v_H have the same cyclic ordering around v_H as the corresponding edges around the hole H in the drawing D_{G_1} . Indeed, by contracting the hole edges in the drawing D_{G_1} , we obtain a planar drawing $D_{\tilde{G}_1}^*$ of \tilde{G}_1 in which the cyclic order around v_H is the same as the cyclic order around H in D_{G_1} . Since \tilde{G}_1 was obtained by edge contractions from a maximal planar graph, it is maximal as well (since an edge contraction cannot create a non-triangular face), and its drawing is unique up to an automorphism of S^2 (Lemma 1). Hence the cyclic ordering of edges around v_H in $D_{\tilde{G}_1}$ and in $D_{\tilde{G}_1}^*$ is either the same (if $D_{\tilde{G}_1}$ and $D_{\tilde{G}_1}^*$ are directly equivalent), or reverse (if $D_{\tilde{G}_1}$ and $D_{\tilde{G}_1}^*$ are mirror-equivalent). However, Theorem 4 allows us to choose the drawing $D_{\tilde{G}_1}$ so that it is directly equivalent to $D_{\tilde{G}_1}^*$, and then the cyclic orderings coincide. A similar consideration applies for the other graph G_2 .

The edges of $D_{\tilde{G}_1}$ may still be placed to wrong positions among the edges in $D_{\tilde{G}_2}$, but this can be rectified at the price of at most one extra crossing for every pair of edges entering v_H , as the following picture indicates (the numbering specifies the cyclic order of the edges around H in $D_{G_1} \cup D_{G_2}$):



It remains to draw the edges of G_1 and G_2 that became loops or multiple edges after the contraction of the hole edges. Loops can be drawn along the circumference of the hole, and multiple edges are drawn very close to the corresponding single edge.

In this way, every edge of G_1 still has at most a constant number of intersections with every edge of G_2 , and every two such edges intersect at least once unless at least one of them became a loop after the contraction. Consequently, whenever (f_1, f_2) is a signature of an inner hole, the corresponding faces f'_1 and f'_2 intersect. This finishes the proof. □

Acknowledgement. We would like to thank the authors of [GHR13] for making a draft of their paper available to us, and, in particular, T. Huynh for an e-mail correspondence.

References

- BKR12. Bläsius, T., Kobourov, S.G., Rutter, I.: Simultaneous embedding of planar graphs (2012) (preprint), <http://arxiv.org/abs/1204.5853>
- EK05. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. *J. Graph Algorithms Appl.* 9(3), 347–364 (2005) (electronic)
- FM11. Farb, B., Margalit, D.: *A primer on mapping class groups*. Princeton University Press, Princeton (2011)
- GHR13. Geelen, J., Huynh, T., Richter, R.B.: Explicit bounds for graph minors (May 2013) (preprint), <http://arxiv.org/abs/1305.1451>
- Huy10. Huynh, T.: Removing intersections of curves in surfaces. *Mathoverflow* (2010), <http://mathoverflow.net/questions/33963/removing-intersections-of-curves-in-surfaces/33970#33970>
- JS03. Jaco, W., Sedgwick, E.: Decision problems in the space of Dehn fillings. *Topology* 42(4), 845–906 (2003)
- KM91. Kratochvíl, J., Matoušek, J.: String graphs requiring huge representations. *J. Combin. Theory Ser. B* 53(1), 1–4 (1991)
- KW02. Kaufmann, M., Wiese, R.: Embedding vertices at points: few bends suffice for planar graphs. *J. Graph Algorithms Appl.* 6(1), 115–129 (2002); (electronic) *Graph drawing and representations*, Prague (1999)
- Lic62. Lickorish, W.B.R.: A representation of orientable combinatorial 3-manifolds. *Ann. Math* (2) 76, 531–540 (1962)
- MT01. Mohar, B., Thomassen, C.: *Graphs on Surfaces*. Johns Hopkins University Press, Baltimore (2001)
- MTW11. Matoušek, J., Tancer, M., Wagner, U.: Hardness of embedding simplicial complexes in \mathbb{R}^d . *J. Eur. Math. Soc.* 13(2), 259–295 (2011)
- SSŠ03. Schaefer, M., Sedgwick, E., Štefankovič, D.: Recognizing string graphs in NP. *J. Comput. Syst. Sci.* 67(2), 365–380 (2003)
- Sti80. Stillwell, J.: *Classical Topology and Combinatorial Group Theory*. Springer, New York (1980)

Drawing Permutations with Few Corners

Sergey Bereg¹, Alexander E. Holroyd², Lev Nachmanson², and Sergey Pupyrev^{3,4,*}

¹ Department of Computer Science, University of Texas at Dallas, USA

² Microsoft Research, USA

³ Department of Computer Science, University of Arizona, USA

⁴ Institute of Mathematics and Computer Science, Ural Federal University, Russia

Abstract. A permutation may be represented by a collection of paths in the plane. We consider a natural class of such representations, which we call tangles, in which the paths consist of straight segments at 45 degree angles, and the permutation is decomposed into nearest-neighbour transpositions. We address the problem of minimizing the number of crossings together with the number of corners of the paths, focusing on classes of permutations in which both can be minimized simultaneously. We give algorithms for computing such tangles for several classes of permutations.

1 Introduction

What is a good way to visualize a permutation? In this paper we study drawings in which a permutation of interest is connected to the identity permutation via a sequence of intermediate permutations, with consecutive elements of the sequence differing by one or more non-overlapping nearest-neighbour swaps. The position of each permutation element through the sequence may then be traced by a piecewise-linear path comprising segments that are vertical and 45° to the vertical. Our goal is to keep these paths as simple as possible and to avoid unnecessary crossings.

Such drawings have applications in various fields; for example, in channel routing for integrated circuit design [12]. Another application is the visualization of metro maps and transportation networks, where some lines (railway tracks or roads) might partially overlap [4]. A natural goal is to draw the lines along their common subpaths so that an individual line is easy to follow; minimizing the number of bends of a line and avoiding unnecessary crossings between lines are natural criteria for map readability; see Fig. 3(b) of [3]. Much recent research in the graph drawing community is devoted to edge bundling. In this setting, drawing the edges of a bundle with the minimum number of crossings and bends occurs as a subproblem [10].

Let S_n be the symmetric group of permutations $\pi = [\pi(1), \dots, \pi(n)]$ on $\{1, \dots, n\}$. The **identity permutation** is $[1, \dots, n]$, and the **swap** $\sigma(i)$ transforms a permutation π into $\pi \cdot \sigma(i)$ by exchanging its i th and $(i + 1)$ th elements. Equivalently, $\sigma(i)$ is the transposition $(i, i + 1) \in S_n$, and \cdot denotes composition. Two permutations a and b of S_n are **adjacent** if b can be obtained from a by swaps $\sigma(p_1), \sigma(p_2), \dots, \sigma(p_k)$ that are not overlapping, that is, such that $|p_i - p_j| \geq 2$ for $i \neq j$. A **tangle** is a finite sequence

* Research supported in part by NSF grant DEB 1053573.

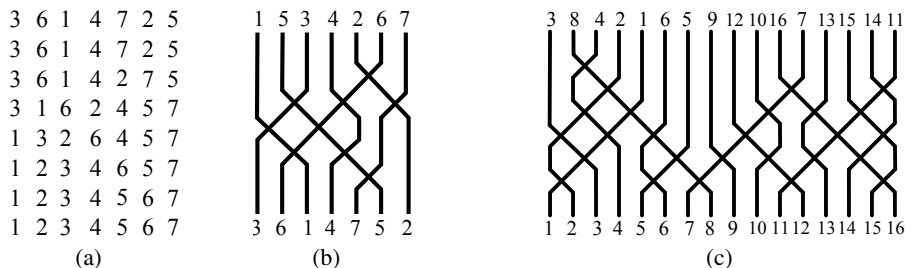


Fig. 1. (a) A tangle solving the permutation $[3, 6, 1, 4, 7, 2, 5]$. (b) A drawing of the tangle. (c) An example of a perfect tangle drawing.

of permutations in which each two consecutive permutations are adjacent. An example of a tangle is given in Fig. 1. The associated drawing is composed of polylines with vertices in \mathbb{Z}^2 , whose segments can be vertical, or have slopes of $\pm 45^\circ$ to the vertical. The polyline traced by element $i \in \{1, \dots, n\}$ is called **path** i . Note that by definition all path crossings occur at right angles. We say that a tangle T **solves** the permutation π (or simply T is a tangle for π) if the tangle starts from π and ends at the identity permutation.

We are interested in tangles with informative and aesthetically pleasing drawings. Our main criterion is to keep the paths straight by using only a few turns. A **corner** of path i is a point at which it changes its direction from one of the allowed directions (vertical, $+45^\circ$, or -45°) to another. A change between $+45^\circ$ and -45° is called a **double corner**. We are interested in the total number of corners of a tangle, where corners are always counted with multiplicity (so a double corner contributes 2 to the total). By convention we require that paths start and end with vertical segments. In terms of the sequence of permutations this means repeating the first and the last permutations at least once each as in Fig. 1(a).

Another natural objective is to minimize path crossings. We call a tangle for π **simple** if it has the minimum number of crossings among all tangles for π . This is equivalent to the condition that no pair of paths cross each other more than once, and this minimum number equals the *inversion number* of π . A simple tangle has no double corner since that would entail an immediate double crossing of a pair of paths.

In general, minimizing corners and minimizing crossings are conflicting goals. For example, let $n = 4k$ and $k \geq 4$ and consider the permutation

$$\pi = [2k, 3, 2, 5, 4, \dots, 2k-1, 2k-2, 1, \quad 4k, 2k+3, 2k+2, \dots, 4k-1, 4k-2, 2k+1].$$

It is not difficult to check that the minimum number of corners in a tangle for π is $4n - 8$, while the minimum among simple tangles is $5n - 20$, which is strictly greater; see Fig. 2 for the case $k = 4$. Our focus in this article is on two special classes of permutations for which corners and crossings can be minimized simultaneously. The first is relatively straightforward, while the second turns out to be much more subtle.

One may ask the following interesting question. Is there an efficient algorithm for *finding a (simple) tangle with the minimum number of corners solving a given permutation?*

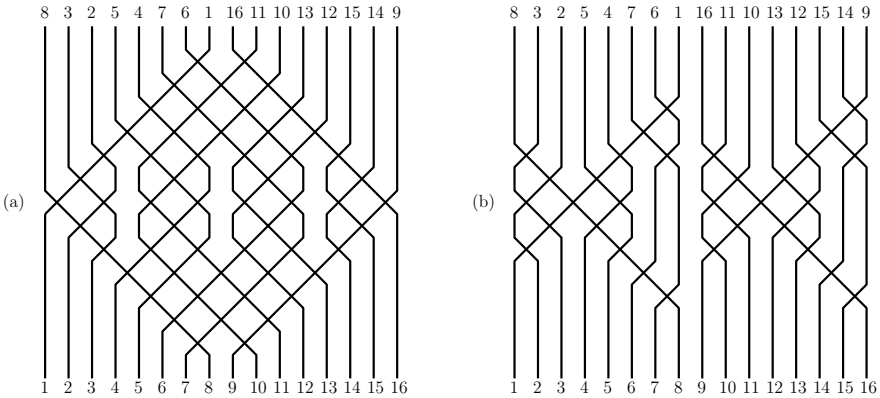


Fig. 2. (a) A tangle with 56 corners. (b) Every *simple* tangle for the same permutation has at least 60 corners.

We do not know whether there is a polynomial-time algorithm, either with or without the requirement of simplicity. Here we present polynomial-time exact algorithms for special classes of permutations.

Even the task of determining whether a given tangle has the minimum possible number of corners among tangles for its permutation does not appear to be straightforward in general (and likewise if we restrict to simple tangles). However, in certain cases, such minimality is indeed evident, and we focus on two such cases. Firstly, we call a tangle **direct** if each of its paths has at most 2 corners (equivalently, at most one non-vertical segment). Note that a direct tangle is simple. Furthermore, it clearly has the minimum number of corners among all tangles (simple or otherwise) for its permutation.

We can completely characterize permutations admitting direct tangles. We say that a permutation $\pi \in S_n$ **contains** a pattern $\mu \in S_k$ if there are integers $1 \leq i_1 < i_2 < \dots < i_k \leq n$ such that for all $1 \leq r < s \leq k$ we have $\pi(i_r) < \pi(i_s)$ if and only if $\mu(r) < \mu(s)$; otherwise, π **avoids** the pattern (or π is μ -avoiding).

Theorem 1. *A permutation has a direct tangle if and only if it is 321-avoiding.*

Our proof yields a straightforward algorithm that constructs a direct tangle for a given 321-avoiding permutation.

Our second special class of tangles naturally extends the notion of a direct tangle, but turns out to have a much richer theory. A **segment** is a straight line segment of a path between two of its corners; it is an **L-segment** if it is oriented from north-east to south-west, and an **R-segment** if it is oriented from north-west to south-east. We call a tangle **perfect** if it is simple and each of its paths has at most one L-segment and at most one R-segment. Any perfect tangle has the minimum possible number of corners among all tangles solving its permutation, and indeed it has the minimum possible corners on path i for each $i = 1, \dots, n$. To see this, note that if i has an L-segment in a perfect tangle for π then there must be an element $j > i$ with $\pi(i) > \pi(j)$, whose path crosses this L-segment. Hence, an L-segment must be present in any tangle for π . The same argument applies to R-segments. We call a permutation **perfect** if it has a perfect tangle.

Theorem 2. *There exists a polynomial-time algorithm that determines whether a given permutation is perfect and, if so, outputs a perfect tangle.*

A straightforward implementation of our algorithm takes $O(n^5)$ time, but we believe this can be reduced to $O(n^3)$, and possibly further. Our proof of Theorem 2 involves an explicit characterization of perfect permutations, but it is considerably more complicated than in the case of direct tangles. We introduce the notion of a *marking*, which is an assignment of symbols to the elements $1, \dots, n$ indicating the directions in which their paths should be routed. We prove that a permutation is perfect if and only if it admits a marking satisfying a *balance* condition that equates numbers of elements in various categories. Finally, we show that the existence of such a marking can be decided by finding a maximum vertex-weighted matching in a certain graph with vertex set $1, \dots, n$ constructed from the permutation.

The number of perfect permutations in S_n grows only exponentially with n (see Section 4), and is therefore $o(|S_n|)$. Nonetheless, perfect permutations are very common for small n : all permutations in S_6 are perfect, as are all but 16 in S_7 , and over half in S_{13} .

Related Work. We are not aware of any other study on the number of corners in a tangle. To the best of our knowledge, the problem formulated here is new. Wang in [12] considered the same model of drawings in the field of VLSI design. However, [12] targets, in our terminology, the tangle height and the total length of the tangle paths. The heuristic suggested by Wang produces paths with many unnecessary corners.

The perfect tangle problem is related to the problem of drawing graphs in which every edge is represented by a polyline with few bends. In our setting, all the crossings occur at right angles, as in so-called RAC-drawings [6].

Decomposition of permutations into nearest-neighbour transpositions was considered in the context of permuting machines and pattern-restricted classes of permutations [1]. In our terminology, Albert et. al. [1] proved that it is possible to check in polynomial time whether for a given permutation there exists a tangle of length k (that is, consisting of k permutations), for a given k . Tangle diagrams appear in the drawings of sorting networks [8,2]. We also mention an interesting connection with change ringing (English-style church bell ringing), where similar visualizations are used [13].

2 Preliminaries

We always draw tangles oriented downwards with the sequence of permutations read from top to bottom as in Fig. 1(b). The following notation will be convenient. We write $\pi = [\dots a \dots b \dots c \dots]$ to mean that $\pi^{-1}(a) < \pi^{-1}(b) < \pi^{-1}(c)$, and $\pi = [\dots ab \dots]$ to mean that $\pi^{-1}(a) + 1 = \pi^{-1}(b)$. A pair of elements (a, b) is an **inversion** in a permutation $\pi \in S_n$ if $a > b$ and $\pi = [\dots a \dots b \dots]$. The **inversion number** $\text{inv}(\pi) \in [0, \binom{n}{2}]$ is the number of inversions of π . The following useful lemma is straightforward to prove.

Lemma 1. *In a simple tangle for permutation π , a pair (i, j) is an inversion in π if and only if some R-segment of path i intersects some L-segment of path j .*

3 Direct Tangles

Here we prove Theorem 1. We need two properties of 321-avoiding permutations.

Lemma 2. *Suppose π, π' are permutations with $\text{inv}(\pi') = \text{inv}(\pi) - 1$ and $\pi' = \pi \cdot \sigma(i)$ for some swap $\sigma(i)$. If π is 321-avoiding then so is π' .*

Proof. Let us suppose that elements i, j, k form a 321-pattern in π' . Then (i, j) and (j, k) are inversions in π' . Inversions of π' are inversions of π , hence, elements i, j, k form a 321-pattern in π . □

Lemma 3. *In a simple tangle solving a 321-avoiding permutation, no path has both an L-segment and an R-segment.*

Proof. Consider a simple tangle solving a 321-avoiding permutation π . Suppose path j crosses path i during j 's R-segment and crosses path k during j 's L-segment. By Lemma 1 we have $\pi = [\dots k \dots j \dots i \dots]$ while $i < j < k$, giving a 321-pattern, which is a contradiction. □

We say that a permutation $\pi \in S_n$ has a **split** at location k if $\pi(1), \dots, \pi(k) \in \{1, \dots, k\}$, or equivalently if $\pi(k + 1), \dots, \pi(n) \in \{k + 1, \dots, n\}$.

Theorem 1. *A permutation has a direct tangle if and only if it is 321-avoiding.*

Proof. To prove the “only if” part, suppose that tangle T solves a permutation π containing a 321-pattern. Then there are $i < j < k$ with $\pi = [\dots k \dots j \dots i \dots]$. Hence by Lemma 1, j has an L-segment and an R-segment, so T is not direct.

We prove the “if” part by induction on the inversion number of the permutation. If $\text{inv}(\pi) = 0$ then π is the identity permutation, which clearly has a direct tangle. This gives us the basis of induction.

Now suppose that π is 321-avoiding and not the identity permutation, and that every 321-avoiding permutation (of every size) with inversion number less than $\text{inv}(\pi)$ has a direct tangle. There exists s such that $\pi(s) > \pi(s + 1)$; fix one such. Note that $(\pi(s), \pi(s + 1))$ is an inversion of π ; hence, the permutation $\pi' := \pi \cdot \sigma(s)$ has $\text{inv}(\pi') = \text{inv}(\pi) - 1$, and is also 321-avoiding by Lemma 2. By the induction hypothesis, let T' be a direct tangle solving π' .

Perform a swap x in position s exchanging elements $\pi(s)$ and $\pi(s + 1)$, and draw it as a cross on the plane with coordinates (s, h) , where $h \in \mathbb{Z}$ is the height (y -coordinate) of the cross (chosen arbitrarily). We assume that the position axis increases from left to right and the height axis increases from bottom to top. Then draw the tangle T' below the cross. This gives a tangle solving π , which is certainly simple. We show that the heights of swaps may be adjusted to make the new tangle direct. To achieve this, the L-segment and R-segment comprising the swap x must either extend existing segments in T' , or must connect to vertical paths having no corners in T' . Consider two cases.

Case 1: Suppose that π' has a split at s . Then T' consists of a tangle T_1 for the permutation $[\pi'(1), \dots, \pi'(s)]$ together with another tangle T_2 for $[\pi'(s + 1), \dots, \pi'(n)]$; see Fig. 3. Starting with T_1 drawn below x , simultaneously shift all the swaps of T_1 upward until one of them touches x ; in other words, until T_1 's first swap in position

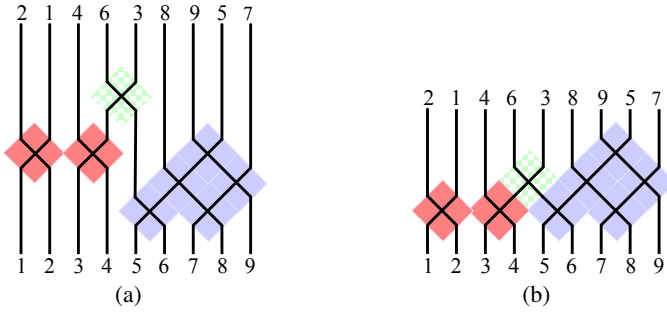


Fig. 3. Shifting two sub-tangles (T_1 is red, T_2 is blue) upward to touch the initial swap x (green) in position $s = 4$.

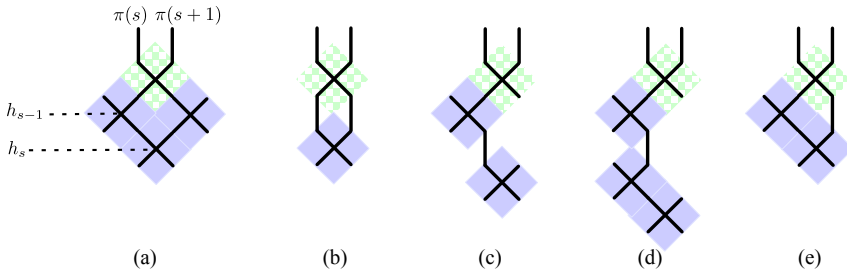


Fig. 4. (a) The tangle T' (blue) touches the swap x (green) on both sides. (b)–(e) Various impossible configurations for the proof.

$s - 1$ occurs at height $h - 1$. Or, if T_1 has no swap in position $s - 1$, no shifting is necessary. Similarly shift T_2 upward until it touches x from the right side. This results in a direct tangle.

Case 2: Suppose that π' has no split at s . Let T' be any direct tangle for π' , and again shift it upward until it touches x , resulting in a tangle T for π . Write h_j for the height of the topmost swap in position j in T' , or let $h_j = -\infty$ if there is none. We claim that $h_{s-1} = h_{s+1} = h_s + 1 > -\infty$, which implies in particular that $1 < s < s + 1 < n$. Thus T' has swaps in the positions immediately left and right of x , both of which touch x simultaneously in the shifting procedure as in Fig. 4(a), giving that T is direct as required. To prove the claim, first note that $h_s > -\infty$ since π' has no split at s . Therefore $\max\{h_{s-1}, h_{s+1}\} > h_s$, otherwise T would not be simple, as in Fig. 4(b). Thus, without loss of generality suppose that $h_{s-1} > h_s$ and $h_{s-1} \geq h_{s+1}$. Then $h_s = h_{s-1} - 1$, otherwise some path would have more than 2 corners in T' , specifically, the path of the element that is in position s after h_{s-1} ; see Fig. 4(c) or (d). Now suppose for a contradiction that $h_{s+1} < h_{s-1}$, which includes the possibility that $h_{s+1} = -\infty$, perhaps because $s + 1 = n$. Then in the new tangle T , path $\pi(s)$ contains both an L-segment and an R-segment as in Fig. 4(e), which contradicts Lemma 3. \square

The proof of Theorem 1 yields an algorithm that returns a direct tangle for $\pi \in S_n$ if one exists, and otherwise stops. The algorithm can be implemented so as to run in $O(n^2)$ time. With a suitable choice of output format, this can be improved to $O(n)$.

4 Perfect Tangles

In this section we give our characterization of perfect permutations. Given a permutation $\pi \in S_n$, we introduce the following classification scheme of elements $i \in \{1, \dots, n\}$. The scheme reflects the possible forms of paths in a perfect tangle, although the definitions themselves are purely in terms of the permutation. We call i a **right** element if it appears in some inversion of the form (i, j) , and a **left** element if it appears in some inversion (j, i) . We call i **left-straight** if it is left but not right, **right-straight** if it is right but not left, and a **switchback** if it is both left and right.

In order to build a perfect tangle we use a notion of marking. A **marking** M is a function from the set $\{1, \dots, n\}$ to strings of letters L and R . For any tangle T , we associate a corresponding marking M as follows. We trace the path i from top to bottom; as we meet an L-segment (resp. R-segment), we append an L (resp. R) to $M(i)$. Vertical segments are ignored for this purpose; hence, a vertical path with no corners is marked by an empty sequence \emptyset . For example, $M(3) = R$ and $M(13) = LR$ in Fig. 1(c). A marking corresponding to a perfect tangle takes only values \emptyset, L, R, LR , and RL . We write $M(i) = R\dots$ to indicate that the string $M(i)$ starts with R .

Given a permutation π and a marking M , there does not necessarily exist a corresponding tangle. However, we will obtain a necessary and sufficient condition on π and M for the existence of a corresponding perfect tangle. Our strategy for proving Theorem 2 will be to find a marking satisfying this condition, and then to find a corresponding perfect tangle. We say that a marking M is a **marking for** a permutation $\pi \in S_n$ if (i) $M(i) = L$ (respectively $M(i) = R$) for all left-straight (right-straight) elements i , (ii) $M(i) \in \{LR, RL\}$ for all switchbacks, and (iii) $M(i) = \emptyset$ otherwise.

To state the necessary and sufficient condition mentioned above, we need some definitions. A quadruple (a, b, c, d) is a **rec** in permutation π if $\pi = [\dots a \dots b \dots c \dots d \dots]$ and $\min\{a, b\} > \max\{c, d\}$. In a perfect tangle, the paths comprising a rec form a rectangle; see Fig. 5 (“rec” is an abbreviation for rectangle). Let M be a marking for $\pi \in S_n$, and let ρ be a rec (a, b, c, d) in π . We call e a **left switchback** of ρ if (i) $M(e) = RL$, (ii) $\pi = [\dots a \dots e \dots b \dots]$, and (iii) $c < e < d$ or $d < e < c$. Symmetrically, we call e a **right switchback** of ρ if $M(e) = LR$, and $\pi = [\dots c \dots e \dots d \dots]$, and $a < e < b$ or $b < e < a$. A rec (a, b, c, d) is **regular** if $a < b$ and $c < d$, otherwise it is **irregular**. A rec is called **balanced** under M if the number of its left switchbacks is equal to the number of its right switchbacks; a rec is **empty** if it has no switchbacks.

Here is our key definition. A marking M for a permutation π is called **balanced** if every regular rec of π is balanced and every irregular rec is empty under M .

Theorem 3. *A permutation is perfect if and only if it admits a balanced marking.*

The proof of Theorem 3 is technical, see full version for the complete proof [5].

Any permutation containing the pattern [7324651] (for example) is not perfect since 4 must be a switchback of one of the irregular recs (7321) and (7651). It follows by [9, 7]

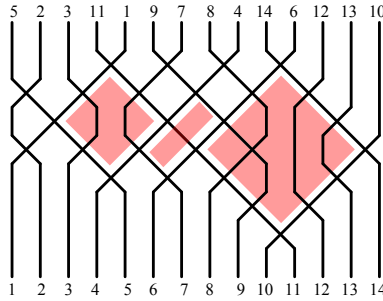


Fig. 5. A permutation with a balanced marking. Some of the recs of the permutation are: $\rho_1 = (5, 11, 1, 4)$, $\rho_2 = (9, 7, 4, 6)$, $\rho_3 = (11, 14, 6, 10)$; ρ_1 and ρ_3 are regular, while ρ_2 is irregular. Left switchbacks of rec ρ_3 are 8 and 9, right switchbacks are 12 and 13. The empty irregular rec ρ_2 has neither left nor right switchbacks.

that the number of perfect permutations in S_n is at most C^n for some constant $C > 1$. Since direct tangles are perfect, it also follows from Theorem 1 that the number is at least c^n for some constant $c > 1$.

We note that Theorem 3 already yields an algorithm for determining whether a permutation is perfect in $\tilde{O}(2^n)$ time¹ by checking all markings. In Section 5 we improve this to polynomial time.

5 Recognizing Perfect Permutations

We provide an algorithm for recognizing perfect permutations. The algorithm finds a balanced marking for a permutation, or reports that such a marking does not exist. We start with a useful lemma.

Lemma 4. *Fix a permutation. For each right (resp., left) element a there is a left-straight (right-straight) b such that the pair (a, b) (resp., (b, a)) is an inversion.*

Proof. We prove the case when a is right, the other case being symmetrical. Consider the minimal b such that (a, b) is an inversion. By definition, b is left. Suppose that it is also a right element, that is, (b, c) is an inversion for some $c < b$. It is easy to see that (a, c) is an inversion too, which contradicts to the minimality of b . □

Recall that a marking is balanced only if (in particular) every regular rec of the permutation is balanced under the marking. We show that this is guaranteed even by balancing of recs of a restricted kind. We call a rec (a, b, c, d) of a permutation π **straight** if a, b, c , and d are straight elements of π . A marking is called **s-balanced** if every straight rec is balanced and every irregular rec is empty under the marking.

Lemma 5. *Let M be a marking of a permutation π . Then M is balanced if and only if it is s-balanced.*

¹ \tilde{O} hides a polynomial factor.

Proof. The “if” direction is immediate, so we turn to the converse. Let M be an s -balanced marking and $\rho = (a, b, c, d)$ be a regular rec of π . We need to prove that ρ is balanced under M . If ρ is straight then ρ is balanced by definition. Let us suppose that ρ is not straight. Then some $u \in \{a, b, c, d\}$ is not a straight element. Our goal is to show that it is possible to find a new rec ρ' in which u is replaced with a straight element so that the sets of left and right switchbacks of ρ and ρ' coincide. By symmetry, we need only consider the cases $u = a$ and $u = b$.

Case $u = a$: Let us suppose that a is not straight. By Lemma 4, there exists a right straight e such that (e, a) is an inversion. Let us denote $\rho' = (e, b, c, d)$ and show that ρ' has the same switchbacks as ρ . Let k be a left switchback of ρ ; then $M(k) = RL$, and $\pi = [\dots e \dots a \dots k \dots b \dots]$, and $c < k < d$. By definition k is a left switchback of ρ' . Let k be a left switchback of ρ' . If $\pi = [\dots e \dots k \dots a \dots b \dots]$ then the irregular rec (e, a, c, d) has a left switchback, which is impossible. Therefore, $\pi = [\dots e \dots a \dots k \dots b \dots]$ and k is a left-switchback of ρ .

Let us suppose that k is a right switchback of ρ , so $a < k < b$. If $k < e$ then k is a right switchback of the irregular (e, a, c, d) ; hence, $e < k < b$ and k is a right switchback of ρ' . On the other hand, if k is a right switchback of ρ' then $a < e < k < b$, which means that k is a right switchback of ρ .

Case $u = b$: Let us suppose that b is not straight. By Lemma 4, there exists a right straight e such that (e, b) is an inversion. Let us denote $\rho' = (a, e, c, d)$ and show that ρ' has the same switchbacks as ρ . Let k be a left switchback of ρ . We have $\pi = [\dots a \dots k \dots b \dots]$. Since k is not a left switchback of the irregular rec (e, b, c, d) , we have $\pi = [\dots a \dots k \dots e \dots]$. Therefore, k is a left switchback of ρ' .

Let k be a right switchback of ρ . Then $a < k < b < e$, proving that k is a right switchback of ρ' . Let k be a right switchback of ρ' . If $b < k$ then k is a right switchback of (e, b, c, d) , which is impossible. Then $k < b$ and k is a right switchback of ρ . \square

We can restrict the set of recs guaranteeing the balancing of a permutation even further. We call a pair a, b of elements **right** (resp. **left**) **minimal** if a and b are right (left) straight elements of π , and $a < b$, and there is no right (left) straight element c such that $\pi = [\dots a \dots c \dots b \dots]$. We call rec $\rho = (a, b, c, d)$ **minimal** in π if a, b is a right minimal pair and c, d is a left minimal pair; see Fig. 6(a). We call a marking for a permutation **ms-balanced** if every minimal regular rec is balanced and every irregular rec is empty under the marking.

Lemma 6. *Let M be a marking of a permutation π . Then M is s -balanced if and only if it is ms-balanced.*

Before giving the proof, we introduce some further notation. Let $\rho = (a, b, c, d)$ be an arbitrary, possibly irregular, rec in π . Let us denote by ρ_ℓ (resp. ρ_r) the set of switchbacks i that can under some marking be left (resp., right) switchbacks of ρ . Formally, $i \in \rho_\ell$ if and only if $\pi = [\dots a \dots i \dots b \dots c \dots d \dots]$ and either $c < i < d$ or $d < i < c$. (And ρ_r is defined symmetrically.) For a rec ρ and marking M let ρ_ℓ^M (ρ_r^M) be the set of left (respectively, right) switchbacks of ρ under M . Of course, $\rho_\ell^M \subseteq \rho_\ell$ and $\rho_r^M \subseteq \rho_r$. It is easy to see from the definition that for two different minimal recs ρ and ρ' we have $\rho_\ell \cap \rho'_\ell = \emptyset$ and $\rho_r \cap \rho'_r = \emptyset$.

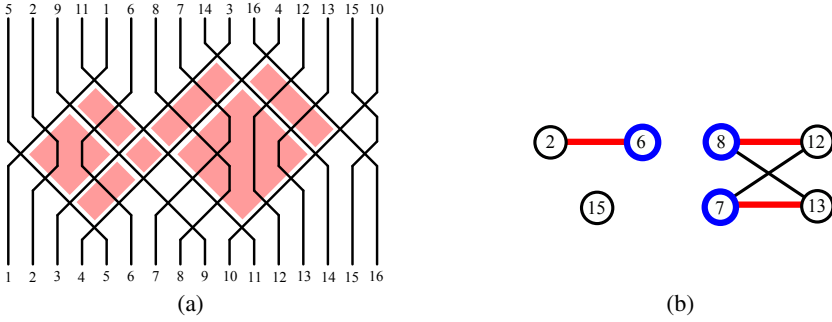


Fig. 6. (a) A perfect tangle for a permutation with 7 minimal straight recs (shown red). (b) The graph constructed in *Step 3* of our algorithm. Here, $\mathcal{J}_\ell = \emptyset$, $\mathcal{J}_r = \{6\}$, $\mathfrak{R}_\ell = \{6, 7, 8, 12, 13, 15\}$, and $\mathfrak{R}_r = \{2, 7, 8\}$. The vertices of the set $F = \{6, 7, 8\}$ are shown blue. The red edges are the computed maximum matching.

Proof (Lemma 6). It suffices to prove that if M is ms-balanced then it is s-balanced. Consider a straight rec $\rho = (a, b, c, d)$. Let $a = r_1, \dots, r_p = b$ be a sequence of right straights in which each consecutive pair r_i, r_{i+1} is right minimal. Define left straights $c = \ell_1, \dots, \ell_q = d$ similarly. Let D be the set of all recs of the form $(r_i, r_{i+1}, \ell_j, \ell_{j+1})$ for $1 \leq i < p$ and $1 \leq j < q$. Notice that all recs of D are minimal. By definition of rec switchbacks, we have $\rho_\ell^M = \bigcup_{u \in D} u_\ell^M$ and $\rho_r^M = \bigcup_{u \in D} u_r^M$. Since every rec $u \in D$ is balanced and for every pair $u, v \in D$ of different recs $u_\ell^M \cap v_\ell^M = u_r^M \cap v_r^M = \emptyset$, we have $|\rho_\ell^M| = |\rho_r^M|$; that is, ρ is balanced under M . \square

Let us show how to construct an ms-balanced marking. For a permutation π , let $\mathcal{J}_\ell = \bigcup\{\rho_\ell : \rho \text{ is an irregular rec in } \pi\}$ and $\mathfrak{R}_\ell = \bigcup\{\rho_\ell : \rho \text{ is a regular rec in } \pi\}$, and define $\mathcal{J}_r, \mathfrak{R}_r$ similarly. Our algorithm is based on finding a maximum vertex-weighted matching, which can be done in polynomial time [11].

The algorithm inputs a permutation π and computes an ms-balanced marking M for π or determines that such a marking does not exist. Initially, $M(i)$ is undefined for every $i \in \{1, \dots, n\}$. The algorithm has the following steps.

Step 1: For every element $1 \leq i \leq n$ that is neither left nor right, set $M(i) = \emptyset$. For every left straight i set $M(i) = L$. For every right straight i set $M(i) = R$.

Step 2: If $\mathcal{J}_\ell \cap \mathcal{J}_r \neq \emptyset$ then report that π is not perfect and stop. Otherwise, for every switchback $i \in \mathcal{J}_\ell$ set $M(i) = LR$; for every switchback $i \in \mathcal{J}_r$ set $M(i) = RL$.

Step 3.1: Build a directed graph $G = (V, E)$ with $V = \mathfrak{R}_\ell \cup \mathfrak{R}_r$ and $E = \bigcup\{(\rho_\ell \setminus \mathcal{J}_\ell) \times (\rho_r \setminus \mathcal{J}_r) : \rho \text{ is a minimal rec in } \pi\}$.

Step 3.2: Create a set $F \leftarrow (\mathfrak{R}_\ell \cap \mathfrak{R}_r) \cup (\mathcal{J}_\ell \cap \mathfrak{R}_r) \cup (\mathcal{J}_r \cap \mathfrak{R}_\ell)$. Create weights w for vertices of G : if $i \in F$ then set $w(i) = 1$, otherwise set $w(i) = 0$.

Step 4: Compute a maximum vertex-weighted matching U on G (viewed as an *unoriented* graph, ignoring the directions of edges) using weights w . If the total weight of U is less than $|F|$ then report that π is not perfect and stop.

Step 5.1: Assign marking based on the matching: for every edge $(i, j) \in U$ set $M(i) = RL$ provided $M(i)$ has not already been assigned, and $M(j) = LR$ provided $M(j)$ has not already been assigned.

Step 5.2: For every switchback $1 \leq i \leq n$ with still undefined marking, if $i \in \mathfrak{R}_\ell$ then set $M(i) = LR$, if $i \in \mathfrak{R}_r$ then $M(i) = RL$, otherwise choose $M(i)$ to be LR or RL arbitrarily. Note that any $i \in \mathfrak{R}_\ell \cap \mathfrak{R}_r$ was already assigned because of Steps 3.2 and 4.

Let us prove the correctness of the algorithm.

Lemma 7. *If the algorithm produces a marking then the marking is ms-balanced.*

Proof. Let M be a marking produced by the algorithm for a permutation π . It is easy to see that $M(i)$ is defined for all $1 \leq i \leq n$ (in Step 1 for straights and in Step 2 and Step 5 for switchbacks). By construction, M is a marking for π .

Let us show that M is ms-balanced. Consider an irregular rec ρ of π , and suppose that $i \in \rho_\ell$. Since $\rho_\ell \subseteq \mathfrak{J}_\ell$, in Step 2 we assign $M(i) = LR$, that is, $i \notin \rho_\ell^M$. Therefore, ρ does not have left switchbacks under M . Similarly, ρ does not have right switchbacks under M . Therefore, ρ is empty.

Consider a regular minimal straight rec ρ in π . Suppose that $i \in \rho_\ell^M$. Then $M(i) = RL$ and $i \in \rho_\ell \subseteq \mathfrak{R}_\ell$. If $i \in \mathfrak{J}_r$ then $i \in \mathfrak{R}_\ell \cap \mathfrak{J}_r \subseteq F$; hence i is incident to an edge in U . Since no directed edge of the form (k, i) is included in G in Step 3.1, there exists $(i, k) \in U$ for some k . On the other hand, if $i \notin \mathfrak{J}_r$ then string RL was not assigned to $M(i)$ in Step 5.2, nor in Step 2. Thus, it was assigned in Step 5.1, and again $(i, k) \in U$ for some k . By definition of E we have $k \in \rho_r$, because k cannot appear in ρ'_r for any other minimal $\rho' \neq \rho$. The algorithm sets $M(k) = LR$ at Step 5.1; it could not have previously set $M(k) = LR$ at Step 2 because $k \notin \mathfrak{J}_r$ by the definition of E . Thus $k \in \rho_r^M$.

By symmetry, an identical argument to the above shows that if $k \in \rho_r^M$ then $i \in \rho_\ell^M$ for some i satisfying $(i, k) \in U$. Since U is a matching, we thus have a bijection between elements of ρ_ℓ^M and ρ_r^M . Therefore, ρ is balanced under M . \square

Lemma 8. *Let π be a perfect permutation. The algorithm produces a marking for π .*

Proof. Since π is perfect, there is a balanced marking M for π . Since M is balanced, all irregular recs are empty under M ; hence, the algorithm does not stop in Step 2. To prove the claim, we will create a matching in the graph G with total weight $|F|$.

Let ρ be a minimal rec in π . Since ρ is balanced under M , we have $|\rho_\ell^M| = |\rho_r^M|$. Hence, let W_ρ be an arbitrary matching connecting vertices of $|\rho_\ell^M|$ with vertices of $|\rho_r^M|$. Of course, $|W_\rho| = |\rho_\ell^M|$. Let $W = \bigcup \{W_\rho : \rho \text{ is a minimal rec in } \pi\}$. We show that every element of set F is incident to an edge of W .

Suppose $i \in \mathfrak{R}_\ell \cap \mathfrak{R}_r$. Since i is a switchback in π , we have $M(i) = RL$ or $M(i) = LR$. In the first case $i \in \rho_\ell^M$ and in the second case $i \in \rho_r^M$ for some minimal rec ρ . Then i is incident to an edge from W_ρ .

Suppose $i \in F \setminus \{\mathfrak{R}_\ell \cap \mathfrak{R}_r\}$. Without loss of generality, let $i \in \mathfrak{J}_\ell \cap \mathfrak{R}_r$. Since M is balanced, every irregular rec has no switchbacks and hence $M(i) = LR$. Thus, $i \in \rho_r^M$ for some minimal rec ρ , and i is incident to an edge of W_ρ .

Therefore, every vertex of F is incident to an edge of the matching W , which means that the total weight of W is $|F|$. \square

Theorem 2 follows directly from Lemmas 7 and 8 and Theorem 3. A straightforward implementation of the algorithm finding a perfect tangle takes $O(n^5)$ time.

6 Conclusion

In this paper we gave algorithms for producing optimal tangles in the special cases of direct and perfect tangles, and for recognizing permutations for which this is possible. Many questions remain open. What is the complexity of determining the tangle with minimum corners for a given permutation? What is the complexity if the tangle is required to be simple? What is the asymptotic behavior of the maximum over permutations $\pi \in S_n$ of the minimum number of corners among simple tangles solving π ?

Acknowledgments: We thank Omer Angel, Franz Brandenburg, David Eppstein, Martin Fink, Michael Kaufmann, Peter Winkler, and Alexander Wolff for fruitful discussions about variants of the problem.

References

1. Albert, M.H., Aldred, R.E.L., Atkinson, M., van Ditmarsch, H.P., Handley, C.C., Holton, D.A., McCaughan, D.J.: Compositions of pattern restricted sets of permutations. *Australian J. Combinatorics* 37, 43–56 (2007)
2. Angel, O., Holroyd, A.E., Romik, D., Virag, B.: Random sorting networks. *Advances in Mathematics* 215(2), 839–868 (2007)
3. Argyriou, E.N., Bekos, M.A., Kaufmann, M., Symvonis, A.: On metro-line crossing minimization. *Journal of Graph Algorithms and Applications* 14(1), 75–96 (2010)
4. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In: Kaufmann, M., Wagner, D. (eds.) *GD 2006*. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
5. Bereg, S., Holroyd, A.E., Nachmanson, L., Pupyrev, S.: Drawing permutations with few corners. *ArXiv e-print abs/1306.4048* (2013)
6. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. *Theoretical Computer Science* 412(39), 5156–5166 (2011)
7. Klazar, M.: The Füredi-Hajnal conjecture implies the Stanley-Wilf conjecture. In: Krob, D., Mikhalev, A., Mikhalev, A. (eds.) *Formal Power Series and Algebraic Combinatorics*, pp. 250–255. Springer, Heidelberg (2000)
8. Knuth, D.: *The art of computer programming*. Addison-Wesley (1973)
9. Marcus, A., Tardos, G.: Excluded permutation matrices and the Stanley-Wilf conjecture. *Journal of Combinatorial Theory, Series A* 107(1), 153–160 (2004)
10. Pupyrev, S., Nachmanson, L., Bereg, S., Holroyd, A.E.: Edge routing with ordered bundles. In: van Kreveld, M.J., Speckmann, B. (eds.) *GD 2011*. LNCS, vol. 7034, pp. 136–147. Springer, Heidelberg (2012)
11. Spencer, T.H., Mayr, E.W.: Node weighted matching. In: Paredaens, J. (ed.) *ICALP 1984*. LNCS, vol. 172, pp. 454–464. Springer, Heidelberg (1984)
12. Wang, D.C.: Novel routing schemes for IC layout, part I: Two-layer channel routing. In: *28th ACM/IEEE Design Automation Conference*, pp. 49–53 (1991)
13. White, A.T.: Ringing the changes. *Mathematical Proceedings of the Cambridge Philosophical Society* 94, 203–215 (1983)

Dynamic Traceroute Visualization at Multiple Abstraction Levels*

Massimo Candela, Marco Di Bartolomeo,
Giuseppe Di Battista, and Claudio Squarcella

Dipartimento di Ingegneria, Università Roma Tre, Italy
{candela,dibartolomeo,gdb,squarcel}@dia.uniroma3.it

Abstract. We present a system, called TPLAY, for the visualization of the traceroutes performed by the Internet probes deployed by active measurement projects. These traceroutes are continuously executed towards selected Internet targets. TPLAY allows to look at traceroutes at different abstraction levels and to animate the evolution of traceroutes during a selected time interval. The system has been extensively tested on traceroutes performed by RIPE Atlas [22] Internet probes.

1 Introduction

The *traceroute* command is one of the most popular computer network diagnostic tools. It can be used on computers connected to the Internet to compute the path (route) towards a given IP address, also called *traceroute path*. It is probably the simplest tool to gain some knowledge on the Internet topology. Because of its simplicity and effectiveness, it attracted the interest of several researchers that developed services for visualizing the Internet paths discovered by executing one or more traceroute commands.

Broadly speaking, there are two groups of traceroute visualization systems: tools developed for local technical debugging purposes and tools that aim at reconstructing and displaying large portions of the Internet topology. Several tools of the first group visualize a single traceroute on a map, showing the geo-location of the traversed routers. A few examples follow. Xtraceroute [10] is a graphical version of the traceroute program. It displays individual routes on an interactive rotating globe as a series of yellow lines between sites, shown as small spheres of different colors. GTrace [20] and Visual-Route [30] are traceroute and network diagnostic tools that provide a 2D geographical visualization of paths. The latter also features more abstract representations taking into account other information, e.g. the round-trip time between intermediate hops. In the second group there are several tools (see e.g. [18,5]) that merge the paths generated by multiple traceroutes into directed graphs and show them in some type of drawing.

In recent years the visualization of Internet measurements has seen a growing interest. This is mainly due to the existence of several projects that deploy *probes* in the Internet. Probes are systems that perform traceroutes and other measurements (e.g. ping,

* Partially supported by the ESF project 10-EuroGIGA-OP-003 GraDR "Graph Drawings and Representations" and by the European Community's Seventh Framework Programme (FP7/2007-2013) grant no. 317647 (Leone). We thank RIPE NCC for collaborating to the development of the graph animation framework used in this work.

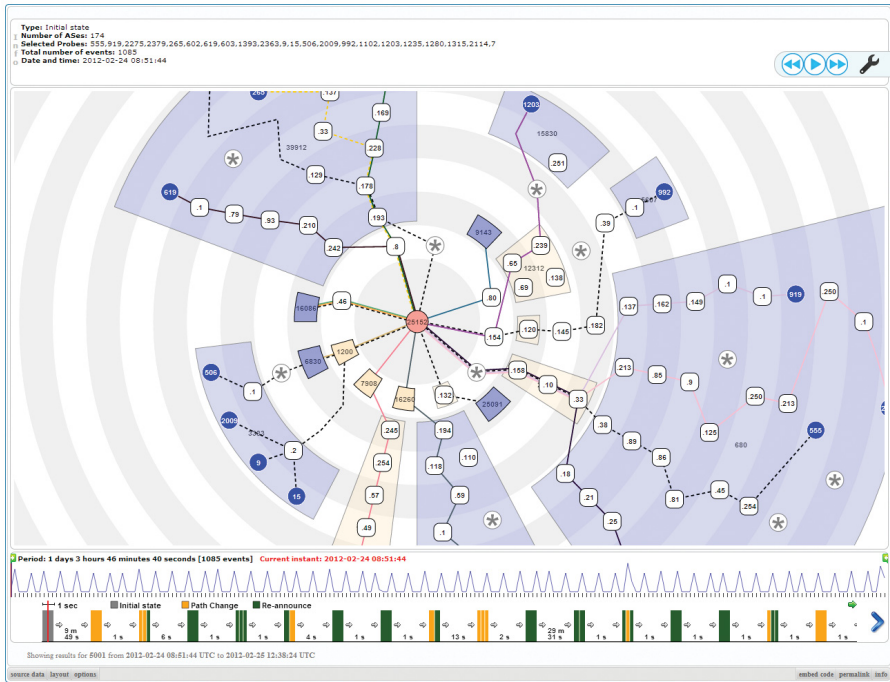


Fig. 1. The main interface of TPLAY

HTTP queries) towards selected targets. They produce a huge amount of data that is difficult to explore, especially when dealing with the network topology. Some examples follow. SamKnows [7] is a broadband measurement service for consumers. MisuraInternet [4] is an Italian project that measures the quality of broadband access. BISmark [28] is a platform for measuring the performance of ISPs. RIPE Atlas [22], CAIDA Ark [2], and M-Lab [3] continuously perform large scale measurements towards several targets.

In this paper we present a system for traceroute visualization called TPLAY, designed for supporting Internet Service Providers (ISPs) and Internet Authorities in the management and maintenance of the network. The requirements were gathered interacting with several ISPs, within the Leone FP7 EC Project, and with the RIPE Network Coordination Center (RIPE NCC). The system works as follows. The user selects a set S of probes of a certain Internet measurement project (all the experiments in this paper have been conducted using RIPE Atlas [22] probes), a target IP address τ , and a time interval \mathcal{T} , and obtains a visualization of how the traceroutes issued by the probes in S reach τ during \mathcal{T} . TPLAY can be used to study several properties of traceroute paths. These include assessing the reachability of τ over time, discovering the ISPs that provide connectivity to reach it, monitoring the length of traceroute paths as a performance indicator, and inferring how routing policies affect the paths of different probes in S .

A snapshot of TPLAY is in Fig. 1. The routing graph is presented with a radial drawing. The geometric distance between τ and any object reflects the topological distance of that object in the network. Also, since traceroutes tend to give too many details, the

system allows to look at the network at different abstraction levels. Finally, the evolution of traceroute paths over time is presented by means of geometric animation.

The paper is organized as follows. In Section 2 we detail the use cases, describe the adopted visualization metaphor, and introduce some formal terminology. In Section 3 we detail the algorithms used to compute the visualization and compare them to the state-of-the-art. In Section 4 we describe the prototype implementation of our tool and the technical challenges we faced. Section 5 contains conclusions and future directions.

2 Use Cases and Visualization Metaphor

The main tasks associated with our system are detailed below. Two of them deal with *Autonomous Systems* (ASes), i.e. entities representing Internet administrative authorities. Once the input is specified as detailed in Section 1, the user is interested in the following. *Security*: knowing what ASes provide connectivity to reach the target over time. That is interesting from the perspective of security, because some ASes may be less trusted than others. *Policy*: seeing how traffic is routed inside a specific AS over time. That helps discover load balancing issues or differences in the routing applied to different probes. *Distance*: knowing the number of hops traversed by each probe over time. Longer paths are indeed potentially responsible for instability and inefficiency. *Dynamics*: seeing how the routing changes at a specific time instant, based on external key indicators. For example, the user may want to check if the routing has changed after a noticeable drop in the round-trip delay experienced when reaching the target.

We discarded solutions based on geographic representations for many reasons. First of all, the fact that a router belongs to a certain ISP or AS is the main piece of information for our purposes, whereas geography is only a secondary feature that further characterizes the nodes in the network. Also, the geo-location data associated with IP addresses is often wrong or missing, and anycast addresses (i.e., those assigned to more than one physical device) can not be mapped to a single location. Finally, the use of landmarks on geographical maps would require special care to avoid geometric cluttering. Motivated by the above, we focused on a topological representation of the data.

The visualization metaphor we adopted is presented below together with supporting motivations. Graphs are represented with *radial layered drawings*, where vertices are placed on concentric circles and targets are in the center. This style of drawing is notably effective for visualizing sparse hierarchical graphs (see, e.g., [31]); in Section 3 we show that our application domain meets such requirement. The probes originating the traceroutes are in the periphery of the drawing. This approach is effective in displaying topological distances. Moreover, radial drawings have their center as the only focus point, which avoids giving probes additional importance due to a privileged geometric position. Finally, the drawing looks like an abstract geography and hence borrows the typical user experience deriving from cartography and geographical visualization.

The need of visualizing the network at different abstraction levels is met by partitioning the set of routers into *clusters*. In our setting, clusters are in correspondence with ASes. The user can modify the representation by interacting with any cluster to either contract or expand it. A *contraction* causes all the routers in the cluster to be merged into a single object representing the cluster, while an *expansion* does the opposite. Collapsing all clusters leads to a high-level, uncluttered view of the graph. On the other

hand, the user can expand all the clusters to see all the traversed routers. In general, the user can arbitrarily expand any subset of clusters to examine them in detail.

Paths for reaching the target from the probes change over time. A natural way to show the evolution of traceroutes at different time instants is to present an animation of the drawing. More precisely, for each instant in a given time interval we show a different drawing, corresponding to the traceroutes that are available at that instant. We animate the change from a drawing to a successive one by means of a geometric morph.

Since the visualization is highly interactive and the graph changes over time, preserving the mental map is of paramount importance. Indeed, the user can both animate the drawing in a specific time interval and expand/contract individual clusters. We require that the same drawing is visualized for any two sequences of cluster expansions/contractions that produce the same graph. Also, the graph should be animated smoothly, even at the expense of traversing drawings that are not aesthetically optimal.

Traceroute paths cannot simply be merged and displayed in an aggregate fashion, since each of them has its own informative value and can change over time. For this reason, we represent paths adopting a metro-line metaphor [24] and draw them using different colors. Further, paths that never change in the selected time interval should be easily distinguished. In this context we adopt the method described in [14]. Paths that do not change are partitioned into sets such that each of them determines a tree on the graph. Each tree is depicted with dashed lines and a distinctive color. This has the effect of reducing the number of lines in the drawing, while preserving the routing information for each probe. Paths that change are instead represented by solid lines.

The objects to be visualized are formally defined as follows. Consider a time interval \mathcal{T} and a set of probes \mathcal{S} . During \mathcal{T} each probe periodically issues a traceroute towards a target IP address τ . A *traceroute* from probe $\sigma \in \mathcal{S}$ produces a simple directed path on the Internet from σ to τ . If such a path is available in Internet at time $t \in \mathcal{T}$, then it is *valid* at time t . Each vertex of a traceroute originated from $\sigma \in \mathcal{S}$ is either a router or a computer. Vertices are identified as follows: (1) σ has a unique identifier selected by the RIPE NCC; (2) vertices with a *public* IP address are identified by it; (3) vertices with a *private* IP address are identified by a pair composed of their address and the identifier of σ ; (4) the remaining vertices are labeled with a “*” (i.e. an unknown IP address). For the sake of simplicity, consecutive vertices labeled with “*” are merged into one. A vertex labeled with “*” is identified by the identifiers of its neighbors in the traceroute.

A digraph G_t is defined at each instant $t \in \mathcal{T}$ as the union of all the paths valid at t produced by the traceroutes issued by the probes of \mathcal{S} . A digraph $G_{\mathcal{T}}$ is defined as the union of all graphs G_t . Each vertex of $G_{\mathcal{T}}$ is assigned to a *cluster* as follows. (1) Each probe is assigned to the cluster that corresponds to the AS where it is hosted. (2) Each vertex identified by a public IP address [6] is assigned to a cluster that corresponds to the AS announcing that address on the Internet. This information is extracted from the RIPEstat [23] database and may occasionally be missing. (3) Each vertex v that is not assigned to a cluster after the previous steps is managed as follows. Consider all traceroute paths containing v . For traceroute p let $\mu(v)$ be the cluster assigned to the nearest predecessor (successor) of v with an assigned cluster. If $\mu = v$ then μ is added to the set

of candidate clusters for v . If such set has exactly one cluster, v is assigned to it. If there is more than one candidate, an inconsistency is detected and the procedure terminates prematurely. (4) Each remaining vertex is assigned to a corresponding *fictitious* cluster. We define V_μ as the set of vertices assigned to cluster μ .

For any $t \in \mathcal{T}$ G_t can be visualized at different abstraction levels. Namely, the user can select a set \mathcal{E} of clusters that are fully visualized and each cluster that is in the complement $\bar{\mathcal{E}}$ of \mathcal{E} is contracted into one vertex. More formally, given the pair G_t, \mathcal{E} the visualized graph $G_{t, \mathcal{E}}(V, E)$ is defined as follows. V is the union of the V_μ for all clusters $\mu \in \mathcal{E}$, plus one vertex for each cluster in $\bar{\mathcal{E}}$. E contains the following edges. Consider edge (u, v) of G_t and clusters μ and ν , with $u \in \mu$ and $v \in \nu$. If $\mu \neq \nu$, $\mu \in \mathcal{E}$, and $\nu \in \mathcal{E}$, then add edge (u, v) . If both μ and ν are in $\bar{\mathcal{E}}$ then add edge (μ, ν) . If $\mu \in \mathcal{E}$ ($\mu \in \bar{\mathcal{E}}$) and $\nu \in \bar{\mathcal{E}}$ ($\nu \in \mathcal{E}$) then add edge (u, ν) ((μ, v)). We define $G_{\mu, t}$ as the subgraph of G_t induced by V_μ . Analogously, we define $G_{\mu, \mathcal{T}}$ as the subgraph of $G_{\mathcal{T}}$ induced by V_μ . We define $G_{\mathcal{T}, \mathcal{E}}$ as the union of the $G_{t, \mathcal{E}}$ for each $t \in \mathcal{T}$.

Fig. 1 shows an overview of our prototype implementation. Let $t \in \mathcal{T}$ be the time instant selected by the user. Graph $G_{t, \mathcal{E}}$ is represented by a radial drawing centered in τ . All vertices and clusters that appear in at least one traceroute in \mathcal{T} are in the drawing, including those that are not traversed by any traceroute at time t . Probes in \mathcal{S} are represented as blue circles and labeled with their identifier. Vertices are represented as white rounded rectangles and labeled with the last byte of their IP address, or with a “*”. Clusters are represented as annular sectors and labeled with their AS number. Note that vertices assigned to expanded clusters are enclosed in their sectors, while sectors of contracted clusters are empty. The light red cluster contains τ . Clusters containing probes in \mathcal{S} are light blue. The remaining clusters are light yellow. Fictitious clusters are not displayed. Each path from a probe $\sigma \in \mathcal{S}$ to τ is represented by a colored curve from σ to τ passing through all intermediate vertices. Paths are either solid or dashed, depending on whether they change or not during the time interval \mathcal{T} . Concentric circles in the background represent the increasing topological distance of vertices.

Fig. 2 contains various details on how the interaction with the visualization works. A graph with static paths and no expanded clusters is presented in Fig. 2(a). It is related to a target τ , a set of probes \mathcal{S} , and a small time interval \mathcal{T}' . Note that some vertices are not enclosed in any cluster: they belong to fictitious clusters. A graph for τ , \mathcal{S} and \mathcal{T}'' ($|\mathcal{T}''| > |\mathcal{T}'|$) is presented in Fig. 2(b). Some dynamic paths are visible. The same graph is presented in Fig. 2(c) with one expanded cluster. Note how the ordering of clusters and vertices on the radial layers is preserved. Fig. 2(d) shows the same expanded graph at a different time instant. The intermediate vertices of two paths are different.

Fig. 2 also helps us explain how the tasks detailed at the beginning of the section can be accomplished. The *Security* task is satisfied in Fig. 2(a): we can see how ASes 1200 and 20965 provide connectivity to reach the target. The *Policy* and *Distance* tasks are addressed in Fig. 2(c), where the length and structure of the paths from each of the three probes 619, 602, 265 is clearly visible. The *Dynamics* task is solved in Fig. 2(c)-(d), where we can see how the paths change for probes 619 and 602 after a routing event.

The user interaction plays a major role in our metaphor. The reader can visit [8] for an example video of the interaction with TPLAY.

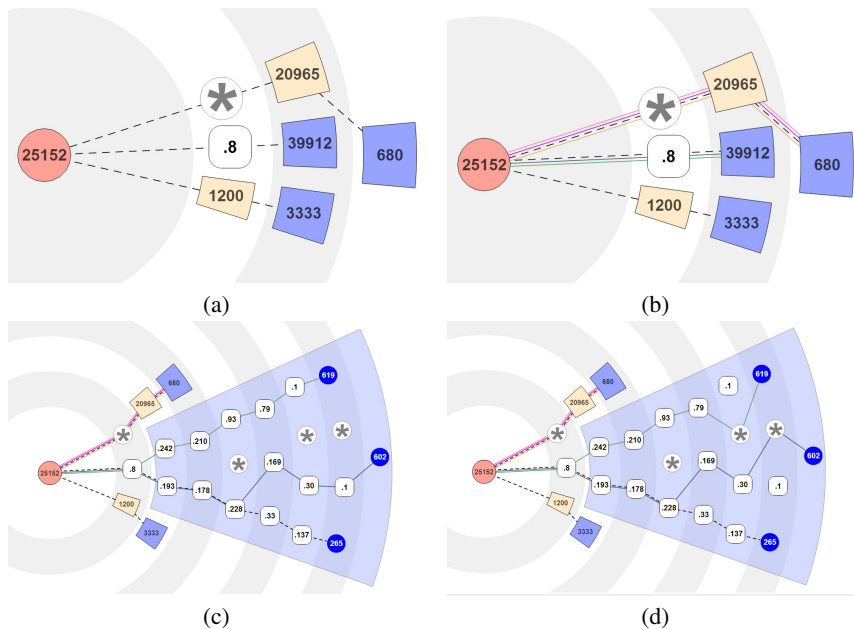


Fig. 2. Details of the interactive features of our visualization. (a) A graph $G_{\mathcal{T}'}$ relative to a target τ , a set of probes \mathcal{S} , and a time interval \mathcal{T}' . All paths in $G_{\mathcal{T}'}$ are static and all clusters contracted. (b) A graph $G_{\mathcal{T}''}$ relative to τ , \mathcal{S} , and \mathcal{T}'' ($|\mathcal{T}''| > |\mathcal{T}'|$). Some paths are dynamic and all clusters are contracted. (c) $G_{\mathcal{T}''}$ with an expanded cluster. (d) $G_{\mathcal{T}''}$ at a different time instant.

3 The Algorithms

We started our analysis by computing several statistics on the RIPE Atlas data set that we used to test the system. It consists of traceroutes executed in one month (July 2012) by 200 probes. Fig. 3 presents the main results of our analysis. In Fig. 3(a) we plot a cumulative distribution function of the length of traceroute paths. That gives us a rough indication on the maximum distance between a probe in \mathcal{S} and τ . The plot shows that traceroutes with more than 15 vertices are rare, confirming the suitability of the radial metaphor. In Fig. 3(b) we plot the number of vertices and the density ($|E|/|V|$) of $G_{\mathcal{T}}$ as a function of \mathcal{T} . It turns out that $G_{\mathcal{T}}$ is quite sparse for time intervals that are compatible with the application domain. In particular, the density ranges between 1.2 and 1.5 for time intervals within 24 hours. The number of vertices is in the range of 2000.

As a second step, we performed experiments using spring embedders and hierarchical drawing algorithms. Layouts produced by spring embedders [29] are unsuitable for our metaphor, because the topological distance between vertices is not always represented and because they produce drawings with not enough regularity. Also, they tend to introduce crossings that are avoidable, for the expected density of the data set. For hierarchical drawing, we experimented both basic algorithms [29] and variations that allow to represent clustered graphs [25,26]. The experiments put in evidence that crossing-reduction heuristics like those in [25,26] are quite effective. However, in our

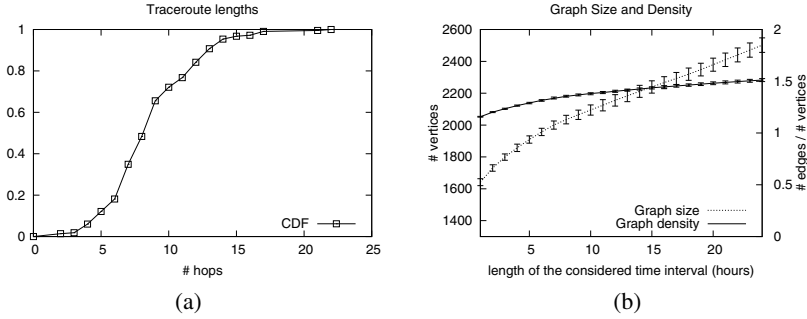


Fig. 3. Statistics on the data set. (a) Cumulative distribution function (CDF) of the length of traceroute paths at July, 1st 2012 at 00:00. CDFs at different instants exhibit similar features. (b) Plot showing the number of vertices and the density of $G_{\mathcal{T}}$ as a function of \mathcal{T} . For each day in the month we set an initial time at 00:00 and grow \mathcal{T} from 1 to 24 hours. For each value of \mathcal{T} we plot the average density and number of vertices. We report the standard deviation with error bars.

case most graphs are planar or quasi-planar and hence planarity-based methods are more attractive. Finally, we discarded upward planar drawings [29]. The main reason is that they tend to use vertical space to resolve crossings, which may result in large geometric distances between vertices that are topologically close.

A very high level and informal description of our algorithmic framework is the following. We pre-compute a hierarchical drawing I_0 of $G_{\mathcal{T}}$ that integrates all the traceroutes in \mathcal{T} . In that drawing all clusters are expanded. The layout is computed in such a way to have few crossings involving connections between clusters. The quality of the layout inside the clusters is considered with lower priority. Moreover, the quality of the drawing of edges that are part of many traceroutes in \mathcal{T} is privileged among the edges of $G_{\mathcal{T}}$. The drawing computed for each cluster is stored and reused in any drawing where that cluster is expanded. The hierarchical drawing is mapped to a radial drawing with a suitable coordinate transformation. Changes in the drawing due to an expansion or contraction of a cluster or a change in traceroutes are visualized with an animation. At any instant $t \in \mathcal{T}$ only the traceroutes that are valid in t are displayed.

For our purposes an interesting reference is [11] that constructs radial drawings adapting techniques of the Sugiyama Framework, but, unfortunately, it does not deal with clusters. The algorithm in [16], which extends the one described in [15], inspired part of our work. However, it proposes a clustered planarity testing algorithm, while we rather need an algorithm for clustered graph planarization, and [16] is not easily extensible for this purpose (neither is the algorithm in [12] that is not suitable for hierarchical drawings). For these reasons we devised a new algorithm to produce clustered hierarchical drawings, as a planarization-oriented variation of [16]. In [21] an algorithm is proposed for the expansion/contraction of clusters of hierarchical drawings, building on [27]. Unfortunately it uses local layering for vertices, while global layering [25,26] is more suitable for our needs because it produces more compact drawings. Indeed a very common use case of TPLAY is to expand all clusters along one or more traceroutes. Local layering would visualize far from τ also vertices in unrelated paths because of the increased need for vertical space of their layers. For this reason we devised a new algorithm for

expanding/contracting clusters that is based on global layering. Differently from [21] it is not a local update scheme, i.e. it computes a new drawing for the whole graph at each interaction. The lower time efficiency is negligible because the graphs commonly handled by TPLAY are small. Finally, mental map preservation during expansion/contraction of clusters is addressed by a geometric morph, implemented as an animation of objects from their initial position to their final position (see, e.g., [14]).

What follows gives more details on our the algorithmic framework. In a preprocessing step several information are computed on $G_{\mathcal{T}}$ that will be used for actual drawings. Given any $G_{\mu, \mathcal{T}}$, a vertex is a *source* (*sink*) of $G_{\mu, \mathcal{T}}$ if it is the last (first) vertex of $G_{\mu, \mathcal{T}}$ encountered in some traceroute path. Each graph $G_{\mu, \mathcal{T}}$ is augmented with extra vertices and edges so that all the longest paths from a source to a sink have the same length. The added vertices are called *fictitious vertices* of μ and ensure that, given an edge $(u, v) \in G_{\mathcal{T}}$, $u \in \mu$, $v \in \nu$, $\mu \neq \nu$, clusters μ and ν do not share a layer in any drawing of $G_{t, \mathcal{E}}$. Moreover, they force edges that leave a cluster by spanning several layers to be routed inside that cluster. A μ -drawing is pre-computed for each $G_{\mu, \mathcal{T}}$. It consists of 1. assigning vertices to layers so that all edges are between consecutive layers and 2. computing a total order for the vertices of each layer. A partial order \prec is computed for clusters, such that for any two clusters μ and ν with $\mu \prec \nu$, the vertices of μ appear to the left of the vertices of ν for any drawing Γ where μ and ν share one or more layers. This helps preserve the mental map during expansions/contractions. The preprocessing step requires to compute a drawing Γ_0 of $G_{\mathcal{T}}$ with all clusters expanded. Γ_0 gives the information needed to compute a μ -drawing for each cluster and a partial order \prec for clusters. The algorithm to compute Γ_0 is similar to that in [16], where a PQ-tree [13] is used to order vertices along the layers of the drawing. Our PQ-tree is initialized with a spanning tree of $G_{\mathcal{T}}$ and incrementally updated with the remaining edges that induce ordering constraints. An edge is added only if it does not produce a crossing (i.e. the PQ-tree does not return the null tree). A rejected edge will produce crossings in Γ_0 . Edges are added with priority given by their aesthetic importance: namely, they are weighted by the number of traceroutes that traverse them in \mathcal{T} . As an implementation detail, we actually compute a total order for clusters to represent a partial order \prec . Such an order is produced by a DFS visit of the spanning tree of $G_{\mathcal{T}}$. The tree has an embedding induced by the layer orders produced by the PQ-tree algorithm, and children of a vertex are visited in clockwise order. Intuitively, we preserve the geometric left-to-right order for clusters from Γ_0 , and reuse it to produce a drawing of any $G_{t, \mathcal{E}}$.

The computation of the drawing $\Gamma_{\mathcal{T}, \mathcal{E}}$ of $G_{\mathcal{T}, \mathcal{E}}$ is detailed below. Before that, note that once $\Gamma_{\mathcal{T}, \mathcal{E}}$ is computed, we display, for any $t \in \mathcal{T}$ all the vertices of $G_{\mathcal{T}, \mathcal{E}}$ but only the edges of $G_{t, \mathcal{E}}$. This is done to preserve the mental map of the user, using $\Gamma_{\mathcal{T}, \mathcal{E}}$ as a “framework” that “hosts” the drawings of each instant. First, a layering of $G_{\mathcal{T}, \mathcal{E}}$ is computed such that for each vertex the distance from τ is minimized. Also, dummy vertices, called *fictitious vertices* of $G_{\mathcal{T}, \mathcal{E}}$, are added so that each edge spans two consecutive layers. Vertices are horizontally ordered on each layer such that: 1. \prec is enforced; 2. for each cluster μ of \mathcal{E} , the orders on the layers of its μ -drawing are enforced; 3. the fictitious vertices of $G_{\mathcal{T}, \mathcal{E}}$ are placed in such a way to have few crossings. In particular, they must not be interleaved with the vertices of any cluster, that is, the vertices of each cluster must be consecutive on every layer. For this reason, each

fictitious vertex is assigned to a new fictitious cluster, which is inserted in the partial order \prec in an intermediate position between the endpoints of the edge it belongs to. Finally, the ordered layers are used to assign geometric coordinates to vertices. The width of each cluster μ is computed as follows. Consider the layer containing the largest number of vertices assigned to μ . The cluster is assigned a width proportional to this number. Vertices of μ are assigned horizontal coordinates such that they can be enclosed by a rectangle with height proportional to the number of layers assigned to the vertices of μ and width equal to the width of μ . We avoid intersection between enclosing rectangles by means of an auxiliary directed acyclic graph where vertices are clusters of $G_{\mathcal{T},\varepsilon}$ and edges are selected from \prec depending on which pairs of clusters share a layer in the current layering of $G_{\mathcal{T},\varepsilon}$. Edges are weighted based on the widths of the clusters they are incident to. The total width of the drawing is given by the longest path in this graph. The above is applied recursively to compute the horizontal spacing among all clusters. The vertical coordinate of a vertex is equal to the one assigned to its layer, which is proportional to the index of that layer in the total order of layers.

Going back to the state-of-the-art, concerning restrictions $R1$, $R2$ and $R3$, described in [16], that a planar clustered hierarchical drawing must obey, drawings produced by our algorithm satisfy $R1$ and $R2$, while we consider $R3$ too restrictive for our application. $R1$ is satisfied in the preprocessing step by merging, for each cluster, all sources into one vertex. The PQ-tree is initialized with a spanning tree that contains all these new vertices, which has the effect to keep the vertices of each cluster consecutive on any layer. $R2$, as shown in [16], is automatically satisfied for the initial drawing Γ_0 , and is satisfied for any drawing of $G_{t,\varepsilon}$ by exploiting the partial order \prec .

To obtain a radial drawing, the geometric coordinates of vertices so computed are transformed as follows. Each vertex is placed on the perimeter of a circle centered in an arbitrary fixed point and having radius equal to the vertical coordinate of the vertex. Then the horizontal coordinate of the vertex is mapped to a circular coordinate on the perimeter of that circle. The perimeters of clusters are mapped with a similar radial transformation. An edge (u, v) is drawn either as a straight segment or a curved arc, depending on the angle it must sweep to connect vertices u and v . Note that in our setting each edge connects only vertices in two consecutive layers, hence a curved edge can be drawn only in the space between these layers.

4 Implementation and Technical Challenges

The implementation of TPLAY is split into three main blocks: 1. a visualization front-end; 2. a layout engine; and 3. a data back-end.

The *visualization front-end* is a Web application. It allows the user to specify input parameters and to visualize and animate interactive graphs. The main interface is presented in Fig. 1 and additional images are provided at [8]. It is composed of four main elements: the controller, the graph panel, the info panel, and the timeline panel. We detail their functionalities below.

The *controller* is a sliding panel located in the upper right corner. It allows the user to input queries composed of a target τ , a time interval \mathcal{T} , and a set of probes \mathcal{S} . Once the visualization is ready, the controller can be used to animate the graph with the traceroute

paths available during \mathcal{T} . The play/pause/stop, repeat-last, step-back, and step-forward buttons allow for a fine-grained management of the graph animation.

The *graph panel* displays the interactive graph, initially centered and fitted to the window. The user can pan and zoom it with the mouse. The animation of the graph consists of a sequence of morphing steps. Each step transform the graph by applying the effects of an event involving one or more traceroute paths. Given a probe $\sigma \in \mathcal{S}$, an event can consist of: (a) the availability of a new traceroute path from σ to τ ; (b) a change in the sequence of vertices in the traceroute path from σ to τ ; (c) a disconnection resulting in an empty traceroute path. Events of type (a) are rendered with a gradual introduction of new paths in the graph. Events of type (b) are rendered with a geometric morph of curves representing the involved paths. Events of type (c) are rendered with a blinking effect after which paths disappear. We introduce a delay between each pair of consecutive animation steps. The delay is proportional to the logarithm of the elapsed time between the corresponding routing events. This gives an approximate perception of elapsed time, while limiting the overhead on the total animation time. The elements of the graph are interactive and show additional information on request. Hovering a vertex with the mouse for a few seconds highlights all the paths passing through it. Hovering a path for a few seconds highlights the path and all its vertices.

The *info panel* is in the upper part of the window. It shows all the available information about any selected network component represented in the graph. It also displays a textual description for the latest event that caused an update of the visualized graph.

The *timeline panel* is in the lower part of the window and contains two timelines that allow to accurately navigate the traceroute information in \mathcal{T} . The first, called *control timeline*, provides a fast overview of the trend in the number of events over time. The second, called *selection timeline*, shows individual events ordered in time and is designed for fine-grained analysis. Each block in the selection timeline contains a sequence of events happening at the same time, represented with colored rectangles. Different colors are used for different types of events. The elapsed time between any two consecutive blocks is reported in the area between them. Both timelines feature a red cursor that points at the current time instant and is continuously updated during the animation. The user can drag the cursors, changing the current instant and updating the graph accordingly. The selection timeline can only show a limited number of events due to its constrained area. In case there are more events, the animation causes involved events to be smoothly translated into the visible part of selection timeline. The user can scroll horizontally to reveal hidden events. Further, the user can limit the animation to a particularly interesting period within \mathcal{T} by dragging the two green sliders at the top of the control timeline. The sliders on the selection timeline are updated accordingly.

The implementation of the front-end required to focus on some algorithmic details. The arrangement of paths in a metro-line fashion is implemented as follows. First of all, an arbitrary total ordering is computed on the set of visualized paths. For each edge without bends in the graph, the paths that traverse it are drawn as parallel segments connecting the two endpoints of the edge. The ordering of such segments reflects the total ordering of paths to promote consistency between edges. In case the edge contains bends, the drawing is computed in two steps. First, we split the bended edge in a sequence of intermediate edges e^1, \dots, e^n and compute the path segments for each of

them. Second, for each pair of consecutive intermediate edges (e^i, e^{i+1}) and for each path that traverses it, we call (u, v) and (w, z) the two segments computed respectively for e^i and e^{i+1} . If there is an intersection point p between (u, v) and (w, z) , we rewrite the two segments as (u, p) and (p, z) . Otherwise, we add a connection (v, w) between (u, v) and (w, z) . Path colors are computed with the algorithm described in [19] to ensure that they are distinguishable from each other.

The front-end is written in JavaScript and HTML. It is based on the BGPlay.js framework [1] that we developed in collaboration with the RIPE NCC. The objective of the framework is to simplify the implementation of web-based tools for the representation of evolving data described in terms of graph components. The framework consists of a solid implementation of graph domain objects and a set of modules. Modules provide functionalities and representation of data and can be used to compose ad-hoc tools. We use Scalable Vector Graphics for the representation and animation of the graph.

The visualization always starts with an overview of the traceroutes. Hence, the *layout engine* is invoked to produce a drawing of $G_{\mathcal{T}, \emptyset}$. When the user expands/contracts a cluster (a cluster is added/removed from \mathcal{E}) the layout engine is invoked again on $G_{\mathcal{T}, \mathcal{E}}$. In the implementation of the radial drawing we artificially increase the radius of each layer by an additional offset, such that vertices on dense layers are not overlapped. For the sake of simplicity, curved segments are uniformly sampled and drawn as polylines.

The layout engine is implemented in Java. We initially designed it to be implemented as part of the visualization front-end, but later moved to a back-end implementation in order to make use of already existing libraries. In particular we adopted a PQ-tree implementation [17] and Apache Commons Graph [9] for general graph models and algorithms. We optimized the output of the layout engine after the initial layout, so that only graph elements with new drawing coordinates are included.

The *data back-end* is mainly responsible of retrieving and preprocessing traceroute data. The result is then used by the front-end to animate traceroute events and by the layout engine to compute the drawings.

5 Conclusions and Open Problems

We presented a metaphor for the visualization of traceroute measurements towards specific targets on the Internet. It consists of a radial drawing of a clustered graph where vertices are routers or computers and clusters are administrative authorities that control them. Our metaphor allows the user to interact with the visualization, both exploring the content of individual clusters and animating the graph to see how traceroute paths change over a time interval of interest.

In the future we will take into account the *DNS resolution* of selected targets in the visualization. That means that some targets may be represented by more than one vertex, giving rise to an anycast behavior of the target, depending on the policies implemented at the DNS level. We will also explore the possibility to process streams of incoming data, adding or removing elements in the visualization incrementally.

References

1. BGPlayJS, <http://www.dia.uniroma3.it/~compunet/www/view/tool.php?id=bgplayjs>
2. CAIDA Ark, <http://www.caida.org/projects/ark/>

3. Measurement Lab, <http://www.measurementlab.net/>
4. MisuraInternet, <https://www.misurainternet.it/>
5. Monitor Scout Traceroute, <http://tools.monitorscout.com/traceroute/>
6. RFC 1918, address allocation for private internets,
<http://www.ietf.org/rfc/rfc1918.txt>
7. SamKnows, <http://www.samknows.com/broadband/>
8. TPlay, <http://www.dia.uniroma3.it/~compunet/projects/tplay>
9. Apache Software Foundation. Apache Commons Graph,
<http://commons.apache.org>
10. Augustsson, B.: Xtraceroute,
<http://www.dtek.chalmers.se/~d3august/xt/index.html>
11. Bachmaier, C.: A radial adaptation of the sugiyama framework for visualizing hierarchical information. *IEEE Trans. on Visualization and Computer Graphics* 13(3), 583–594 (2007)
12. Di Battista, G., Didimo, W., Marcandalli, A.: Planarization of clustered graphs. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) *GD 2001*. LNCS, vol. 2265, pp. 60–74. Springer, Heidelberg (2002)
13. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *JCSS* 13(3), 335–379 (1976)
14. Colitti, L., Di Battista, G., Mariani, F., Patrignani, M., Pizzonia, M.: BGPlay: A System for Visualizing the Interdomain Routing Evolution. In: Liotta, G. (ed.) *GD 2003*. LNCS, vol. 2912, pp. 295–306. Springer, Heidelberg (2004)
15. Di Battista, G., Nardelli, E.: Hierarchies and planarity theory. *IEEE Transactions on Systems, Man and Cybernetics* 18(6), 1035–1046 (1988)
16. Forster, M., Bachmaier, C.: Clustered level planarity. In: Van Emde Boas, P., Pokorný, J., Bielíková, M., Štuller, J. (eds.) *SOFSEM 2004*. LNCS, vol. 2932, pp. 218–228. Springer, Heidelberg (2004)
17. Harris, J.: A graphical Java implementation of PQ-Trees, <http://www.jharris.ca>
18. Hokstad, V.: Traceviz: Visualizing traceroute output with graphviz,
<http://www.hokstad.com>
19. Kistner, G.: Generating visually distinct colors,
<http://phrogz.net/css/distinct-colors.html>
20. Periakaruppan, R., Nemeth, E.: Gtrace - a graphical traceroute tool. In: *Proc. 13th USENIX Conference on System Administration*, pp. 69–78. USENIX Association (1999)
21. Raitner, M.: Visual navigation of compound graphs. In: Pach, J. (ed.) *GD 2004*. LNCS, vol. 3383, pp. 403–413. Springer, Heidelberg (2005)
22. RIPE NCC. RIPE Atlas, <http://atlas.ripe.net/>
23. RIPE NCC. RIPEstat, <https://stat.ripe.net/>
24. Roberts, M.J.: *Underground Maps Unravalled - Explorations in Information Design* (2012)
25. Sander, G.: Layout of compound directed graphs. Technical report, FB Informatik, Universität Des Saarlandes (1996)
26. Sander, G.: Graph layout for applications in compiler construction. *Theoretical Computer Science* 217(2), 175–214 (1999)
27. Sugiyama, K., Misue, K.: Visualization of structural information: automatic drawing of compound digraphs. *IEEE Trans. on Systems, Man and Cybernetics* 21(4), 876–892 (1991)
28. Sundaresan, S., de Donato, W., Feamster, N., Teixeira, R., Crawford, S., Pescapè, A.: Broadband internet performance: A view from the gateway. In: *Proc. SIGCOMM* (2011)
29. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall (1998)
30. Visualware. VisualRoute, <http://www.visualroute.com/>
31. Yee, K.-P., Fisher, D., Dhamija, R., Hearst, M.: Animated exploration of dynamic graphs with radial layout. In: *Proc. INFOVIS 2001*. IEEE Computer Society (2001)

Graph Drawing Contest Report

Christian A. Duncan¹, Carsten Gutwenger², Lev Nachmanson³, and Georg Sander⁴

¹ Quinnipiac University, USA

Christian.Duncan@quinnipiac.edu

² Technische Universität Dortmund, Germany

carsten.gutwenger@tu-dortmund.de

³ Microsoft, USA

levnach@microsoft.com

⁴ IBM, Germany

georg.sander@de.ibm.com

Abstract. This report describes the 20th Annual Graph Drawing Contest, held in conjunction with the 2013 Graph Drawing Symposium in Bordeaux (Talence), France. The purpose of the contest is to monitor and challenge the current state of graph-drawing technology.

1 Introduction

This year, the Graph Drawing Contest was divided into an *offline contest* and an *online challenge*. The offline contest had three categories: two dealt with creating and visualizing a graph from a given data set and one was a review network. The data sets for the offline contest were published months in advance, and contestants could solve and submit their results before the conference started. The submitted drawings were evaluated according to aesthetic appearance and how well the data was visually represented. For the visualization itself, typical drawings, interactive tools, animations, or other innovative ideas were allowed.

The online challenge took place during the conference in a format similar to a typical programming contest. Teams were presented with a collection of challenge graphs and had approximately one hour to submit their highest scoring drawings. This year's topic was to minimize the area for orthogonal grid layouts, where we allowed crossings (the number of crossings was not judged, only the area counted).

Overall, we received 12 submissions: 3 submissions for the offline contest and 9 submissions for the online challenge.

2 Creative

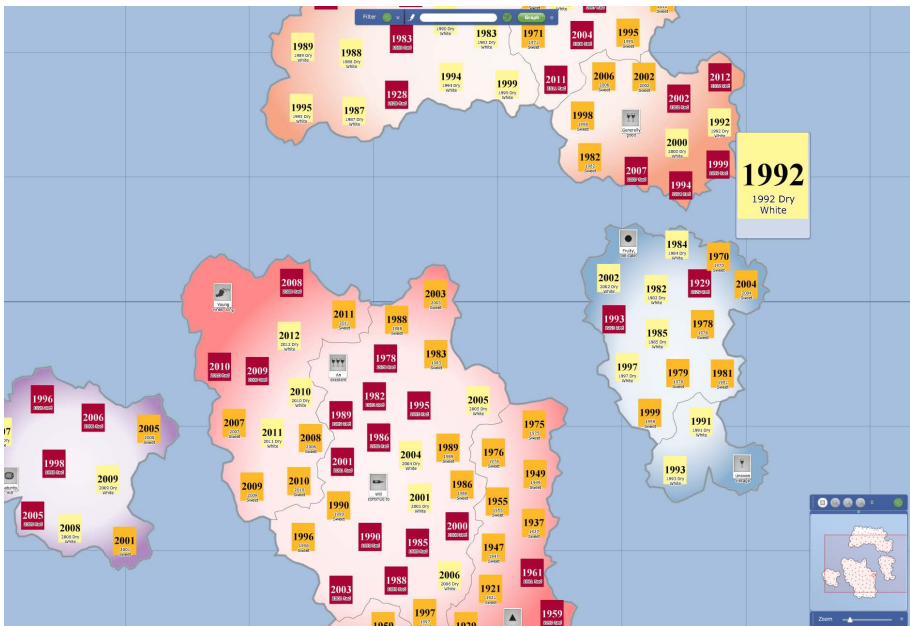
For the two categories in this topic, the task was to create a meaningful graph from data found on a specific website and visualize it in a suitable way. Any kind of visualization was allowed. We proposed pictures, map-like drawings, animations, and interactive tools, but any other innovative idea was also welcome. Submissions were to include the graph itself as well as the visualization.

Find yourself a good wine year



data source: <http://www.bordeaux.com/us/vineyard/bordeaux-wine-vintages>

(a) de Jong, Pazienza



(b) Zelina et al.

Fig. 1. Creative, Category A (Bordeaux Wines)

2.1 Category A: Bordeaux Wines

The first data set could be found on the Bordeaux Wines website¹.

We received two submissions in this category, both included links to an interactive web site for visualizing the data. Fig. 1(a) shows the submission by Jos de Jong and Giovanni Pazienza; their interactive web page provides filtering and shows a dynamic layout, which is smoothly adjusted when nodes appear or disappear due to changes in the filter. The submission by Remus Zelina et al. (see Fig. 1(b)) is also an interactive web page providing filtering, but here the graph is visualized in a map-like fashion obtained by a preceding clustering of the data.

The winner in this category was the team of Jos de Jong and Giovanni Pazienza for their clear and easy-to-use visualization of Bordeaux wines.

2.2 Category B: Bordeaux City

The second data set could be found on the official website of the city of Bordeaux². Unfortunately, we did not receive any submissions in this category.

3 Review Network

In this category the task was to visualize a review network that had been obtained from Amazon reviews on fine foods. The data set for the review network could be obtained from the SNAP website³ of Stanford University. The network included 568,454 reviews, collected over a period of more than 10 years, including 74,258 products and 256,059 reviewers. Each review contained the product ID of the food, allowing access to the product at Amazon, the user ID of the reviewer, and further interesting information like the score and the date of the review. Although any kind of visualization was again allowed, a good submission should nevertheless highlight the quality of the products and the importance of the reviewers.

We only received a single submission in this category, which was again an interactive web page; see Fig. 2. The network was drawn in a map-like fashion clustering the nodes into islands like Chocolate, Tea, or Coffee. When zooming in, more details are revealed and important nodes are highlighted. Hence, the winner in this category was the team Remus Zelina, Sebastian Bota, Siebren Houtman, and Radu Balaban from Meurs, Romania.

4 Online Challenge

The online challenge, which took place during the conference, dealt with minimizing the area in an orthogonal grid drawing. The challenge graphs were not necessarily planar and had at most four incident edges per node. Edge crossings were allowed and

¹ <http://www.bordeaux.com/us/vineyard/bordeaux-wine-vintages>

² <http://www.bordeaux.fr/>

³ <http://snap.stanford.edu/data/web-FineFoods.html>

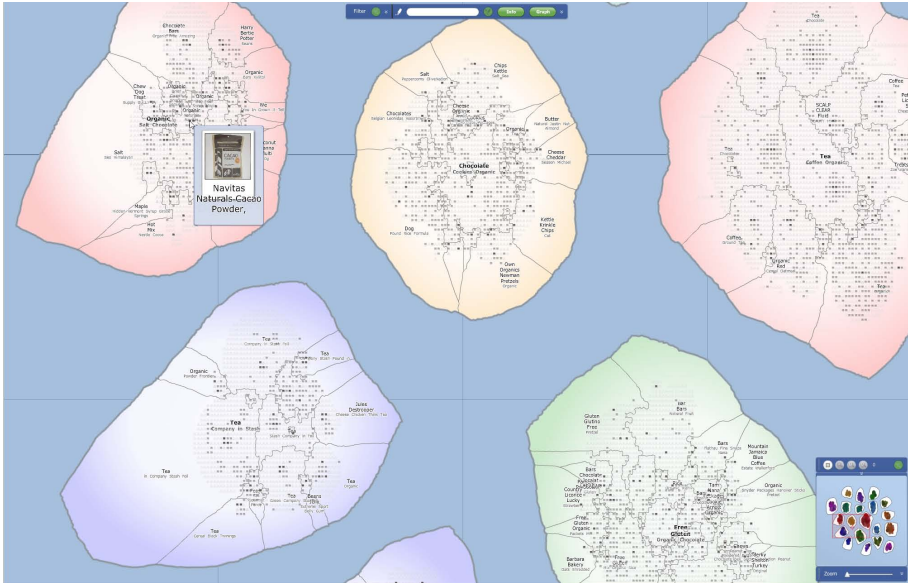


Fig. 2. Review Network

their number did not affect the score of a layout. Since typical drawing systems first try to minimize the number of crossings, which might result in long edges increasing the required area, we were in particular interested in the effect of allowing crossings on the quality of layouts when trying to reduce the area.

The task was to place nodes, edge bends, and crossings on integer coordinates so that the edge routing is orthogonal and the layout contains no overlaps. At the start of the one-hour on-site competition, the contestants were given six graphs with an initial legal layout with a large area. The goal was to rearrange the layout to reduce the area, defined as the number of grid points in the smallest rectangle enclosing the layout. Only the area was judged; other aesthetic criteria, such as the number of crossings or edge bends, were ignored.

The contestants could choose to participate in one of two categories: *automatic* and *manual*. To determine the winner in each category, the scores of each graph, determined by dividing the area of the best submission in this category by the area of the current submission and then taking the square root, were summed up. If no legal drawing of a graph was submitted (or a drawing worse than the initial solution), the score of the initial solution was used.

In the automatic category, contestants received six graphs ranging in size from 59 nodes / 87 edges to 3393 nodes / 4080 edges and were allowed to use their own sophisticated software tools with specialized algorithms. Manually fine-tuning the automatically obtained solutions was allowed. However, no team participated in this category, hence we had no winner in the automatic category this year.

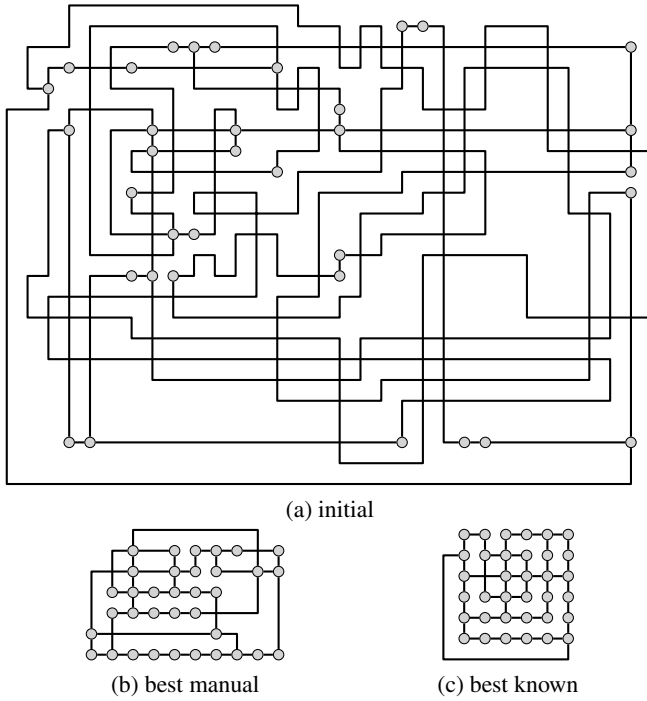


Fig. 3. Challenge graph with 35 nodes and 44 edges: (a) initial layout (area: 768), (b) best manual result obtained by the team of Spisla and Gronemann (area: 70), and (c) best known solution (area 49)

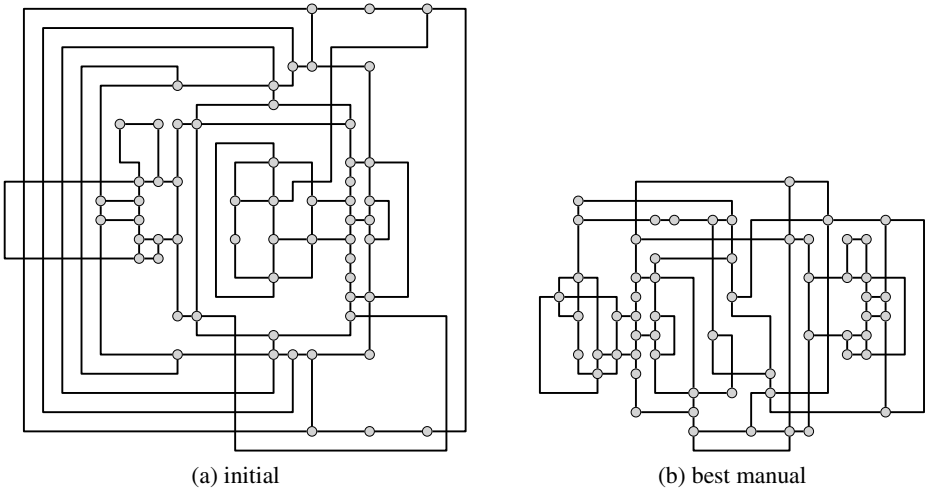


Fig. 4. Challenge graph with 59 nodes and 87 edges: (a) initial layout (area: 600) and (b) best manual result by the team of Bläsius and Rutter (area: 315)

The 9 manual teams solved the problems by hand using IBM's *Simple Graph Editing Tool* provided by the committee. They received six graphs ranging in size from 19 nodes / 26 edges to 150 nodes / 188 edges. Three of the larger input graphs were also in the automatic category; unfortunately we could not compare automatic solutions with manual solutions this time. With a score of 5.485, the winner in the manual category was the team of Thomas Bläsius and Ignaz Rutter from Karlsruhe Institute of Technology, who found the best results for two of the six contest graphs.

Fig. 3 shows a challenge graph with 35 nodes and 44 edges and a very bad initial layout; the best manually obtained result improved the area from 768 to 70, and the best solution we know for this graph has an area of 49. Fig. 4 shows a larger challenge graph with 59 nodes and 87 edges and a better initial layout; here the best submitted solution improved the area from 600 to 315.

Acknowledgments. The contest committee would like to thank the generous sponsors of the symposium and all the contestants for their participation. Further details including winning drawings and challenge graphs can be found at the contest website:

<http://www.graphdrawing.de/contest2013/results.html>

3D Graph Printing in GLuskap^{*}

Joel Bennett and Stephen Wismath

University of Lethbridge, Canada

Abstract. The **GLuskap** software package for creating and editing graphs in 3D has been extended to include 3D printing of graphs by exporting the graph to a common file format capable of being printed on most commercially available 3D printers.

The **GLuskap** software package [1] allows for the creation and editing of graphs in three dimensions (3D). A new plugin for **GLuskap** allows graphs to be exported as an STL (stereolithography) file. The STL file can then be sent to a 3D printer to create a physical representation of the graph; for example, the graph K_7 drawn orthogonally with bends is shown in Fig. 1. Printed physical models of 2D layouts can also be constructed.

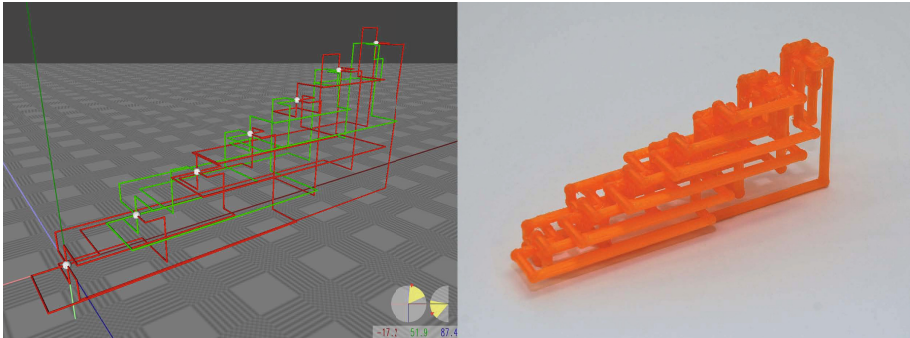


Fig. 1. A graph in **GLuskap** and a physical 3D print of that graph

The Export to STL Plugin

Most graph drawing software, including **GLuskap**, rely on OpenGL, Qt, or DirectX to ultimately display a graph on some device. Although such graphics-based models are very effective for interactive use, they do not produce 3D models appropriate for printing. The only option for 3D graph printing previously, was to completely start from scratch using non-trivial 3D modeling software tools such as Blender, 3DStudioMax, AutoCAD, etc.

^{*} Research supported by N.S.E.R.C.

GLuskap's *Graph to STL* plugin generates a set of geometries that expresses the currently loaded graph. Vertices are represented by tessellated icosahedrons and edges are represented by 12-sided cylinders. The plugin parses the graph, creates the geometry, and saves it out to the specified directory as a 3D model in STL format. Options are available for overriding the vertex and edge sizes of the graph to assist in creating a printable model. The model in STL format may be directly input into printer software or imported into modeling software for post-processing. Since this conversion feature is written in python as a plugin [2] for GLuskap, users with different needs can easily tailor the code as required without recompiling the source code.

Printing Considerations

Considerations must be taken when attempting to print graphs. Depending on the size of the desired 3D print, vertex and edge sizes may need to be scaled to ensure that the graph is sufficiently rigid to be held together during printing and handling.

Printers that use an additive process and build the model up layer by layer without having a supporting substrate may require the use of supports to ensure a rigid print. Supports can later be removed or left in place to reduce the chance that later handling will break the physical print. The ideal orientation when printing a graph depends on the nature of the graph and the type of 3D printer being used. For most graphs, the printer software is used to rotate the model of the graph so that an appropriate side of the graph is aligned with the printing platform.

Graph Drawing

Although there are many theoretical results on 3D graph drawing, the visualization effectiveness of such drawings has certainly been questioned. Previously, the display of 3D drawings was hindered by its reliance on projection onto a 2D display device, at best with active or passive shutter glasses for stereoscopic viewing. The advent of inexpensive, commercial 3D printers removes a major constraint – physical models of graphs can now be produced.

References

1. Dyck, B., Johnson, G., Hanlon, S., Smith, A., Wismath, S.: GLuskap 3.0 manual (2007), <http://people.uleth.ca/~wismath/gluskap/>
2. GLuskap source code and plugins, <http://github.com/ulethHCI/GLuskap/>

Optical Graph Recognition on a Mobile Device

Christopher Auer, Christian Bachmaier, Franz J. Brandenburg,
Andreas Gleißner, and Josef Reislhuber

University of Passau, 94030 Passau, Germany
{auerc,bachmaier,brandenb,gleissner,reislhuber}@fim.uni-passau.de

In [1] we proposed Optical Graph Recognition (OGR) as the reversal of graph drawing. A drawing transforms the topological structure of a graph into a graphical representation. Primarily, it maps vertices to points and displays them by icons, and it maps edges to Jordan curves connecting the endpoints. In reverse, OGR transforms the digital image of a drawn graph into its topological structure. The recognition process is divided into the four phases preprocessing, segmentation, topology recognition, and postprocessing. In the preprocessing phase OGR detects which pixels of an image are part of the graph (graph pixels) and which pixels are not. The segmentation phase recognizes the vertices of the graph and classifies the graph pixels as vertex and edge pixels. The topology recognition phase first recognizes edge sections. Edge crossings divide edges into edge sections, i. e., the regions between crossings and vertices. The edge sections are merged into edges in the most probable way based on direction vectors. The postprocessing phase concludes OGR with tasks like converting the recognized graph into different file formats, like GraphML or adding coordinates to the vertices and edges, such that the recognized graph resembles the input graph.

Our OGR Java implementation OGR^{up} is able to recognize drawings of undirected graphs with the following properties: Vertices are drawn as filled objects such as circles, edges are drawn as contiguous curves of a width significantly smaller than the diameter of the vertices, and they should exactly end at the vertices. Our desktop version of OGR^{up} is of limited use, because it needs a camera as a second device to take a picture of the graph.

In contrast, the new Android version of OGR^{up} needs only a single device. The picture of the graph, e. g., drawn on a whiteboard, is directly taken with the camera of the mobile device at hand. The part of the image that contains the graph can be selected via touch gestures, as seen in Fig. 1. Finally, the graph is recognized and used for further processing. It can be shared, visualized and edited on the mobile device, e. g., as proposed by Da Lozzo et al. [2].

For the Android version, we had to re-implement parts of the graph recognition algorithm and we developed a GUI that fits the capabilities of a mobile device. Whereas the computation time is acceptable in the desktop version of OGR^{up} (≈ 10 seconds for high resolution images), the computation time becomes unacceptably long in the Android version due to hardware limitations. The established digital image processing library OpenCV [3] helped to improve the computation time of OGR^{up} . To circumvent further performance issues, and to make OGR^{up} available for different mobile operating system, like iOS or Windows Phone, we plan a web service implementation of OGR^{up} .

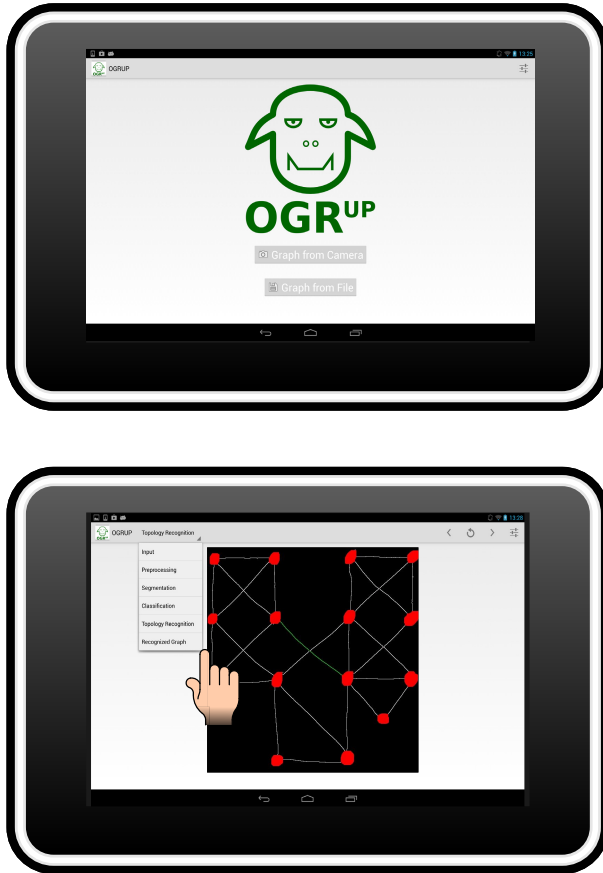


Fig. 1. Two screenshots of OGR^{UP} on an Android tablet

References

1. Auer, C., Bachmaier, C., Brandenburg, F.J., Gleißner, A., Reislhuber, J.: Optical graph recognition. *J. Graph Alg. App.* 17(4), 541–565 (2013)
2. Da Lozzo, G., Di Battista, G., Ingrassia, F.: Drawing graphs on a smartphone. *J. Graph Alg. App.* 16(1), 109–126 (2012)
3. Itseez: OpenCV, <http://www.opencv.org/>

Browser-Based Graph Visualization of Dynamic Data with VisGraph

Jos de Jong and Giovanni E. Paziienza

Almende B.V., Westerstraat 50, 3016 DJ, Rotterdam, The Netherlands
{jos,giovanni}@almende.org

Abstract. VisGraph is an open-source JavaScript library for the real-time graph visualization of dynamic data in a browser. Its characteristics – such as the high portability and the compatibility with multiple data formats – make it suitable for a broad range of research applications.

1 Overview of VisGraph

VisGraph has been created to fill a niche in the graph visualization market for research purposes: the need of a software for graph visualization that is comprehensive enough for a standard non-professional user and does not require any complex installation. Also, VisGraph has two key features: high portability (because it is browser independent) and possibility to work with dynamic data (for instance, coming from a sensor network or from a server). Their combination results in a key feature of VisGraph: a real-time graph representing dynamic data can be shared with a multitude of other users who work on different platforms, including smartphones and tablets. This unique characteristic of VisGraph make it particularly useful in a number of research applications ranging from real-time social network analysis to logistics.

It is crucial to emphasise that VisGraph is a lightweight browser-based library and hence it is *not* competing with desktop/server-based graph visualization softwares, such as Graphviz [1], Gephi [2], or WiGis [3], just to name a few. Other web-based tools (like D3.js [4], Springy [5], Dracula Graph Library [6], or Arbor [7]) have served as inspiration for VisGraph which, as a result, is more customizable and easier to use than its predecessors. Figure 1 shows a qualitative comparison of VisGraph with the aforementioned graph visualization softwares.

VisGraph supports multiple data formats which are all eventually translated into the native JSON format, which consists of two arrays (one for the nodes and the other for the edges) containing information regarding the topology and style of the graph; in the current version, Google DataTable and the DOT language are fully supported. As for the graph layout, VisGraph uses a force-directed algorithm where nodes try to keep a minimum distance from each other. The user can interact with the graphs by dragging the nodes and customise them by changing the colour and size of the elements etc; graphs of up to a few hundreds of nodes can be smoothly rendered by using HTML5 Canvas elements. VisGraph is part of a JavaScript library called vis.js [8], which includes other visualization

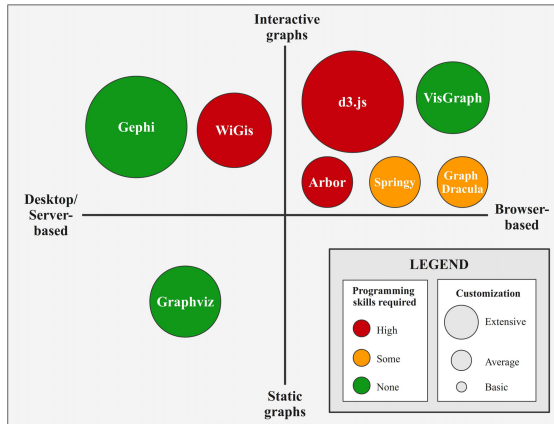


Fig. 1. Qualitative comparison of VisGraph with other graph visualization softwares

tools as well as some components to manipulate dynamic data, which VisGraph uses. The whole vis.js library (including VisGraph) requires no installation and it can be easily included in a web page by adding a single line of code.

Several additional features of VisGraph are under development, such as: i) the option to export the graphs to the formats supported by VisGraph; ii) the addition of clustering functionalities and further layout mechanisms (which will be particularly useful when graphs are visualized on portable devices); iii) the possibility for the user to edit the graph on the fly by creating and removing attributes without modifying the data set.

References

1. Ellson, J., Gansner, E.R., Koutsofios, L., North, S.C., Woodhull, G.: Graphviz - open source graph drawing tools. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 483–484. Springer, Heidelberg (2002)
2. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks. In: International AAAI Conference on Weblogs and Social Media, vol. 2. AAAI Press, Menlo Park (2009)
3. Gretarsson, B., Bostandjiev, S., O'Donovan, J., Höllerer, T.: WiGis: A framework for scalable web-based interactive graph visualizations. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 119–134. Springer, Heidelberg (2010)
4. Data-driven documents (2013), <http://d3js.org>
5. Springy.js: a force directed graph layout algorithm in java script (2013), <http://getspringy.com>
6. Dracula graph library (2013), <http://www.graphdracula.net>
7. Arbor.js: a graph visualisation library using web workers and query (2013), <http://arborjs.org>
8. de Jong, J.: The vis.js library (2013), <http://visjs.org/>

Exact and Fixed-Parameter Algorithms for Metro-Line Crossing Minimization Problems[★]

Yoshio Okamoto¹, Yuichi Tatsu², and Yushi Uno²

¹ The University of Electro-Communications,
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
okamotoy@uec.ac.jp

² Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai 599-8531, Japan
sr301023@edu.osakafu-u.ac.jp, uno@mi.s.osakafu-u.ac.jp

The *metro-line crossing minimization problem* (MLCM), proposed by Benkert et al. [3], is to draw multiple lines on a fixed drawing of an underlying graph that models stations and rail tracks, so that the number of crossings of lines is minimum. The input of MLCM is defined by an underlying graph G and a line set \mathcal{L} on G . This paper studies a variation of MLCM, called MLCM-P, with the restriction that line terminals have to be drawn at a verge of a station (*periphery condition*), which is known to be NP-hard even when its underlying graphs are paths [1]. In this paper we focus on such cases of MLCM-P, which we call MLCM-P_PATH, and present the following three results.

1. Planarity in MLCM-P_PATH. Our first result is a linear-time algorithm for deciding whether a given instance of MLCM-P_PATH has a layout without crossings. This is in a contrast with the NP-hardness of MLCM-P_PATH. Recently, Fink and Pupyrev [4] independently gave an algorithm for the same problem when a given graph is not necessarily a path but any planar graph. However, its running time is $O(|E(G)||\mathcal{L}|^2)$.

We now describe the outline of our algorithm. First, we reduce our problem to the following PLANARITY INSIDE CIRCLE (PIC) problem in linear time. As the second step, we reduce PIC to the usual planarity testing. In the PIC problem, we are given a graph $H = (V, E)$ and a bijection $\delta: V \rightarrow \{1, 2, \dots, |V|\}$. Then, we want to place the vertices of V on a single line in the order defined by δ and draw the edges in E and a circle passing through $\delta^{-1}(1)$ and containing all the other vertices, so that all the edges in E

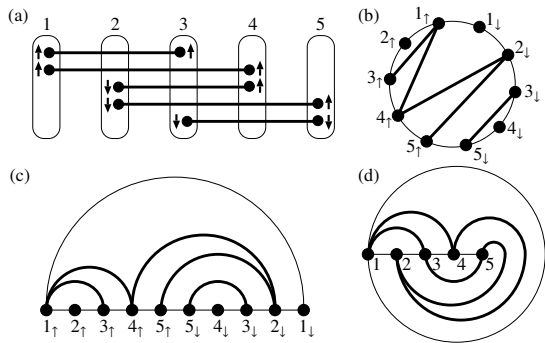


Fig. 1. (a) A layout of MLCM-P without crossings, (b) a circular drawing, (c) a transformation into a half-circle drawing, (d) a drawing inside a circle

are drawn within the circle without crossings. If such a drawing exists, the output is yes; otherwise, the output is no. The reduction at the first step is illustrated in Fig. 1. At the

[★] For more details, refer to the full version [6].

second step, to force the drawing to reside in a circle, we use K_4 , which is a minimal non-outerplanar graph.

II. An Exact Exponential Algorithm for MLCM-P_PATH. The second result is an $O^*(2^{|\mathcal{L}|})$ -time exact algorithm for MLCM-P_PATH.¹ A naive approach is to compute the number of crossings for all possible layouts of lines in \mathcal{L} by using an algorithm of Bekos et al. [2], and then to output an optimal one among them. Since the number of different assignments of two ends (to top or bottom) of a line is four, this approach yields an algorithm running in $O^*(4^{|\mathcal{L}|})$ time. To improve the running time, we look at all possible assignments of left ends of lines and greedily determine the assignments of right ends for each assignment of left ends so that the number of crossings is minimum. A lemma guarantees the correctness.

III. Fixed-Parameter Tractability for MLCM-P_PATH. The third result is an $O^*(2^k)$ -time exact algorithm for MLCM-P_PATH, where k is the multiplicity of lines, that is, the maximum number of lines on an edge of an underlying graph. Our result partially answers a question by Nöllenburg [5]. A recent paper by Fink and Pupyrev [4] independently gave an $O^*((k!)^2)$ -time algorithm for the same problem with the same parameter. Thus, our algorithm is superior in its running time.

A naive approach performs dynamic programming along the path, and at each vertex of the path we look at all possible pairs of permutations of lines. Since the multiplicity is k , this leads to an $O^*((k!)^2)$ -time algorithm. To improve the running time, we exploit several non-trivial properties of optimal solutions. For example, the algorithm by Bekos et al. [2] implies that among the optimal layouts there exists one that satisfies the following condition: If two lines $l = [i, j]$ and $l' = [i', j']$ ($j < j'$) cross, then the crossing occurs on edge $(j - 1, j)$ of the path. Such properties reduce the number of permutations we should look at, and yield an $O^*(2^k)$ -time algorithm.

References

1. Argyriou, E., Bekos, M.A., Kaufmann, M., Symvonis, A.: Two polynomial time algorithms for the metro-line crossing minimization problem. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 336–347. Springer, Heidelberg (2009)
2. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Line crossing minimization on metro maps. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 231–242. Springer, Heidelberg (2008)
3. Benkert, M., Nöllenburg, M., Uno, T., Wolff, A.: Minimizing intra-edge crossings in wiring diagrams and public transportation maps. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 270–281. Springer, Heidelberg (2007)
4. Fink, M., Pupyrev, S.: Metro-line crossing minimization: hardness, approximations, and tractable cases. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 328–339. Springer, Heidelberg (2013)
5. Nöllenburg, M.: An improved algorithm for the metro-line crossing minimization problem. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 381–392. Springer, Heidelberg (2010)
6. Okamoto, Y., Tatsu, Y., Uno, Y.: Exact and fixed-parameter algorithms for metro-line crossing minimization problems. arXiv:1306.3538 (cs.DS) (2013)

¹ The O^* -notation ignores a polynomial factor, commonly used in the exponential-time algorithm literature.

Convex-Arc Drawings of Pseudolines^{*}

David Eppstein¹, Mereke van Garderen²,
Bettina Speckmann², and Torsten Ueckerdt³

¹ Computer Science Dept., University of California, Irvine, USA
eppstein@uci.edu

² Dept. of Mathematics and Computer Science, TU Eindhoven, the Netherlands
m.v.garderen@student.tue.nl, speckman@win.tue.nl

³ Dept. of Mathematics, KIT, Germany
torsten.ueckerdt@kit.edu

Introduction. A *pseudoline* is formed from a line by stretching the plane without tearing: it is the image of a line under a homeomorphism of the plane [13]. In *arrangements* of pseudolines, pairs of pseudolines intersect at most once and cross at their intersections. Pseudoline arrangements can be used to model sorting networks [1], tilings of convex polygons by rhombi [4], and graphs that have distance-preserving embeddings into hypercubes [6]. They are also closely related to oriented matroids [11]. We consider here the visualization of arrangements using well-shaped curves.

Primarily, we study *weak outerplanar* pseudoline arrangements. An arrangement is *weak* if it does not necessarily have a crossing for every pair of pseudolines [12], and *outerplanar* if every crossing is part of an unbounded face of the arrangement. We show that these arrangements can be drawn with all curves convex, either as polygonal chains with at most two bends per pseudoline or as semicircles above a line. Arbitrary pseudolines can also be drawn as convex curves, but may require linearly many bends.

Related Work. Several results related to the visualization of pseudoline arrangements are known. In *wiring diagrams*, pseudolines are drawn on parallel horizontal lines, with crossings on short line segments that connect pairs of horizontal lines [10]. The graphs of arrangements have drawings in small grids [8] and the dual graphs of weak arrangements have drawings in which each bounded face is centrally symmetric [5]. The pseudoline arrangements in which each pseudoline is a translated quadrant can be used to visualize *learning spaces* representing the states of a human learner [7]. Researchers in graph drawing have also studied force-directed methods for schematizing systems of curves representing metro maps by replacing each curve by a spline; these curves are not necessarily pseudolines, but they typically have few crossings [9].

Results. Below we state our results for outerplanar and arbitrary arrangements.

Theorem 1. *Every weak outerplanar pseudoline arrangement may be represented by a set of chords of a circle.*

Corollary 1. *Every weak outerplanar pseudoline arrangement may be represented by lines in the hyperbolic plane, or by semicircles with endpoints on a common line.*

* Parts of this work originated at Dagstuhl seminar 13151, *Drawing Graphs and Maps with Curves*. D.E. was supported in part by the National Science Foundation under grants 0830403 and 1217322, and by the Office of Naval Research under MURI grant N00014-08-1-1015.

This result complements the fact that a weak arrangement with no 3-clique can always be represented by hyperbolic lines, regardless of outerplanarity [2].

Corollary 2. *Every weak outerplanar pseudoline arrangement may be represented by convex polygonal chains with only two bends.*

Theorem 2. *Every n -element pseudoline arrangement can be drawn with convex polylines, each of complexity at most n .*

For smooth curves composed of multiple circular arcs and straight line segments, Bekos et al. [3] defined the *curve complexity* to be the maximum number of arcs and segments in a single curve. By replacing each bend of the above result by a small circular arc, one obtains a smooth convex representation of the arrangement with curve complexity $O(n)$. We can show that these bounds are optimal.

Theorem 3. *There exist simple arrangements of n pseudolines that, when represented by polygonal chains require some pseudolines to have $\Omega(n)$ bends.*

Theorem 4. *There exist simple arrangements of n pseudolines that, when represented by smooth piecewise-circular curves require some curves to have $\Omega(n)$ arcs.*

References

1. Angel, O., Holroyd, A.E., Romik, D., Virág, B.: Random sorting networks. *Advances in Mathematics* 215(2), 839–868 (2007)
2. Bandelt, H.-J., Chepoi, V., Eppstein, D.: Combinatorics and geometry of finite and infinite squaregraphs. *SIAM Journal of Discrete Mathematics* 24(4), 1399–1440 (2010)
3. Bekos, M.A., Kaufmann, M., Kobourov, S.G., Symvonis, A.: Smooth orthogonal layouts. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 150–161. Springer, Heidelberg (2013)
4. da Silva, I.P.F.: On fillings of $2N$ -gons with rhombi. *Disc. Math.* 111(1-3), 137–144 (1993)
5. Eppstein, D.: Algorithms for drawing media. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 173–183. Springer, Heidelberg (2005)
6. Eppstein, D.: Cubic partial cubes from simplicial arrangements. *Electronic Journal of Combinatorics* 13(1), 79 (2006)
7. Eppstein, D.: Upright-quad drawing of st -planar learning spaces. *Journal of Graph Algorithms and Applications* 12(1), 51–72 (2008)
8. Eppstein, D.: Drawing arrangement graphs in small grids, or how to play Planarity. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 436–447. Springer, Heidelberg (2013)
9. Fink, M., Haverkort, H., Nöllenburg, M., Roberts, M., Schuhmann, J., Wolff, A.: Drawing metro maps using bézier curves. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 463–474. Springer, Heidelberg (2013)
10. Goodman, J.E.: Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics* 32(1), 27–35 (1980)
11. Goodman, J.E.: Pseudoline arrangements. In: *Handbook of Discrete and Computational Geometry*, page 97 (2010)
12. Goodman, J.E., Pollack, R.: Allowable sequences and order types in discrete and computational geometry. In: Pach, J. (ed.) *New Trends in Discrete and Computational Geometry. Algorithms and Combinatorics*, vol. 10, pp. 103–134. Springer, Heidelberg (1993)
13. Shor, P.W.: Stretchability of pseudolines is NP-hard. In: Gritzmann, P., Sturmfels, B. (eds.) *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 4, pp. 531–554. Amer. Math. Soc., Providence (1991)

The Density of Classes of 1-Planar Graphs^{*}

Christopher Auer, Christian Bachmaier, Franz J. Brandenburg,
Andreas Gleißner, Kathrin Hanauer, and Daniel Neuwirth

University of Passau, 94030 Passau, Germany

{auerc,bachmaier,brandenb,gleissner,hanauer,neuwirth}@fim.uni-passau.de

The *density* of a graph $G = (V, E)$ is the number of edges $|E|$ as a function of the number of vertices $n = |V|$. It is an important graph parameter, and is often used to exclude a graph from a particular class. We survey the density of relevant subclasses of 1-planar graphs and establish some new and improved bounds. A graph is *1-planar* if it can be drawn in the plane such that each edge is crossed at most once.

We consider simple and connected graphs. A graph $G \in \mathcal{G}$ is *maximal* for a particular class of graphs \mathcal{G} if the addition of any edge e implies $G + e \notin \mathcal{G}$. Let $M(\mathcal{G}, n)$ and $m(\mathcal{G}, n)$ denote the *maximum* and *minimum numbers* of edges of a maximal n -vertex graph in \mathcal{G} . Graphs $G \in \mathcal{G}$ with density $M(\mathcal{G}, |G|)$ ($m(\mathcal{G}, |G|)$) are the *densest* (*sparsest maximal*) graphs of \mathcal{G} . Thus $M(\mathcal{G}, n)$ is an upper and $m(\mathcal{G}, n)$ a lower bound. It is well-known that $M(\mathcal{G}, n)$ and $m(\mathcal{G}, n)$ coincide for planar, bipartite planar, and outerplanar graphs with $3n - 6$, $2n - 4$, and $2n - 3$, respectively. For 1-planar graphs the upper and lower bounds diverge. $M(\mathcal{G}, n) = 4n - 8$ was proved first of all by Bodendiek et al. [3] and was rediscovered several times. Surprisingly, there are much sparser maximal 1-planar graphs that are even sparser than maximum planar graphs. In [4] it was proved that $\frac{28}{13}n - \mathcal{O}(1) \leq m(\mathcal{G}, n) \leq \frac{45}{17}n - \mathcal{O}(1)$.

We consider the density of maximal graphs of subclasses of 1-planar graphs, with emphasis on sparse graphs. Our focus is on 3-connected [1], bipartite [7, 8], and outer 1-planar [2] graphs. An *outer 1-planar* graph is drawn with all vertices in the outer face. Moreover, we restrict the drawings by fixed *rotation systems*, which specify the cyclic ordering of the edges at each vertex, and then may allow crossings of incident edges, which are generally excluded for 1-planarity.

Theorem 1. *For the classes of graphs \mathcal{G} from Table 1, the stated upper bound on $M(\mathcal{G}, n)$ on the density is tight. The minimum density $m(\mathcal{G}, n)$ ranges between the functions in column “lower example” and “lower bound m ”.*

3-connected. For 3-connected 1-planar graphs \mathcal{G} the upper bound is obvious. The lower bound $m(\mathcal{G}, n) = \frac{10}{3}n + \frac{20}{3}$ is tight. It improves the example of $3.625n + \mathcal{O}(1)$ from [6] and disproves their conjecture of $3.6n + \mathcal{O}(1)$.

A graph $G \in \mathcal{G}$ consists of non-planar K_4 s and a planar remainder, which is triangulated such that two adjacent triangles imply a K_4 . The removal of all pairs of crossing edges from G leaves a planar graph with t triangles and q quadrangles and the relation $t \leq q$, which together with Euler’s formula yields the

^{*} Research partially supported by the German Science Foundation, DFG, Grant Br-835/18-1

Table 1. Upper and lower bounds on the number of edges in maximal graphs

	upper bound M	lower example	lower bound m
2-connected	$4n - 8$ [3]	$\frac{45}{17}n - \frac{84}{17}$ [4]	$\frac{28}{13}n - \frac{10}{3}$ [4]
3-connected	$4n - 8$ [3]	$\frac{10}{3}n - \frac{20}{3}$	$\frac{10}{3}n - \frac{20}{3}$
straight-line	$4n - 9$ [5]	$\frac{8}{3}n - \frac{11}{3}$	$\frac{28}{13}n - \frac{10}{3}$ [4]
fixed rotation, 2-connected	$4n - 8$ [3]	$\frac{7}{3}n - 3$ [4]	$\frac{21}{10}n - \frac{10}{3}$ [4]
fixed rotation, intersect incident	$4n - 8$ [3]	$\frac{3}{2}n + 1$	$\frac{5}{4}n$
bipartite	$3n - 8$ [7]	$n - 1$	$n - 1$
bipartite, 2-connected	$3n - 8$ [7]	$2n - 4$	n
outer 1-planar	$\frac{5}{2}n - 4$	$\frac{11}{5}n - \frac{18}{5}$	$\frac{11}{5}n - \frac{18}{5}$

bound for $m(\mathcal{G}, n)$. The bound is achieved by a recursive construction of planar K_4 s surrounded by non-planar K_4 s surrounded by planar K_4 s.

Outer 1-planar. A maximal outer 1-planar graph G is composed of planar K_3 s and non-planar K_4 s [2], such that two K_3 s are not adjacent. Removing the pairs of crossing edges from the K_4 s results in an outerplanar graph whose dual is a tree with vertices of degree 3 and 4. Each vertex of degree 3 adds one vertex and two edges, and each vertex of degree 4 adds two vertices and five edges to the density of G . Maximizing the degree-4 vertices yields $M(\mathcal{G}, n) = \frac{5}{2}n - 4$ and minimizing yields $m(\mathcal{G}, n) = \frac{11}{5}n - \frac{18}{5}$. Both bounds are tight.

References

1. Alam, M.J., Brandenburg, F.J., Kobourov, S.G.: Straight-line drawings of 3-connected 1-planar graphs. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 83–94. Springer, Heidelberg (2013)
2. Auer, C., Bachmaier, C., Brandenburg, F.J., Gleißner, A., Hanauer, K., Neuwirth, D., Reislhuber, J.: Recognizing outer 1-planar graphs in linear time. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 107–118. Springer, Heidelberg (2013)
3. Bodendiek, R., Schumacher, H., Wagner, K.: Über 1-optimale Graphen. *Mathematische Nachrichten* 117, 323–339 (1984)
4. Brandenburg, F.J., Eppstein, D., Gleißner, A., Goodrich, M.T., Hanauer, K., Reislhuber, J.: On the density of maximal 1-planar graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 327–338. Springer, Heidelberg (2013)
5. Didimo, W.: Density of straight-line 1-planar graph drawings. *Inform. Process. Lett.* 113(7), 236–240 (2013)
6. Hudák, D., Madaras, T., Suzuki, Y.: On properties of maximal 1-planar graphs. *Discuss. Math. Graph Theory* 32(4), 737–747 (2012)
7. Karpov, D.V.: Upper bound on the number of edges of an almost planar bipartite graph. Tech. Rep. arXiv:1307.1013v1 (math.CO), Computing Research Repository (CoRR) (July 2013)
8. Xu, C.: Algorithmen für die Dichte von Graphen. Master’s thesis, University of Passau (2013)

BGPlay3D: Exploiting the Ribbon Representation to Show the Evolution of Interdomain Routing

Patrizio Angelini, Lorenzo Antonetti Clarucci, Massimo Candela,
Maurizio Patrignani, Massimo Rimondini, and Roberto Sepe

Roma Tre University, Italy

Representing flows in networks is a challenging task: different flows usually traverse the same edges; flows may split and join again along their routes; and some flows may go so far as to traverse the same nodes and edges several times. Traditional 2D visualizations exploit drawing techniques like parallel multicolor curves to address these challenges and tell apart the different flows, but they tend to exhibit readability problems, for example when lots of flows traverse a single edge.

We investigate a novel 2.5D visualization metaphor, conceived to clearly distinguish flows in networks. Instead of drawing flows as parallel curves on the plane, we use the third dimension to stack the flows one above the other. Namely, in a *ribbon representation* each node of the network is represented by a pin, and each flow is represented by a ribbon (a long stripe) winding around the pins corresponding to the nodes traversed by the flow [1]. Pins are distributed on a flat surface, yielding a 2.5D representation rather than a pure 3D one.

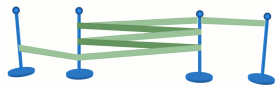


Fig. 2. A ribbon representation of a cycling flow

Finally, when the thickness of the ribbon allows it, labels may be accommodated in the ribbons themselves.

On the other hand, we expect the ribbon representation to have some intrinsic limitations: navigation in a 3D scene may be less intuitive for the user than panning and zooming a 2D representation. Also, when the network is big, occlusions among pins and ribbons tend to hamper a clear perception of the 3D environment.

In order to explore both the potentialities and the limitations of using the ribbon representation to depict flows in a network, compared with a traditional 2D drawing and interface, we realized a tool, called BGPlay3D [7], which extends with a 3D interface the functionalities of the well-known BGPlay tool [6]. We took strong advantage of the modularity of the BGPlay.js JavaScript open-source framework, which offers the basis for several network visualization tools [4,3,5], including the latest release of BGPlay: starting from the latter, we replaced the existing GraphView module, responsi-

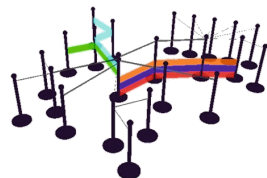


Fig. 1. A ribbon representation of a flow in a small network

The potentialities of the ribbon representation with respect to a traditional 2D visualization are manifold: the order in which ribbons are stacked, as well as their thickness, may correspond to extra information associated with the ribbons themselves. Even self-intersecting flows are clearly representable (see Fig. 2). Furthermore, colors or shades of grey are not strictly needed anymore to tell the flows apart and can be therefore used with different semantics.

ble of managing events in the canvas area of the interface, with a GraphView3D module that exploits the three.js lightweight cross-browser JavaScript API [8] to create and display animated 3D computer graphics on a Web browser (three.js, in turn, relies on the WebGL JavaScript API for rendering interactive 3D graphics [9]).

The interface of BGPlay3D, like that of BGPlay, is vertically split into three panels: an information panel on the top, the main visualization window in the middle, and a control timeline at the bottom (see Fig. 3). In the visualization window the activity graph evolves based on the current event. Namely, new flows appear, old flows disappear, and some flow changes its route. Like in the 2D interface, appearances are handled by flashing the new ribbons. Flows that disappear are represented by flashing the corresponding ribbons and then letting them drop on the floor. Changes in the route of a flow have a special representation in 3D: rather than morphing the old route into the new one as it happened in the 2D interface, we let the old ribbon disappear towards the destination just as if it was a cut film, while the new ribbon originates from the source and moves towards the destination along the new route. A prototype of BGPlay3D, offering a promising alternative to BGPlay, is available online [2].

Several further directions of research remain open. For example, BGPlay3D uses the same algorithm as BGPlay for laying out the pins on the plane and no specialized algorithm has been developed to take into account the 3D nature of the interface. Also, the position of the camera is chosen by the user, and no attempt has been made to offer a point of view which is more convenient for understanding the current events.

References

1. Antonetti Clarucci, L.: La ribbon representation: un nuovo modello di visualizzazione di flussi e sue applicazioni. Master's thesis, Roma Tre University, Rome, Italy (2013) (in Italian)
2. BGPlay3D, <http://bgplayjs.com/?section=bgplay3d> (accessed 2013)
3. BGPlay.js, <http://bgplayjs.com/> (accessed 2013)
4. Candela, M.: Adaptive and responsive web-oriented visualization of evolving data: the inter-domain routing case. Master's thesis, Roma Tre University, Rome, Italy (2012)
5. Candela, M., Di Bartolomeo, M., Di Battista, G., Squarcella, C.: Dynamic traceroute visualization at multiple abstraction levels. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 496–507. Springer, Heidelberg (2013)
6. Colitti, L., Di Battista, G., Mariani, F., Patrignani, M., Pizzonia, M.: Visualizing interdomain routing with BGPlay. *Journal of Graph Algorithms and Applications* 9(1), 117–148 (2005)
7. Sepe, R.: Visualizzazione tridimensionale del routing interdominio. Master's thesis, Roma Tre University, Rome, Italy (2013) (in Italian)
8. three.js, <http://threejs.org/> (accessed 2013)
9. WebGL (Web Graphics Library), <http://www.khronos.org/webgl/> (accessed 2013)

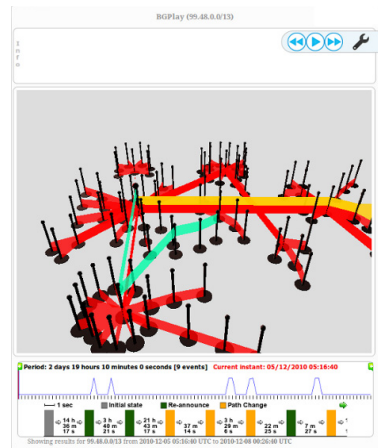


Fig. 3. The interface of BGPlay3D

Ravenbrook Chart: A New Library for Graph Layout and Visualisation

Nick Levine and Nick Barnes

Ravenbrook Limited, PO Box 205, Cambridge, CB2 1AN, United Kingdom
chart@ravenbrook.com

Abstract. *Ravenbrook Chart* is a newly-available library which implements layout and interactive display of large, complex graphs. It provides spring-embedded, layered and circular layouts. It has been deployed in mature applications and a free web service uses it to visualise graphs. It is now available free of charge as a permissively licensed library.

Between 2001 and 2013, the authors developed two desktop applications whose requirements included interactive graph layout and display. We assessed several commercial graphing products, and indeed used two such libraries in production versions at different times. This approach was abandoned for commercial and technical reasons: no third-party tool met all our requirements for reliability, performance, and flexible interactivity. In 2009 work began on a new layout and display library, *Ravenbrook Chart*. Its implementation was led by customer requirements and it is now used for visualisation in both applications. More recent uses of this library include *Chart Server* and *Chart Desktop* both of which, as described here, are now available for use free of charge.

Chart implements three layout types: spring-embedded (see Fig. 1), hierarchical, and circular. The time complexities of the algorithms are $O(v \log v + e \log e)$ for the circular layout, $O((v + e)^{1.5})$ for hierarchical, and $O(v^2 \log v)$ for spring-embedded, where v and e are the numbers of nodes and edges respectively.

Chart's visualisation system supports multiple graphs each of which can be shown on any number of displays. Displays can be "aligned" with each other, so that they automatically pan and zoom in tandem. The library supports standard controls over node and edge appearance, UTF-8 strings throughout, a wide range of callbacks, dynamic context-sensitive menus, incremental changes to the graph, retrieval and setting of node and edge-bend locations (per display), and object hiding. It is designed for high data throughput, full thread-safety, and sophisticated exception handling with logging, restarts, and detailed backtraces.

Chart was originally created for two custom desktop tools. The *Critical Network Analysis Tool* is a U.S. Government off-the-shelf application for processing, visualising and analysing large social networks. It includes a large suite of analytical tools based on social network theory, including many numerical and visualisation tools independent of graph drawing. *WorldView* is used to construct, analyse, and compare "cognitive maps", identifying and contrasting core beliefs of individuals and groups based on texts such as interviews and speeches.

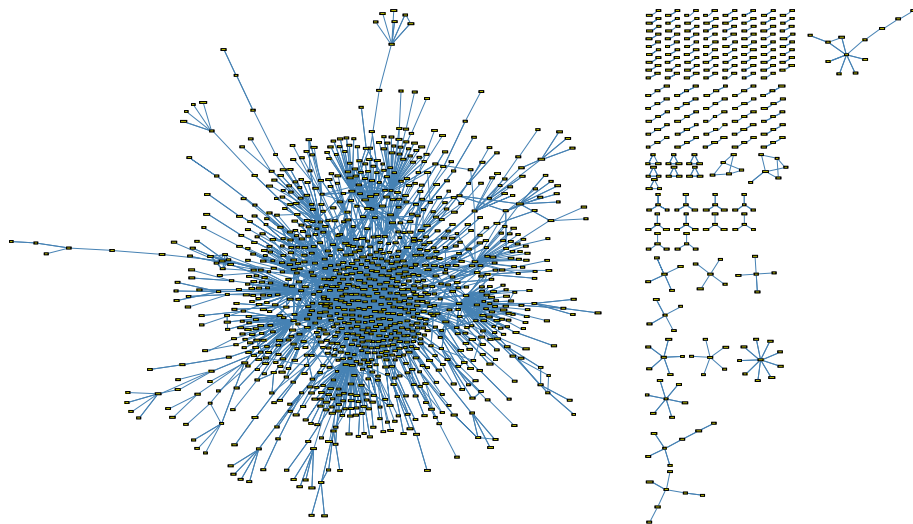


Fig. 1. Spring-embedded layout of sample social network: 1513 nodes, 3359 edges. The component on the left has 1149 nodes and 2959 edges. The two most-connected nodes have degrees d of 342 and 131; 37 nodes have $d > 10$.

Chart server (<http://chart.ravenbrook.com>) is a free web service for converting graph specifications into images in either PNG or PDF format. Its straightforward API supports GET and POST requests, all three layouts, and a rich control over node and edge appearance.

Chart Desktop is a freely-available version of the Chart library, distributed as a Windows DLL. Much of the Chart functionality is not exposed through the API in the initial Chart Desktop release; the focus has been on careful, comprehensive documentation and testing. The library is accompanied by a C header file, a high-level Python interface, and examples. It is available free of charge under the very permissive “BSD 2-Clause” license which in particular allows deployment in closed-source commercial applications, and it can be downloaded from: <http://chart.ravenbrook.com/downloads>.

The development of Chart has always been driven by end-user requirements, using a highly responsive evolutionary delivery process. We now seek new applications, users and directions for the library. Potential improvements include planar and 3D spring-embedded layouts, improved graphic quality, and further work on performance. The Chart Desktop API will gradually be enriched to support more of Chart’s functionality, possibly supported by crowd-funding. Priorities for all this work will be set by users.

We are very pleased to offer up our work for public use. *Chart Desktop* differs from other available components in being a library rather than an application, in being flexibly licensed, and in being developed by a small company which can be highly responsive to users’ needs.

Small Grid Embeddings of Prismatoids and the Platonic Solids*

Alexander Igamberdiev, Finn Nielsen, and André Schulz

Institut für Mathematische Logik und Grundlagenforschung,
Universität Münster, Germany

{alex.igamberdiev, andre.schulz}@uni-muenster.de

1 Embedding Prismatoids on the Grid

The question if every polyhedral graph can be embedded as a convex polyhedron on a polynomially sized 3d grid is one of the main open problems in lower dimensional polytope theory. Currently, the best known algorithm requires a grid of size $O(2^{7 \cdot 21n})$, for n being the number of the vertices [2]. We show that *prismatoids* (polytopes, whose graphs are coming from triangulated polygonal annuli) can be embedded as convex polyhedra on a grid of size $O(n^4) \times O(n^3) \times 1$.

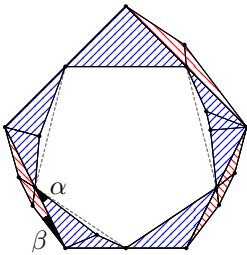


Fig. 1

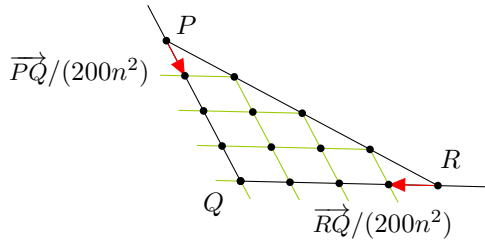


Fig. 2

We call the triangles sharing an edge with the “bottom face” *red* and all other triangles *blue*. As a first step we merge consecutive red (blue) triangles to obtain a “reduced” graph of the prismatoid with $2k$ vertices, for some $k \leq n/2$. We then embed the reduced graph as a polyhedron by drawing the bottom face as a convex k -gon in the xy -plane. The top face is realized as a k -gon inscribed in the bottom face, lifted to the $z = 1$ plane. This needs a grid of size $O(n^2) \times O(n) \times 1$ [1].

We continue by reverting the initial contractions geometrically. To do so, we have to substitute edges by polygonal (convex) chains (Fig. 1). To carry out this construction we scale the whole embedding by a factor of $200n^2$. This implies that for every triangle PQR the vectors $\overrightarrow{PQ}/(200n^2)$ and $\overrightarrow{RQ}/(200n^2)$ are integral (Fig. 2). Thus, they define a basis for a grid contained in \mathbf{Z}^2 that is just large enough to add the missing chains back.

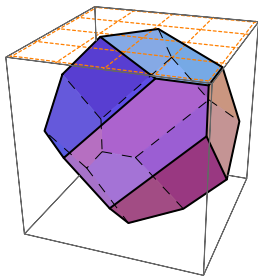
Note that we have to guarantee (for convexity) as additional constraint that for every edge separating a red and a blue triangle, the incident angle in the red

* This work was funded by the German Research Foundation (DFG) under grant SCHU 2458/2-1.

triangle is smaller than the incident angle in the blue triangle ($\beta < \alpha$, Fig. 2). This can be achieved by carefully choosing the slope of the first segment that defines the angle inside the blue triangle.

2 Grid Embeddings of the Platonic Solids

We want to realize the Platonic solids as (combinatorially equivalent) convex polyhedra on a small grid. The tetrahedron, the cube and the octahedron have all a natural realization on the $1 \times 1 \times 1$, resp., on the $1 \times 1 \times 2$ grid. A first nontrivial case is the icosahedron. Its graph can be embedded as a polyhedron using the coordinates $\{(0, \pm 2, \pm 1), ((\pm 2, \pm 1, 0), (\pm 1, 0, \pm 2))\}$. It is possible to reduce the grid size a bit further on one axis. Embedding the dodecahedron is more challenging. The best known realization so far requires a $8 \times 6 \times 4$ grid and is due to Francisco Santos [2]. We did a computer search to find smaller realizations. We had to make some assumptions to search efficiently. First, we fixed the grid size. Then we assumed that one of the faces lies in the xy -plane.



The coordinates of the vertices are:

$$\begin{aligned} &\pm\{(2, 2, 2), (2, 2, 1), (2, 1, 2), (1, 2, 2), \\ &(2, -1, 0), (2, 0, -1), (-1, 2, 0), \\ &(0, 2, -1), (0, -1, 2), (-1, 0, 2)\} \end{aligned}$$

Fig. 3. The smallest embedding of the dodecahedral graph as a convex polyhedron

We tried next to locate the remaining vertices face by face. The combinatorial structure of the dodecahedral graph helped us to limit the search space. In particular, most vertices are restricted to lie on a plane determined by previous choices. We were able to compute a realization on the $4 \times 4 \times 4$ grid as shown in Fig. 3. This result was found within seconds. We scanned through 4.367 choices for the first face. Our experiments have proven that this is the smallest realization in terms of minimizing the largest axis of the grid. Francisco Santos found another solution that reduces one of the axis at the expense of the others, yielding a $3 \times 11 \times 11$ grid embedding.

References

1. Andrews, G.E.: A lower bound for the volume of strictly convex bodies with many boundary lattice points. *Trans. Amer. Math. Soc.* 99, 272–277 (1961)
2. Mor, A.R., Rote, G., Schulz, A.: Small grid embeddings of 3-polytopes. *Discrete & Computational Geometry* 45(1), 65–87 (2011)

The Graph Landscape – a Visualization of Graph Properties

Peter Eades and Karsten Klein *

School of Information Technologies, The University of Sydney, Australia
{peter.eades,karsten.klein}@sydney.edu.au

Motivation

In Graph Drawing and related fields, a large number of graph sets has been collected over years, comprising both real world graphs from application areas as well as generated graphs with specific properties. Each experimental paper makes use of some of those sets or adds a new set of graphs to the existing ones. Recently, a project was started to collect those graphs in a comprehensive database [1] to make them publicly available over a common and easy to use interface for further benchmark experiments and for analysis.

For evaluations of experimental data it is of interest to know the characteristics of the benchmark set, e.g. described in terms of a set of graph properties. Such characteristics show that the set covers a specific range of properties, how the graphs are distributed over this range, and how the set differs from or overlaps with other sets. While the graph archive provides graphs plus several graph properties, it does not allow to analyze the distribution of properties within benchmark sets, or to compare those properties among different sets. We propose a web-based service that provides a visualization of graph properties, the *Graph Landscape*, and thus allows a visual analysis of benchmark sets.

Use-cases involve analyzing the properties of a local graph set in the context of one or more well-established benchmark sets. This can in particular be helpful for new sets to describe the difference and overlap in the graph characteristics compared to existing sets. As there is a variety of graph properties, but graph sets are usually generated by specifying a range for only a few of those properties, a feedback on the characteristics for the generated set and the distribution of property values will often be helpful. It might also help to improve interpretation of experimental results and to detect the shortcomings of graph sets.

Implementation

The project implementation consists of two main parts, the computational analysis of graphs and the visualization of the results for interactive analysis. The Graph Landscape visualization is implemented as a web-based service. The web-interface allows the user to select visualization views, and graphs as well as graph properties to populate the views for analysis, see Fig.1.

* K. Klein was partly supported by ARC grant H2814 A4421, Tom Sawyer Software, and NewtonGreen Software.

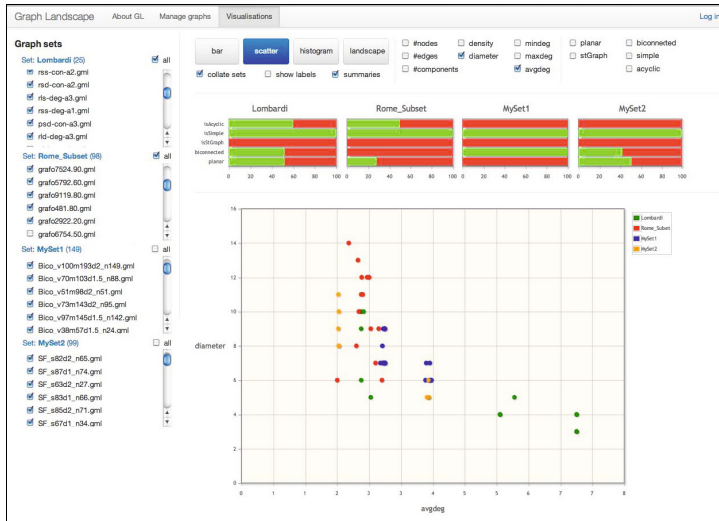


Fig. 1. User interface of the system during graph set comparison. Basic properties of the graph sets are presented in a side-by-side comparison, while user-selected properties are shown in the current main view.

Properties for graph sets available from the graph archive are precomputed, and can be extended by further analysis of other graph sources, and by uploading graphs to our server. Computation of the graph properties on the server is done by an extensible analysis engine, currently implemented as a stand-alone software using the OGDF library [2]. The handling of properties in the system is kept flexible to allow easy addition of new properties, only a few basic properties as e.g. size or connectivity are always mapped to small standard views to allow a quick overview on the graph set characteristics.

We aim at transparently connecting to the graph archive to provide up-to-date information and to allow users to retrieve or store graphs using our interface.

Remarks. We see our project as an extension to the graph archive project, with the goal to provide an easily accessible visualization that can be used as a first step in the design or analysis of benchmark sets. Our web service is still under development, but we think it should be presented to the community at this stage to allow valuable feedback and suggestions. We will have a prototype available at the Graph Drawing conference to demonstrate the system and have users try it. The feedback will then be used to guide further development decisions.

References

1. The Open Graph Archive, <https://kandinsky.informatik.uni-tuebingen.de/forschung/graphdb/>
2. The Open Graph Drawing Framework, <http://www.ogdf.net>

Application of Graph Layout Algorithms for the Visualization of Biological Networks in 3D

Tim Angus¹, Tom Freeman¹, and Karsten Klein²

¹ The Roslin Institute, University of Edinburgh, Midlothian Scotland, UK
{tim.angus,tfreeman}@roslin.ed.ac.uk

² School of Information Technologies, The University of Sydney, Australia
karsten.klein@sydney.edu.au

The visualization and analysis of biological systems and data as networks has become a hallmark of modern biology. Relationships between biological entities; individuals, proteins, genes, RNAs etc., can all be better understood at one level or another when modelled as networks. As the size of these data has grown, so has the need for better tools and algorithms to deal with the complex issue of network visualization and analysis. We describe application and evaluation of a state-of-the-art graph layout method for use within biological workflows.

BioLayout *Express*^{3D} is a powerful tool specifically designed for visualization, clustering, exploration, and analysis of very large networks in 2D and 3D space derived primarily from biological data [1]. In particular, its development has been driven by the need to analyse gene expression data, which typically consists of 10's of thousands of rows of quantitative gene expression measurements. First, the tool calculates a correlation matrix and then builds relationship networks, where nodes represent genes and edges expression similarities above a given r threshold. The resulting graphs can be very large e.g. 20-30,000 nodes, 5 million edges and possess a high degree of local structure with modules of co-expressed genes forming distinct cliques of high connectivity within the networks. BioLayout has for a long time used a modified CPU/GPU parallelised version of the Fruchterman-Reingold (FR) algorithm for graph layout, and visualization of the graphs in 3D offers distinct advantages when viewing such complex graph structures. MCL clustering is used to divide the graph into coexpression clusters for further analysis. Whilst the existing FR implementation is capable and in many ways adequate at laying out these types of graph, the results for other graphs derived from biological data are less satisfactory, in particular DNA assembly graphs, which are inherently different in structure. The overlapping nature of DNA fragments when joined based on read-similarity form 'chain graphs'. Layout using the FR algorithm places nodes efficiently on a local scale, but a lack of global awareness results in a knot-like graph structure (Figure 1A) inhibiting the efficient visualisation of the overall assembly.

The development of scalable approaches for high-quality layout computation is ongoing research in the graph drawing community. While successful results like multilevel methods are already established methods in graph drawing, they have as yet been little applied in the field of biology. In order to investigate their potential to improve the visual analysis of biological networks as described above, we integrated the fast multipole multilevel method (FMMM) in

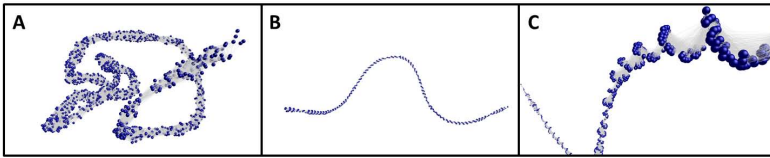


Fig. 1. Graph-based visualization of RNA-seq data for the gene BUB1. Nodes (2,886) represent individual 100 bp DNA reads and edges (132,764) the overlap between them. Graphs have appearance of a linear 'chain' reflecting the overlapping DNA sequences. (A) layout using our standard FR algorithm (B) layout using our 3D implementation of the FMMM algorithm (C) a close up of the graph shows the 'corkscrew' layout of the individual nodes.

BioLayout *Express*^{3D} such that the method can be used as part of the usual biological workflow. FMMM is available in OGDF [2], but this implementation cannot directly be used by Java tools and is also restricted to 2D drawings. We converted this implementation to Java, and extended the approach to compute 3D layouts.

We first tested this implementation of FMMM on a variety of graph types, like classic example graphs used by the graph drawing community, and were very pleased with the 3D layouts; the structure of wrapped networks being immediately obvious when visualised in 3D. After this testing, we applied it to graphs derived from biological data. In the case of graphs derived from correlation matrices the FMMM layout is quite different with major topological structures being teased apart, but the advantages of this visualisation are at this stage uncertain. The real gain comes with the DNA assembly graphs where when laid out by FR twist and turn to become knotted structures (Fig. 1A). However, when the FMMM algorithm is used the graphs are relatively linear (Fig. 1B), showing an interesting 'corkscrew' appearance at the node level (Fig. 1C). In this way the structure of the overall assembly is far more apparent and useful in cases of splice variation or other graphs structures of interest.

In summary, we are delighted in the results of this implementation, which will be available soon in the next version of BioLayout (www.biayout.org). We would like to further improve the qualitative performance for particular graphs, and investigate and discuss the potential of other methods for use with large biological networks in 3D. We believe that our tool will also prove useful for graph drawing researchers to visualise outputs of other scalable layout methods, in particular in a 3D setting.

References

1. Theocharidis, A., van Dongen, S., Enright, A., Freeman, T.C.: Network visualisation and analysis of gene expression data using Biayout *Express*^{3D}. *Nature Protocols* 4(10) (2009)
2. The Open Graph Drawing Framework, <http://www.ogdf.net>

Plane Cubic Graphs and the Air-Pressure Method*

Stefan Felsner and Linda Kleist

Institut für Mathematik, Technische Universität Berlin, Germany
{felsner, kleist}@math.tu-berlin.de

Abstract. Thomassen [1] proved that plane cubic graphs are area-universal, i.e., for a plane cubic graph G with prescribed face areas there exists a straight-line (re-)drawing G' that realizes these areas. Thomassen uses induction and proves the existence of a degenerate drawing where distinct vertices may be placed at the same position. We show that plane cubic graphs are area-universal using the air-pressure method. In [2,3], a similar method has been applied in the context of area-universality of rectangular layouts. With the poster, we give the idea of how the method can be adapted for other classes of plane graphs, in particular for plane cubic graphs.

The Air-Pressure Method

Let G be a plane graph and $a : F' \rightarrow \mathbb{R}_+$ a function that prescribes an area for each bounded face. Assume that the outer face of G is a k -gon and fix a convex k -gon Ω whose area is equal to the sum of the prescribed areas. We consider drawings of G such that the outer vertices of G are represented by the corners of Ω .

Let D be a drawing of G . For a face f_i let $m(i)$ be the area of f_i in D and let $a(i)$ be the prescribed area. The quotient $p(i) := \frac{a(i)}{m(i)}$ of these two values will be interpreted as the *pressure* in the face. Face f_i is pushing against an incident vertex v with a force $f(v, i)$ that depends on the pressure, this force is defined as $f(v, i) := p(i) \cdot n_s \|s\|$, where s is the segment connecting the two neighbors of v incident to f_i and n_s is the normal vector of s pointing out of f_i at v . The effective force acting on vertex v is $f(v) := \sum_{i:v \in f_i} f(v, i)$. The intuition is that shifting vertices in the direction of the effective force yields a better balance of pressure in the faces and hence a better approximation of the prescribed areas.

- A face f_i is in balance if $p(i) = 1$, i.e., the prescribed area is realized in the drawing.
- A vertex v is in balance if $f(v) = 0$, i.e., the effective force acting on v is neutral.
- A drawing is a *deadlock* if all vertices are in balance although there are faces f_i with $p(i) \neq 1$, (Fig. 1 shows an example).
- The *entropy of a drawing* is $E := \sum_i -a(i) \log p(i)$.

Fact: For all drawings $E \geq 0$, and $E = 0$ if and only if all faces are in balance.

An *improving shift* is a shift of an unbalanced vertex v into a new position $v + d$ such that (a) the new drawing is planar and nondegenerate, and (b) the entropy increases with the shift.

* Partially supported by ESF EuroGIGA project GraDR.

Suppose that whenever there is an unbalanced vertex v there exists a d such that $v \rightarrow v + d$ is an improving shift. Then, given any initial drawing D_0 , there is a sequence of improving shifts that yields a converging sequence of drawings $(D_i)_i$ such that in the limit D all vertices are in balance. The drawing D is either a deadlock or a drawing realizing the prescribed areas.

Plane Cubic Graphs

In the case of plane cubic graphs we can show that iterated shifting yields a limit drawing D realizing the prescribed areas. The result is proved with the following two claims:

1. All faces are in balance if and only if all vertices are in balance, i.e., there is no deadlock.
2. If there is an unbalanced vertex v , then there is a d such that $v \rightarrow v + d$ is an improving shift.

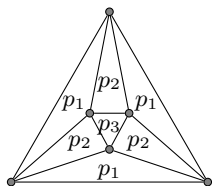


Fig. 1. Let x and 1 be the side length of the inner and outer regular triangle. If pressures satisfy $p_1 = (1 - x)p_2 + xp_3$, then this is a deadlock.

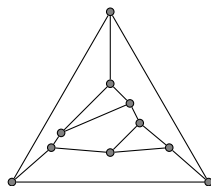


Fig. 2. This cubic graph is area-universal, it admits no deadlock.

Conclusion

We reprove area-universality for plane cubic graphs by using the air-pressure paradigm. The air-pressure method seems promising for showing area-universality for further classes of plane graphs. Time will tell how far this method can take us.

References

1. Thomassen, C.: Plane cubic graphs with prescribed face areas. *Combinatorics, Probability & Computing* 1(371-381), 2–10 (1992)
2. Felsner, S.: Exploiting air-pressure to map floorplans on point sets. In: Wismath, S., Wolff, A. (eds.) *GD 2013. LNCS*, vol. 8242, pp. 196–207. Springer, Heidelberg (2013)
3. Izumi, T., Takahashi, A., Kajitani, Y.: Air-pressure model and fast algorithms for zero-wasted-area layout of general floorplan. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 81(5), 857–865 (1998)

Author Index

- Aerts, Nieke 119
Alam, Md. Jawaherul 83
Angelini, Patrizio 37, 49, 292, 526
Angus, Tim 534
Antonetti Clarucci, Lorenzo 526
Auer, Christopher 25, 107, 516, 524
- Bachmaier, Christian 107, 516, 524
Bannister, Michael J. 208, 340
Barnes, Nick 528
Bekos, Michael A. 244, 424
Bennett, Joel 514
Bereg, Sergey 484
Biedl, Therese 460
Binucci, Carla 292
Bläsius, Thomas 220, 460
Brandenburg, Franz J. 25, 83, 107, 516, 524
Brandes, Ulrik 388
- Candela, Massimo 496, 526
Chan, Timothy M. 376
Chang, Yi-Jun 400
Chaplick, Steven 131
Cheng, Zhanpeng 208
Chimani, Markus 13
Cornelsen, Sabine 244
- Da Lozzo, Giordano 37, 292
Devanny, William E. 208
Di Bartolomeo, Marco 496
Di Battista, Giuseppe 37, 496
Didimo, Walter 292, 304
Di Giacomo, Emilio 304
Duncan, Christian A. 508
Durocher, Stephane 143
Dwyer, Tim 448
- Eades, Peter 71, 532
Eppstein, David 208, 340, 352, 436, 522
- Felsner, Stefan 119, 196, 536
Fink, Martin 244, 328
Fowler, J. Joseph 155
Fрати, Fabrizio 1, 37, 49
- Freeman, Tom 534
Fulek, Radoslav 131
- Gansner, Emden R. 268
Garderen, Mereke van 522
Gleißner, Andreas 25, 107, 516, 524
Goodrich, Michael T. 161, 256
Grilli, Luca 292
Gutwenger, Carsten 508
- Hanauer, Kathrin 25, 107, 524
Hoffmann, Hella-Franziska 376
Holroyd, Alexander E. 484
Holten, Danny 352
Hong, Seok-Hee 71, 244
Hu, Yifan 268
Hurtado, Ferran 280
- Igamberdiev, Alexander 173, 530
- Jong, Jos de 518
- Karrer, Annette 220
Katoh, Naoki 71
Kaufmann, Michael 1, 244, 424
Kiazyk, Stephen 376
Kieffer, Steve 448
Klavík, Pavel 131
Klein, Karsten 532, 534
Kleist, Linda 536
Kobourov, Stephen G. 83
Korman, Matias 280
Krevelde, Marc van 280
Krishnan, Shankar 268
Krug, Robert 424
- Lenhart, William 412
Levine, Nick 528
Liotta, Giuseppe 71, 304, 412
Löffler, Maarten 280, 352
Lubiw, Anna 376
- Marriott, Kim 448
Matoušek, Jiří 472
Mchedlidze, Tamara 316
Mondal, Debajyoti 143, 412
Montecchiani, Fabrizio 292, 304

- Mubayi, Dhruv 364
 Munzner, Tamara 61

 Nachmanson, Lev 484, 508
 Näher, Stefan 424
 Neuwirth, Daniel 107, 524
 Niedermann, Benjamin 460
 Nielsen, Finn 530
 Nishat, Rahnuma Islam 412
 Nocaj, Arlind 388
 Nöllenburg, Martin 244, 316, 352, 460

 Okamoto, Yoshio 520

 Pach, János 1
 Patrignani, Maurizio 49, 292, 526
 Pазienza, Giovanni E. 518
 Prutkin, Roman 460
 Pszona, Paweł 161, 256
 Pupyrev, Sergey 328, 484
 Purchase, Helen C. 232

 Reislhuber, Josef 107, 516
 Rimondini, Massimo 526
 Roselli, Vincenzo 49, 424
 Rutter, Ignaz 220, 244, 316, 460

 Sacristán, Vera 280
 Sander, Georg 508
 Schaefer, Marcus 185
 Schulz, André 173, 530

 Schweitzer, Pascal 71
 Sedgwick, Eric 472
 Sepe, Roberto 526
 Silveira, Rodrigo I. 280
 Simons, Joseph A. 340
 Speckmann, Bettina 280, 352, 522
 Squarcella, Claudio 496
 Štefankovič, Daniel 185
 Suk, Andrew 95, 364
 Suzuki, Yusuke 71
 Symvonis, Antonios 244

 Tancer, Martin 472
 Tatsu, Yuichi 520
 Tollis, Ioannis G. 292, 304
 Tóth, Csaba D. 1

 Ueckerdt, Torsten 522
 Uno, Yushi 520

 Verbeek, Kevin 352

 Wagner, Uli 472
 Walczak, Bartosz 95
 Wismath, Stephen 514
 Wood, David R. 1
 Wybrow, Michael 448

 Yen, Hsu-Chun 400

 Zeranski, Robert 13