Computational Geometry 55 (2016) 26–40

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Orthogonal graph drawing with inflexible edges ☆, ☆☆

Thomas Bläsius [a,b,*], Sebastian Lehmann [a], Ignaz Rutter [a,*]

[a] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[b] Hasso Plattner Institute, Potsdam, Germany

ARTICLE INFO

Article history:
Received 19 June 2015
Received in revised form 23 December 2015
Accepted 26 February 2016
Available online 3 March 2016

Keywords:
Orthgonal graph drawing
Bend minimization
Planar embedding
Parameterized algorithm
Computational complexity

ABSTRACT

We consider the problem of creating plane orthogonal drawings of *4-planar graphs* (planar graphs with maximum degree 4) with constraints on the number of bends per edge. More precisely, we have a *flexibility function* assigning to each edge e a natural number $\text{flex}(e)$, its *flexibility*. The problem FlexDraw asks whether there exists an orthogonal drawing such that each edge e has at most $\text{flex}(e)$ bends. It is known that FlexDraw is NP-hard if $\text{flex}(e) = 0$ for every edge e [1]. On the other hand, FlexDraw can be solved efficiently if $\text{flex}(e) \geq 1$ [2] and is trivial if $\text{flex}(e) \geq 2$ [3] for every edge e.

To close the gap between the NP-hardness for $\text{flex}(e) = 0$ and the efficient algorithm for $\text{flex}(e) \geq 1$, we investigate the computational complexity of FlexDraw in case only few edges are *inflexible* (i.e., have flexibility 0). We show that for any $\varepsilon > 0$ FlexDraw is NP-complete for instances with $O(n^{\varepsilon})$ inflexible edges with pairwise distance $\Omega(n^{1-\varepsilon})$ (including the case where they induce a matching), where n denotes the number of vertices in the graph. On the other hand, we give an FPT-algorithm with running time $O(2^k \cdot n \cdot T_{\text{flow}}(n))$, where $T_{\text{flow}}(n)$ is the time necessary to compute a maximum flow in a planar flow network with multiple sources and sinks, and k is the number of inflexible edges having at least one endpoint of degree 4.

1. Introduction

Bend minimization in orthogonal drawings is a classical problem in the field of graph drawing. We consider the following problem called OptimalFlexDraw. The input is a 4-planar graph G (from now on all graphs are 4-planar) together with a cost function $\text{cost}_e\colon \mathbb{N} \to \mathbb{R} \cup \{\infty\}$ assigned to each edge. We want to find an orthogonal drawing Γ of G such that $\sum \text{cost}_e(\beta_e)$ is minimal, where β_e is the number of bends of e in Γ. The basic underlying decision problem FlexDraw restricts the cost function of every edge e to $\text{cost}_e(\beta) = 0$ for $\beta \in [0, \text{flex}(e)]$ and $\text{cost}_e(\beta) = \infty$ otherwise, and asks whether there exists a *valid* drawing (i.e., a drawing with finite cost). The value $\text{flex}(e)$ is called the *flexibility* of e. Edges with flexibility 0 are called *inflexible*.

Note that FlexDraw represents the important base case of testing for the existence of a drawing with cost 0 that is included in solving OptimalFlexDraw.

Garg and Tamassia [1] show that FlexDraw is NP-hard in this generality, by showing that it is NP-hard if every edge is inflexible. For special cases, namely planar graphs with maximum degree 3 and series-parallel graphs, Di Battista et al. [4]

☆ Partially supported by grant WA 654/21-1 of the German Research Foundation (DFG).

☆☆ A preliminary version of this paper has appeared as T. Bläsius, S. Lehmann, I. Rutter, Orthogonal graph drawing with inflexible edges, in: Proceedings of the 9th International Conference on Algorithms and Complexity, in: Lecture Notes in Computer Science, vol. 9070, Springer, 2015, pp. 153–166.

* Corresponding authors.
E-mail addresses: thomas.blaesius@hpi.de (T. Bläsius), sebastian@leemes.de (S. Lehmann), rutter@kit.edu (I. Rutter).

http://dx.doi.org/10.1016/j.comgeo.2016.03.001
0925-7721/

give an algorithm minimizing the total number of bends, which solves OptimalFlexDraw with $\text{cost}_e(\beta) = \beta$ for each edge e. Their approach can be used to solve FlexDraw, as edges with higher flexibility can be modeled by a path of inflexible edges. Biedl and Kant [3] show that every 4-planar graph (except for the octahedron) admits an orthogonal drawing with at most two bends per edge. Thus, FlexDraw is trivial if the flexibility of every edge is at least 2. Bläsius et al. [2,5] tackle the NP-hard problems FlexDraw and OptimalFlexDraw by not counting the first bend on every edge. They give a polynomial time algorithm solving FlexDraw if the flexibility of every edge is at least 1 [2]. Moreover, they show how to efficiently solve OptimalFlexDraw if the cost function of every edge is convex and allows the first bend for free [5].

When restricting the allowed drawings to those with a specific planar embedding, the problem OptimalFlexDraw becomes significantly easier. Tamassia [6] shows how to find a drawing with as few bends as possible by computing a flow in a planar flow network. This flow network directly extends to a solution of OptimalFlexDraw with fixed planar embedding, if all cost functions are convex. Cornelsen and Karrenbauer [7] recently showed, that this kind of flow network can be solved in $O(n^{3/2})$ time.

Contribution & outline In this work we consider OptimalFlexDraw for instances that may contain inflexible edges, closing the gap between the general NP-hardness result [1] and the polynomial-time algorithms in the absence of inflexible edges [2,5]. After presenting some preliminaries in Section 2, we show in Section 3 that FlexDraw remains NP-hard even for instances with only $O(n^\varepsilon)$ (for any $\varepsilon > 0$) inflexible edges that are distributed evenly over the graph, i.e., they have pairwise distance $\Omega(n^{1-\varepsilon})$. This includes the cases where the inflexible edges are restricted to form very simple structures such as a matching.

On the positive side, we describe a general algorithm that can be used to solve OptimalFlexDraw by solving smaller subproblems (Section 4). This provides a framework for the unified description of bend minimization algorithms which covers both, previous work and results presented in this paper. We use this framework in Section 5 to solve OptimalFlexDraw for series-parallel graphs with non-decreasing cost functions. This extends the algorithm by Di Battista et al. [4] to non-biconnected series-parallel graphs and thus solves one of their open problems. Moreover, we allow a significantly larger set of cost functions (in particular, the cost functions may be non-convex).

In Section 6, we present our main result, which is an FPT-algorithm with running time $O(2^k \cdot n \cdot T_{\text{flow}}(n))$, where k is the number of inflexible edges incident to degree-4 vertices, and $T_{\text{flow}}(n)$ is the time necessary to compute a maximum flow in a planar flow network of size n with multiple sources and sinks. Note that we can allow an arbitrary number of edges whose endpoints both have degree at most 3 to be inflexible without increasing the running time. Thus, our algorithm can also test the existence of a 0-bend drawing (all edges are inflexible) in FPT-time with respect to the number of degree-4 nodes. This partially solves another open problem of Di Battista et al. [4]. We conclude with open questions in Section 7.

2. Preliminaries

2.1. Connectivity & the composition of graphs

A graph G is *connected* if there exists a path between every pair of vertices. A *separating k-set* S is a subset of vertices of G such that $G - S$ is not connected. Separating 1-sets are called *cutvertices* and separating 2-sets *separation pairs*. A connected graph without cutvertices is *biconnected* and a biconnected graph without separation pairs is *triconnected*. The *blocks* of a connected graph are its maximal (with respect to inclusion) biconnected subgraphs.

An *st-graph* G is a graph with two designated vertices s and t such that $G + st$ is biconnected and planar. The vertices s and t are called the *poles* of G. Let G_1 and G_2 be two st-graphs with poles s_1, t_1 and s_2, t_2, respectively. The *series composition* G of G_1 and G_2 is the union of G_1 and G_2 where t_1 is identified with s_2. Clearly, G is again an st-graph with the poles s_1 and t_2. In the *parallel composition* G of G_1 and G_2 the vertices s_1 and s_2 and the vertices t_1 and t_2 are identified with each other and form the poles of G. An st-graph is *series-parallel*, if it is a single edge or the series or parallel composition of two series-parallel graphs.

To be able to compose all st-graphs, we need a third composition. Let $G_1, \dots, G_\ell$ be a set of st-graphs with poles s_i and t_i associated with G_i. Moreover, let H be an st-graph with poles s and t such that $H + st$ is triconnected and let $e_1, \dots, e_\ell$ be the edges of H. Then the *rigid composition* G with respect to the so-called *skeleton* H is obtained by replacing each edge e_i of H by the graph G_i, identifying the endpoints of e_i with the poles of G_i. It follows from the theory of SPQR-trees that every st-graph is either a single edge or the series, parallel or rigid composition of st-graphs [8,9].

2.2. SPQR-tree

The SPQR-tree $\mathcal{T}$ of a biconnected st-graph G containing the edge st is a rooted tree encoding series, parallel and rigid compositions of st-graphs that result in the graph G [8,9]. The leaves of $\mathcal{T}$ are *Q-nodes* representing the edges of G and thus the st-graphs we start with. The root of $\mathcal{T}$ is also a Q-node, representing the special edge st. Each inner node is either an *S-node*, representing one or more series compositions of its children, a *P-node*, representing one or more parallel compositions of its children, or an *R-node*, representing a rigid composition of its children.

Recall that the rigid composition is performed with respect to a skeleton. For an R-node μ, let H be the skeleton of the corresponding rigid composition with poles s_μ and t_μ. We call $H + s_\mu t_\mu$ the *skeleton* of the μ and denote it by $\text{skel}(\mu)$.