UNIVERSITY OF CALIFORNIA,
IRVINE

Optimization Problems in Directed Graph Visualization

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Juan José Besa Vial

Dissertation Committee:
Distinguished Professor Michael T. Goodrich, Chair
Professor Amelia Regan
Chancellor's Professor David Eppstein

2019

DEDICATION

To my wife Teresa; who left her country, built a wonderful home, pushed me to succeed, and gave me the three greatest joys of my life.

"Cause it's getting better..."

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

First, I would like to thank my advisor Michael Goodrich for his mentorship. I would also like to thank David Eppstein and Amelia Regan for serving on my defense committee, and Sandy Irani and Jay Jayakrishnan for serving on my advancement committee.

I would like to thank my coauthors: Timothy Johnson, Giordano Da Lozzo, William Devanny, David Eppstein, Michael T. Goodrich, and Martha Osegueda. I would also like to thank Michael Adams for his assistance and guidance in the DARPA STAC project.

I would like to thank my fellow graduate students in the Center for Algorithms and Theory of Computation. Our community has been a source of entertainment, inspiration, collaboration, and valuable feedback. In particular, I would like to thank Haleh, Sid, and Nil for the hours of conversation and help.

I would like to thank Aniket Shivam for countless lunches and discussions, Neftali Watkinson for always lending a helping hand, and Alex Veidenbaum for accepting me in the beginning and helping me find my way.

I would like to thank the Donald Bren School of Information and Computer Science for their funding support. I would like to thank DARPA for supporting this work under agreement no. AFRL FA8750-15-2-0092. This work was also supported in part by the U.S. National Science Foundation under grants 1526631, and 1815073. The views expressed in this dissertation are those of myself and my coauthors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. I would like to thank funding by CONICYT Doctorado Becas Chile 2015 - 72160069

CURRICULUM VITAE

Juan José Besa Vial

EDUCATION

Doctor of Philosophy in Computer Science	**2019**
University of California, Irvine	*Irvine, California*
Masters in Science of Engineering	**2011**
Pontificia Universidad Católica de Chile	*Santiago, Chile*
Bachelor in Computer Science Engineering	**2011**
Pontificia Universidad Católica de Chile	*Santiago, Chile*

TEACHING EXPERIENCE

Lecturer	**2012–2013**
Pontificia Universidad Católica de Chile	*Santiago, Chile*

PUBLICATIONS

Computing k-Modal Embedding of Planar Digraphs **September 2019**
European Symposium on Algorithms

Optimally Sorting Evolving Data **July 2018**
International Colloquium on Automata, Languages and Programming

Quadratic Time Algorithms Appear to be Optimal for Sorting Evolving Data **January 2018**
Algorithm Engineering and Experiments

Scheduling autonomous vehicle platoons through an unregulated intersection **September 2016**
Workshop on Algorithmic Approaches for Transportation Modeling, Optimization and Systems

A Concurrent Red-Black Tree **April 2013**
Journal of Parallel and Distributed Computing

ABSTRACT OF THE DISSERTATION

Optimization Problems in Directed Graph Visualization

By

Juan José Besa Vial

Doctor of Philosophy in Computer Science

University of California, Irvine, 2019

Distinguished Professor Michael T. Goodrich, Chair

Drawing digraphs presents unique challenges that do not occur when drawing undirected graphs. Many digraphs tend to represent transitive relationships; that is they have a flow, and show a progression. When drawing digraphs the drawing style used must attempt to transmit this structural characteristic. In this dissertation, we study several optimization problems that are unique to drawing digraphs. First, we study the complexity the k-Modality problem of planar digraphs. Second, we turn to the practical task of drawing minimum width phylogenetic trees, which are used in the study of evolutionary relationships. Finally, we study a classical graph drawing problem, the One-Sided Crossing Minimization problem, in the novel evolving data setting.

An embedding is k-modal if every vertex is incident to at most k pairs of consecutive edges with opposite orientations. We study the k-Modality problem, which asks for the existence of a k-modal embedding of a planar digraph. This combinatorial problem is at the very core of a variety of constrained embedding questions for planar digraphs and flat clustered networks. We characterize the complexity of finding minimum modality embeddings, relate it to other graph drawing problems, both directed and undirected, and present fixed-parameter tractable algorithms for some important families.

We then study the problem of drawing small width phylogenetic trees. Phylogenetic trees are rooted trees that describe the evolutionary relationships derived from a common ancestor. In these the vertical distance represents the amount of time that passes; thus, the length of the edges is fixed. The PHYLOGENETIC TREE DRAWING problem asks for a minimum-width orthogonal upward drawing of a phylogenetic tree. We show that finding such a drawing is NP-hard for binary trees with unconstrained combinatorial order and provide a linear-time algorithm for ordered trees. We also study several heuristic algorithms for the unconstrained case and show their effectiveness through experimentation.

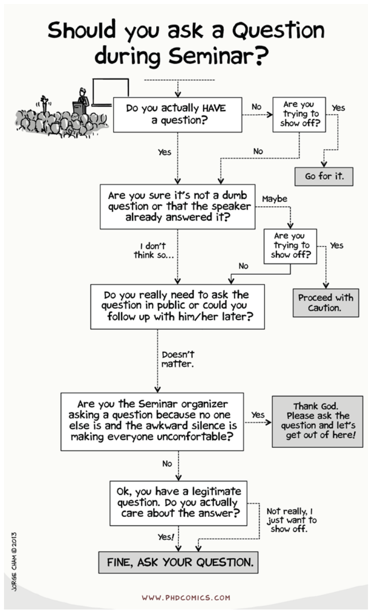
Finally, we study a restricted version of the ONE-SIDED CROSSING MINIMIZATION problem in the evolving data setting. Algorithms that solve this problem attempt to minimize the number of crossings between two adjacent layers when drawing layered graphs. We reduce the problem to that of sorting a list in the evolving data setting and study it from this viewpoint. In this model, a sorting algorithm maintains an approximation to the sorted order of a list of data items while simultaneously, with each comparison made by the algorithm, an adversary randomly swaps the order of adjacent items in the true sorted order. The experiments we perform in this dissertation provide empirical evidence that some quadratic-time algorithms such as insertion sort and bubble sort are asymptotically optimal for any constant rate of random swaps. In fact, these algorithms perform as well as or better than algorithms such as quicksort that are more efficient in the traditional algorithm analysis model.

Chapter 1

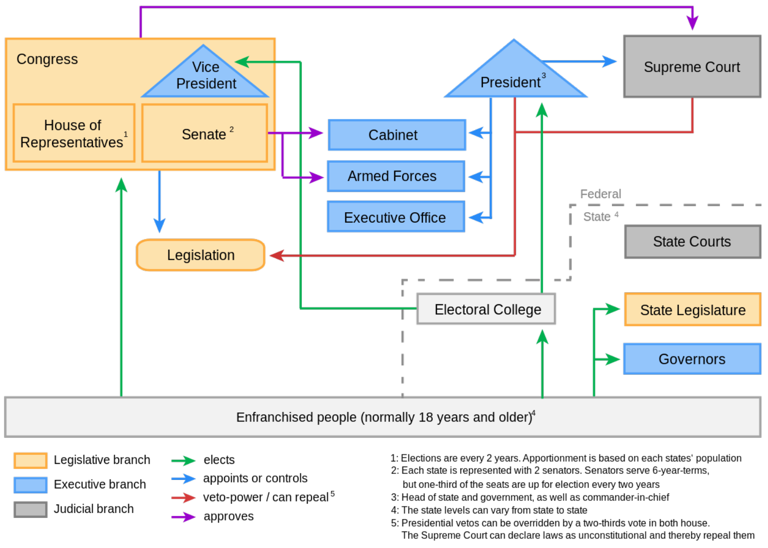
Introduction

A well-drawn diagram gives insight beyond the surface information to the hidden structure beneath. By engaging our visual cortex, it enables us to see patterns and it improves recall [79] compared to textual representation. Diagrams permit the visual exploration of the relationships between data. As we are immersed in the era of Big Data, there is an increasing need for techniques to visualize large and complex data sets. Such data sets cannot be drawn by hand, so they require computers to decide where to place the objects and where to draw the lines. Graph drawing is the research field dedicated to the automated generation of diagrams. More broadly it "includes all aspects of visualizing structural relations between objects" [70].

Formally, an (undirected) *graph* $G = (V, E)$ consists of a nonempty set V of objects called vertices together with a set E of 2-element subsets of V called edges. A *digraph* is a graph whose edges are ordered pairs. Drawings are an intuitive approach to understanding graphs, for example, any textbook on graph theory will contain drawings of graphs in the first few pages. A single graph can have many drawings emphasizing different structural aspects, for example symmetry or connectivity, or satisfying different constraints. Graph


Should you ask a Question during Seminar?
Do you actually HAVE a question?
No
Are you trying to show off?
Yes
Go for it.
Yes
No
Are you sure it's not a dumb question or that the speaker already answered it?
Maybe
Are you trying to show off?
Yes
Proceed with Caution.
I don't think so...
No
Do you really need to ask the question in public or could you follow up with him/her later?
Doesn't matter.
Are you the Seminar organizer asking a question because no one else is and the awkward silence is making everyone uncomfortable?
Yes
Thank God. Please ask the question and let's get out of here!
No
Ok, you have a legitimate question. Do you actually care about the answer?
Not really, I just want to show off.
Yes!
FINE, ASK YOUR QUESTION.
JORGE CHAM © 2013
WWW.PHDCOMICS.COM

(a)


Congress
Vice President
House of Representatives 1
Senate 2
President 3
Supreme Court
Cabinet
Armed Forces
Executive Office
Legislation
Federal
State 4
State Courts
Electoral College
State Legislature
Governors
Enfranchised people (normally 18 years and older) 4
Legislative branch
Executive branch
Judicial branch
elects
appoints or controls
veto-power / can repeal 5
approves
1: Elections are every 2 years. Apportionment is based on each states' population
2: Each state is represented with 2 senators. Senators serve 6-year-terms, but one-third of the seats are up for election every two years
3: Head of state and government, as well as commander-in-chief
4: The state levels can vary from state to state
5: Presidential vetos can be overridden by a two-thirds vote in both house. The Supreme Court can declare laws as unconstitutional and thereby repeal them

(b)

Figure 1.1: (a) A useful flow chart[1]. (b) An orthogonal clustered digraph diagram of US political system [1].

drawing combines aspects of graph theory, computational geometry and (sometimes) aesthetic considerations.

The most common style of graph drawing is a node-link diagram where vertices are represented as points and each edge is drawn as a line between two points, but this is not the only possible representation. Vertices can be represented by many shapes such as rectangles (Fig. 1.1), images, or a mixture of styles, while edges can be represented as straight lines, polylines, curves, the intersection of objects, and many other ways. Even the notion of graph can be expanded to include clusters or hierarchies of clusters of vertices, which can also be captured in the drawings (Fig. 1.1b). .

Creating drawings of digraphs presents unique problems that do not occur when creating drawings of undirected graphs. An edge $e = (u, v)$ in a digraph indicates that the edge is *oriented* from vertex u to vertex v. Drawings must indicate the orientation of the edges

[1]With permission from "Piled Higher and Deeper" by Jorge Cham www.phdcomics.com

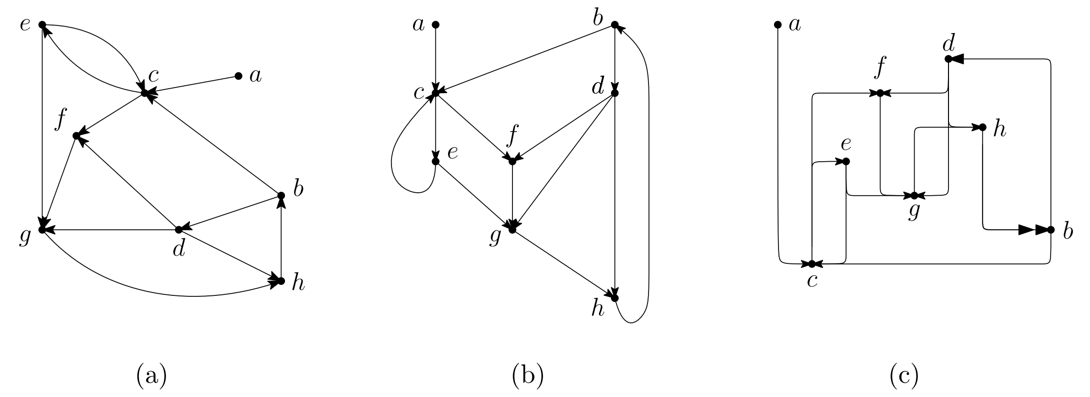


Figure 1.2: Illustration of a digraph in 3 styles (a) Node-link drawing. (b) Layered graph drawing. (c) L-drawing.

clearly. The most common way of doing this is using arrowheads, but these are not always clear and easy to follow due to occlusion problems and visual clutter [62]. An alternative is to use a convention that implicitly indicates the orientation. Upward planar drawings (such as those in Chapter 3) use the convention that every edge is oriented from a lower vertex to a higher vertex. This is the natural style to draw rooted trees, where traditionally the root is the highest node in the drawing and every child node lies below its parent. Unfortunately, the digraphs that can be drawn in such a way are limited. We discuss an important property they must have in Chapter 2. A more general approach is layered graph drawings where vertices are drawn in horizontal rows, or layers, and edges mostly flow in the same direction. Such drawings are effective at displaying more complex digraphs, for example remaining useful in graphs with a limited quantity of cycles. Other styles include L-drawings [7], confluent drawings [43], elastic hierarchies [105], and overloaded orthogonal drawings [75], see Fig. 1.2.

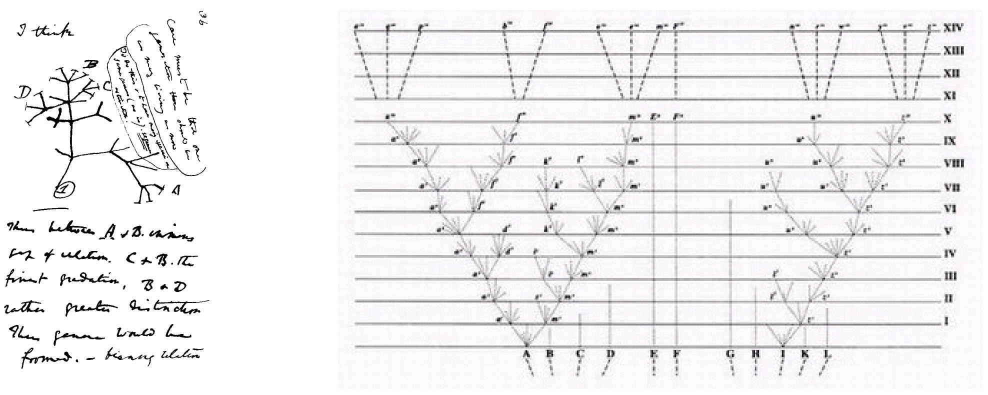
Once a representation has been decided then the question remains about what distinguishes a good drawing from a bad one. Due to the multiplicity of graphs, structures, and purposes, there can be no single optimization criterion. Over time, the graph drawing community has focused on different criteria which capture characteristics which tend to produce high quality drawings in many different settings. In fact, a good drawing typically fulfills several of these optimization criteria. One class of graphs that is particularly relevant is planar graphs. A *planar graph* is a graph that can be drawn in the plane without edge crossings. Many times planarity is not a goal but a constraint that must be satisfied before considering other optimization goals. Such is the case in Chapters 2 and 3. Other criteria are minimizing the number of edge crossings, area, total edge length, angle resolution (both at vertices and at crossings), and number of bends, among many others.

1.1 K-Modality

Given a planar digraph G and a positive even integer k, a drawing of G in the plane is *k-modal* if every vertex of G is incident to at most k pairs of consecutive edges with opposite orientations, i.e., the incoming and the outgoing edges at each vertex are grouped by the drawing into at most k sets of consecutive edges with the same orientation. A necessary, but not sufficient, condition for upward-planar drawings and level-planar drawings is for the graph to be 2-modal. In an L-drawing (Fig. 1.2c) vertices are placed on a $n \times n$ grid so that each vertex is assigned a unique x-coordinate and a unique y-coordinate, and each directed edge (uv) is represented as a 1-bend orthogonal polyline composed of a vertical segment incident to u and a horizontal segment incident to v. It is known that 4-modality is a necessary condition for L-drawings [35].

In Chapter 2 we study the k-Modality problem, which asks for the existence of a k-modal embedding of a planar digraph. This combinatorial problem is at the very core of a variety of constrained embedding questions for planar digraphs and flat clustered networks.

First, since the 2-Modality problem can be easily solved in linear time, we consider the general k-Modality problem for any value of $k > 2$ and show that the problem is NP-complete for planar digraphs of maximum degree $\Delta \geq k+3$. We then relate its computational


I think
A
B
C
D
Then between A & B immense gap of relation. C & B the finest gradation, B & D rather greater distinction. Thus genera would be formed. — bearing relation
XIV
XIII
XII
XI
X
IX
VIII
VII
VI
V
IV
III
II
I
A B C D E F G H I K L

(a) (b)

Figure 1.3: (a) Charles Darwin's 1837 sketch of an evolutionary tree. (b) The final drawing he used in 1859 in "On the Origin of Species".

complexity to that of two notions of planarity for flat clustered networks. On the positive side, we provide a simple algorithm that runs in $f(k)O(n)$ time for partial 2-trees of arbitrary degree, whose running time is exponential in k and linear in the input size.

Second, we focus our attention on $k = 4$. On the algorithmic side, we show a complexity dichotomy for the k-Modality problem with respect to Δ, by providing a linear-time algorithm for planar digraphs with $\Delta \leq 6$.

1.2 Phylogenetic Trees

A *phylogenetic tree* is a rooted tree that describes the relationships among evolutionary lineages(Fig. 1.3b). The root is the common ancestor of all the species in the tree. Each internal node is a speciation event, an event that produces two or more lineages. Such an event is for example the rapid speciation of the Faroe Island house mouse, *Mus musculus faeroensis*, which was introduced to the island by man less than 300 years ago [95].

There are many techniques to generate phylogenetic trees based upon similarities and differences in physical or genetic characteristics [49]. In some phylogenetic tree drawings the vertical distance between two nodes represents the time between the two events: such trees are also called *clock trees*. For this reason, they are well suited to be drawn as orthogonal upward-planar trees. There is extensive research on optimization problems related to planar upward tree drawings without the edge-length constraints. Likewise, for phylogenetic trees, there are many software applications for drawing them, but they either do not respect the edge lengths or do not generate orthogonal drawings. We are not familiar with any previous work on characterizing the complexity of the minimum-width orthogonal phylogenetic tree drawing problem.

In Chapter 3 we show that finding such drawings is NP-hard even for binary trees if the ordering of the children of each node is unconstrained. On the other hand, if the order is fixed, we provide a linear-time algorithm for finding minimum width drawings. We also study several heuristic algorithms for the unconstrained case and show their effectiveness through experimentation.

1.3 Layered Drawings

In many cases directed graphs represent a hierarchy, or near hierarchy, and we want a drawing that transmits this structure. Some notable applications are project management diagrams, software program call graphs, and file systems. The most popular method of drawing these digraphs is the *Sugiyama framework* which partitions the vertices of the digraph into discrete layers. The layer of each vertex is chosen so that (most of) the edges of the digraph “flow” in a uniform direction. One of the steps in this framework is the *vertex ordering* step. Consider two layers of vertices drawn in two parallel lines, with edges only going from the top layer to the bottom layer. The vertex ordering step attempts to minimize the number of edge crossings by reordering the vertices of each layer.

In Chapter 4 we study the vertex ordering step in the evolving data setting. In a traditional setting an algorithm takes an input, runs for some amount of time, and produces an output. The evolving data setting asks the question what happens if the data is changing at a speed that is similar (or faster) than the computation speed, i.e.,what happens if the algorithm can't keep up. The goal of the algorithm in this setting is to efficiently maintain an output instance that is "close" to the true output.

We experimentally study how to reduce the crossings between two adjacent layers in a layered drawing in this setting. Our setup is as follows: we consider two adjacent layers where each element in the top layer has a one-to-one correspondence to an element in the bottom layer. The bottom layer mutates its order by performing random swaps on adjacent elements, while the algorithm attempts to order the top layer so as to minimize edge crossings. Clearly, at any single moment there is an order of the top layer that produces a drawing with no crossings, when both are equally "sorted", but in practice this is never achieved. We show that this constrained version can be reduced to maintaining a sorted list where the rank of the elements is evolving and this is exactly the approach we take.

Previous work on sorting in the evolving data setting studies only two versions of quicksort, and has a gap between the lower bound of $\Omega(n)$ and the (then known) best upper bound of $O(n \log \log n)$. The experiments we present in Chapter 4 provide empirical evidence that some quadratic-time algorithms such as insertion sort and bubble sort are asymptotically optimal for any constant rate of random swaps per comparison. In fact, these algorithms perform as well as or better than algorithms such as quicksort that are more efficient in the traditional algorithm analysis model. In a later paper Besa *et al.* [18] confirmed that in fact insertion sort can be optimal in this setting.

Chapter 2

Computing k-Modal Embeddings of Planar Digraphs

2.1 Introduction

Computing k-modal embeddings of planar digraphs, for some positive even integer k called *modality*, is an important algorithmic task at the basis of several types of graph visualizations. In *2-modal embeddings*, also called *bimodal embeddings*, the outgoing and the incoming edges

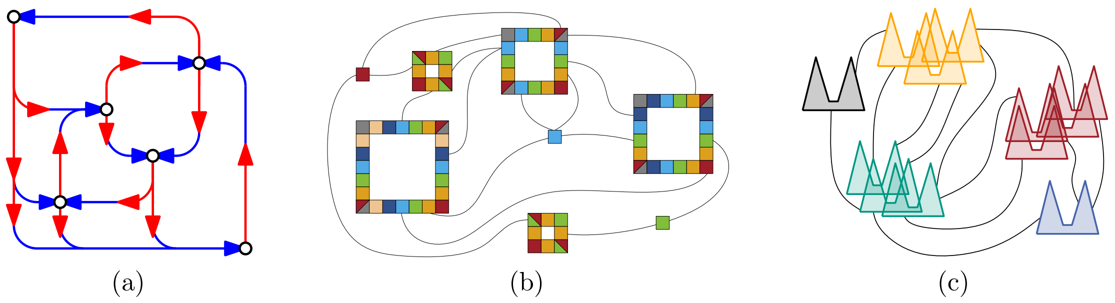


Figure 2.1: (a) A planar L-drawing, which determines a 4-modal embedding. (b) A planar NodeTrix representation. (c) A planar intersection-link representation using comb-shaped polygons.

at each vertex form two disjoint sequences. Bimodal embeddings are ubiquitous in Graph Drawing. For instance, level planar drawings [39, 68] and upward-planar drawings [17, 53]—two of the most deeply-studied graph drawing standards—determine bimodal embeddings. *4-modal embeddings*, where the outgoing and the incoming edges at each vertex form up to four disjoint sequences with alternating orientations, arise in the context of planar L-drawings of digraphs. In an *L-drawing* of an n-vertex digraph, introduced by Angelini *et al.* [7], vertices are placed on the $n \times n$ grid so that each vertex is assigned a unique x-coordinate and a unique y-coordinate and each edge uv (directed from u to v) is represented as a 1-bend orthogonal polyline composed of a vertical segment incident to u and of a horizontal segment incident to v. Recently, Chaplick *et al.* [35] addressed the question of deciding the existence of *planar L-drawings*, i.e., L-drawings whose edges might possibly overlap but do not cross and observe that the existence of a 4-modal embedding is a necessary condition for a digraph to admit such a representation (Fig. 2.1a).

To the best of our knowledge, no further relationships have been explicitly pointed out in the literature between modal embeddings and notable drawing models for modality values greater than four, yet they do exist. Da Lozzo *et al.* [37] and Di Giacomo *et al.* [54] study the planarity of *NodeTrix representations* of flat clustered networks, a hybrid representational model introduced by Henry, Fekete, and McGuffin [61], where clusters and intra-cluster edges are represented as adjacency-matrices, with rows and columns for the vertices of each cluster, and inter-cluster edges are Jordan arcs connecting different matrices (Fig. 2.1b). For clusters containing only two vertices, it is possible to show that the problem of computing planar NodeTrix representations coincides with the one of testing whether a special digraph, called the *canonical digraph*, associated to the network admits a 6-modal embedding. For higher values of modality, k-modal embeddings occur in the context of Intersection-Link representations of flat clustered networks. In an *intersection-link representation* [9, 11], vertices are represented as translates of the same polygon, intra-cluster edges are represented via intersections between the polygons corresponding to their endpoints, and inter-cluster edges—similarly to NodeTrix representations—are Jordan arcs connecting the polygons corresponding to their endpoints. For any modality $k \geq 2$, it can be shown that testing the existence of a k-modal embedding of the canonical digraph of a flat clustered network with clusters of size two is equivalent to testing the existence of an intersection-link representation in which the curves representing inter-cluster edges do not intersect, when vertices are drawn as comb-shaped polygons (Fig. 2.1c).

Related Work. It is common knowledge that the existence of bimodal embeddings can be tested in linear time: Split each vertex v that has both incoming and outgoing edges into two vertices v_{in} and v_{out}, assign the incoming edges to v_{in} and the outgoing edges to v_{out}, connect v_{in} and v_{out} with an edge, and test the resulting (undirected) graph for planarity using any of the linear-time planarity-testing algorithms [27, 64]. Despite this, most of the planarity variants requiring bimodality are NP-complete; for instance, upward planarity [53], windrose planarity [12], partial-level planarity [29], clustered-level planarity and T-level planarity [10, 73], ordered-level planarity and bi-monotonicity [73]. In this scenario, a notable exception is represented by the classic level planarity problem, which can be solved in linear time [68], and its generalizations on the standing cylinder [13], rolling cylinder and the torus [8]. Although the existence of a bimodal embedding is easy to test, Binucci, Didimo, and Giordano [23] prove that the related problem of finding the maximum bimodal subgraph of an embedded planar digraph is an NP-hard problem. Moreover, Binucci, Didimo, and Patrignani [24] show that, given a *mixed planar graph*, i.e., a planar graph whose edge set is partitioned into a set of directed edges and a set of undirected edges, orienting the undirected edges in such a way that the whole graph admits a bimodal embedding is an NP-complete problem. On the other hand, the question regarding the computational complexity of constructing k-modal embedding for $k \geq 4$ has not been addressed, although the related problem of testing the existence of planar L-drawings has been recently proved NP-complete [35].

Our results. We study the complexity of the k-MODALITY problem, which asks for the existence of k-modal embeddings of planar digraphs—with an emphasis on $k = 4$. Our results are as follows:

- We demonstrate a complexity dichotomy for the 4-Modality problem with respect to the maximum degree Δ of the input digraph. Namely, we show NP-completeness when $\Delta \geq 7$ (Theorem 2.12) and give a linear-time testing algorithm for $\Delta \leq 6$ (Theorem 2.8). Further, we extend the hardness result to any modality value larger than or equal to 4, by proving that the k-Modality problem is NP-complete for $k \geq 4$ when $\Delta \geq k + 3$.

- We provide an FPT-algorithm for k-Modality that runs in $f(k)O(n)$ time for the class of directed partial 2-trees (Theorem 2.7), which includes series-parallel and outerplanar digraphs.

- In Section 2.3, we relate k-modal embeddings with hybrid representations of flat clustered graphs, and exploit this connection to give new complexity results (Theorems 2.2 and 2.4) and algorithms (Theorems 2.1 and 2.3) for these types of representations. In particular, our NP-hardness results allow us to answer two open questions. Namely, we settle in the strongest possible way an open question, posed by Di Giacomo *et al.* [54, Open Problem (i)], about the complexity of computing planar NodeTrix representations of flat clustered graphs with clusters of size smaller than 5. Also, we address a research question by Angelini *et al.* [9, Open Problem (2)] about the representational power of intersection-link representations based on geometric objects that give rise to complex combinatorial structures, and solve it when the considered geometric objects are k-combs.

- Finally, in Section 2.10, we show that not every outerplanar digraph admits a bimodal embedding, whereas any outerplanar (multi-)digraph admits a 4-modal embedding.

The algorithms presented in this chapter employ the SPQ- and SPQR-tree data structures to succinctly represent the exponentially-many embeddings of series-parallel and biconnected planar digraphs, respectively, and can be easily modified to output an embedding of the input digraph in the same time bound. In particular, our positive result for $\Delta \leq 6$ is based on a set of simple reduction rules that exploit the structure of the rigid components of bounded-degree planar digraphs. These rules allow us to tackle the algorithmic core of the problem, by enabling a final reduction step to special instances of Not-All-Equal SAT (NAESAT), previously studied by Porschen *et al.* [86]. NAESAT is a variant of the boolean satisfiability problem SAT, which asks for a truth assignment such that evey clause contains both a true and a false literal. If such a truth assignment exists then the instance is NAE-satisfiable. We prove that the special instance we construct is always NAE-satisfiable in Section 2.7.[1]

2.2 Definitions and Preliminaries

In this section we give preliminaries and definitions that will be useful throughout.

Planar digraphs and embeddings. Let $G = (V, E)$ be a digraph. We also denote the sets V and E by $V(G)$ and $E(G)$, respectively. The *underlying graph* of G is the undirected graph obtained from G by disregarding edge directions. Let v be a vertex, we denote by $E(v)$ the set of edges of G incident to v and by $\deg(v) = |E(v)|$ the *degree* of v. If uv is an edge, then vertex v is a *successor* of u in G and vertex u is a *predecessor* of v in G. For an edge $e = uv$ directed from u to v and an end-point $x \in \{u, v\}$ of e, we define the *orientation* $\sigma(e, x)$ of e *at* x as $\sigma(e, x) = ⫰$, if $x = u$, and $\sigma(e, x) = ⫯$, if $x = v$, and we say that uv is *outgoing from u* and *incoming at v*. Also, we define $\overline{\sigma}(e, x) = ⫯$, if $\sigma(e, x) = ⫰$, and $\overline{\sigma}(e, x) = ⫰$, if $\sigma(e, x) = ⫯$.

[1] In "*Stefan Porschen, Bert Randerath, Ewald Speckenmeyer: Linear Time Algorithms for Some Not-All-Equal Satisfiability Problems. SAT 2003: 172-187*" [86], the authors state in the abstract "First we show that a NAESAT model (if existing) can be computed in linear time for formulas in which each variable occurs at most twice.". We give a strengthening of this result by showing that the only negative formulas with the above properties are those whose variable-clause graph contains components isomorphic to a simple cycle. Then, we provide a recursive linear-time algorithm for computing a NAE-truth assignment for formulas with those properties, when one exists. Our algorithm is also considerably simpler than the one presented in [86].

A digraph is *planar* if it admits a drawing in the plane without edge crossings. A *combinatorial embedding* (for short, *embedding*) is an equivalence class of planar drawings, where two drawings of a (di)graph are *equivalent* if they determine the same circular ordering of the edges around each vertex v, called *rotation* at v, and the same relative positions of connected components to one another. A planar drawing partitions the plane into topologically connected regions, called *faces*. The bounded faces are the *inner faces*, while the unbounded face is the *outer face*. A combinatorial embedding together with a choice for the outer face defines a *planar embedding*.

Out definition does not allow the existence of multi-edges, although alternative definitions may allow them. A multi-edge is a set of edges with the same end-points and the same orientation; thus, edges uv and vu do not form a multi-edge. All the digraphs considered in this chapter are *simple*, i.e., their edge set does not contain neither multi-edges nor loops, planar, and connected. Observe that, requiring the digraph to be simple is not a loss of generality. In fact, it is possible to modify a digraph G, by removing loops and replacing multi-edges with a single directed edge, while preserving the possibility of having a k-modal embedding, for any $k > 0$, if any such an embedding of G exists.

Let $\mathcal{E}$ be a combinatorial embedding (planar embedding) of a planar digraph G and let H be a subgraph of G, that is, $H \subseteq G$. Also, let $\mathcal{E}_H$ be the embedding of H obtained by restricting $\mathcal{E}$ to the vertices and edges of H. We say that embedding $\mathcal{E}$ *induces* embedding $\mathcal{E}_H$.

Connectivity. A graph $G = (V, E)$ is *connected*, if there is a path between any two vertices, and it is *disconnected*, otherwise. A digraph is *connected* or *disconnected*, if its underlying graph is connected or disconnected. A *cutvertex* is a vertex whose removal disconnects the graph. A separating pair $\{u, v\}$ is a pair of vertices whose removal disconnects the graph. A *split pair* is a separation pair or a pair of adjacent vertices. A connected digraph is *biconnected* if it does not have a cutvertex and a biconnected graph is *triconnected* if it does not have a separating pair. The *blocks* of a connected digraphs are its maximal biconnected subgraphs. A block is *trivial*, if it consists of a single-edge, and *non-trivial*, otherwise. Clearly, a non-trivial block contains a cycle. We remark that, by the above definition, the connectivity of a digraph as the same as the connectivity of its underlying graph.

BC-trees. To handle the decomposition of a connected digraph into its biconnected components, we use *block-cutvertex* trees—usually referred to as *BC-trees*, a data structure introduced by Harary and Prins [60]. The BC-tree T of a connected digraph $G = (V, E)$ is a tree with a B-node for each block of G and a C-node for each cutvertex of G. Edges in $\mathcal{T}$ connect each B-node β to the C-nodes associated with the cutvertices in the block of β.

SPQR-trees. To handle the decomposition of a biconnected digraph into its triconnected components we use SQPR-trees. Triconnected components, also known as *Tutte components*, was independently considered by Tutte [99], Hopcroft and Tarjan [63], and Mac Lane [78]. Later such a decomposition was named by Di Battista and Tamassia [40] as *SPQR-trees*. Although not in their current formalization, SPQR-trees were already exploited by Bienstock and Monma [21, 22].

A graph is st-*biconnectible* if adding the edge (s, t) yields a biconnected graph. Let G be an st-biconnectible graph. A *separation pair* of G is a pair of vertices whose removal disconnects the graph. A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *maximal split component* of G with respect to a split pair $\{u, v\}$ (or, simply, a maximal split component of $\{u, v\}$) is either an edge (u, v) or a maximal subgraph G' of G such that G' contains u and v, and $\{u, v\}$ is not a split pair of G'. A vertex $w \neq u, v$ belongs to exactly one maximal split component of $\{u, v\}$. We call *split component* of $\{u, v\}$ the union of any number of maximal split components of $\{u, v\}$.

In this chapter, we will assume that a SPQR-tree of a graph G is rooted at one edge of G, called its *reference edge*.

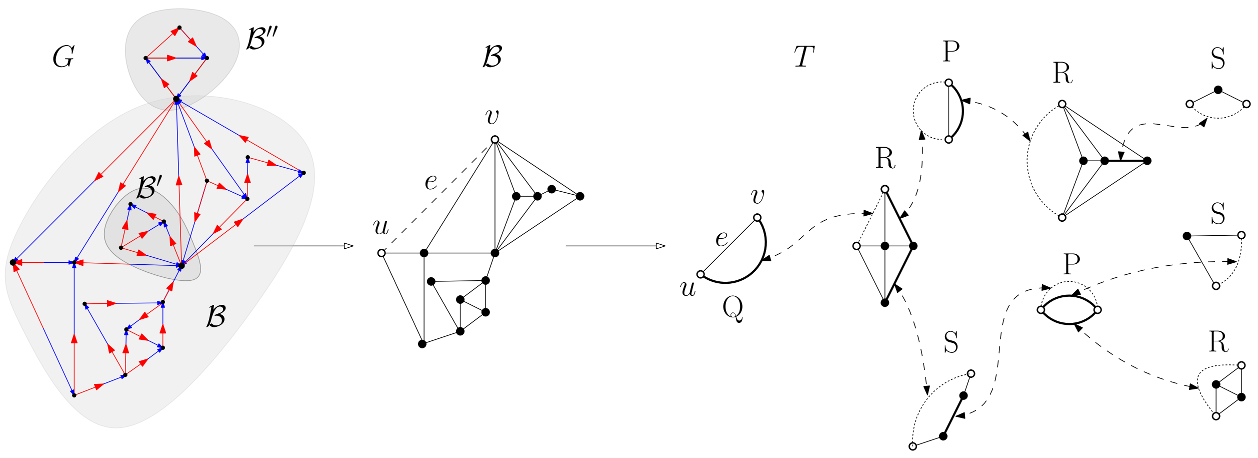


Figure 2.2: (left) A 4-modal embedding of a simply-connected planar digraph G; the part of each edge that is incoming at (outgoing from) a vertex is drawn blue (red). The three blocks $\mathcal{B}$, $\mathcal{B}'$, and $\mathcal{B}''$ of G are enclosed in gray shaded regions. (right) SPQR $\mathcal{T}$ of block $\mathcal{B}$ (considered as undirected) rooted at the edge $e = uv$. The extended skeletons of all non-leaf nodes of $\mathcal{T}$ are shown; virtual edges corresponding to edges of G are thin, whereas virtual edges corresponding to S-, P-, and R-nodes are thick. Dashed arrowed curves connect the (dotted) parent edge in the skeleton of a child node with the virtual edge representing the child node in the skeleton of its parent.

The rooted SPQR-tree $\mathcal{T}$ of a biconnected graph G, with respect to its reference edge e, describes a recursive decomposition of G induced by its split pairs. The nodes of $\mathcal{T}$ are of four types: S, P, Q, and R. Their connections are called *arcs*, in order to distinguish them from the edges of G.

Each node μ of $\mathcal{T}$ has an associated *st*-biconnectible multigraph, called the *skeleton* of μ and denoted by skel(μ). Skeleton skel(μ) shows how the children of μ, represented by "virtual edges", are arranged into μ. The virtual edge in skel(μ) associated with a child node ν, is called the *virtual edge of ν in* skel(μ).

The subgraph of G that can be obtained in the following manner is denoted the *pertinent graph* of μ. For each virtual edge e_i of skel(μ), recursively replace e_i with the skeleton skel(μ_i) of its corresponding child μ_i.

Let G be a biconnected graph. Given the reference edge $e = (u', v') \in E$, the SPQR-tree $\mathcal{T}$ is recursively defined as follows. At each step, a split component G^*, a pair of vertices $\{u, v\}$, and a node ν in $\mathcal{T}$ are given. A node μ corresponding to G^* is introduced in $\mathcal{T}$ and attached to its parent ν. Vertices u and v are the *poles* of μ and denoted by u_μ and v_μ, respectively. The decomposition possibly recurs on some split components of G^*. At the beginning of the decomposition $G^* = G - \{e\}$, $\{u_\mu, v_\mu\} = \{u', v'\}$, and ν is a Q-node corresponding to e.

Base Case: If G^* consists of exactly one edge between u_μ and v_μ, then μ is a Q-node whose skeleton is G^* itself.

Parallel Case: If G^* is composed of at least two maximal split components $G_1, \ldots, G_k$ ($k \geq 2$) of G with respect to $\{u_\mu, v_\mu\}$, then μ is a P-node. The graph skel(μ) is a multigraph consisting of k virtual edges between u_μ and v_μ, denoted by $e_1, \ldots, e_k$ and corresponding to $G_1, \ldots, G_k$, respectively. The decomposition recurs on $G_1, \ldots, G_k$, with $\{u_\mu, v_\mu\}$ as pair of vertices for every graph, and with μ as parent node.

Series Case: If G^* is composed of exactly one maximal split component of G with respect to $\{u_\mu, v_\mu\}$ and if G^* has cut vertices $c_1, \ldots, c_{k-1}$ ($k \geq 2$), appearing in this order on a path from u_μ to v_μ, then μ is an S-node. Graph skel(μ) is the path $e_1, \ldots, e_k$, where virtual edge e_i connects c_{i-1} with c_i ($i = 2, \ldots, k-1$), e_1 connects u_μ with c_1, and e_k connects c_{k-1} with v_μ. The decomposition recurs on the split components corresponding to each of $e_1, e_2, \ldots, e_{k-1}, e_k$ with μ as parent node, and with $\{u_\mu, c_1\}, \{c_1, c_2\}, \ldots, \{c_{k-2}, c_{k-1}\}, \{c_{k-1}, v_\mu\}$ as pair of vertices, respectively.

Rigid Case: If none of the above cases applies, the purpose of the decomposition step is that of partitioning G^* into the minimum number of split components and recurring on each of them. We need some further definition. Given a maximal split component G' of a split pair $\{s, t\}$ of G^*, a vertex $w \in G'$ *properly belongs* to G' if $w \neq s, t$. Given a split pair $\{s, t\}$ of G^*, a maximal split component G' of $\{s, t\}$ is *internal* if neither u_μ nor v_μ (the poles of G^*) properly belongs to G', *external* otherwise. A *maximal split pair* $\{s, t\}$ of G^* is a split pair of G^* that is not contained in an internal maximal split component of any other split pair $\{s', t'\}$ of G^*. Let $\{u_1, v_1\}, \dots, \{u_k, v_k\}$ be the maximal split pairs of G^* ($k \geq 1$) and, for $i = 1, \dots, k$, let G_i be the union of all the internal maximal split components of $\{u_i, v_i\}$. Observe that each vertex of G^* either properly belongs to exactly one G_i or belongs to some maximal split pair $\{u_i, v_i\}$. The node μ is an R-node. The graph skel(μ) is the graph obtained from G^* by replacing each subgraph G_i with the virtual edge e_i between u_i and v_i. The decomposition recurs on each G_i with μ as parent node and with $\{u_i, v_i\}$ as pair of vertices.

For each node μ of $\mathcal{T}$ with poles u_μ and v_μ, the construction of skel(μ) is completed by adding a virtual edge (u, v) representing the *rest of the graph*, that is, the graph obtained from G by removing all the vertices of pert μ, except for its poles, together with their incident edges.

The skeleton graph equipped with edge $u_\mu v_\mu$, called the *parent edge*, is the *extended skeleton* of μ. Refer to Fig. 2.2(right). Each edge of skel(μ), called *virtual edge*, is associated with a child of μ in $\mathcal{T}$. The skeleton of μ describes how the pertinent graphs of the children of μ have to be "merged" via their poles to obtain pert(μ). The extended skeleton of an S-, P-, R-, and Q-node is a cycle, parallel, triconnected graph, and a multigraph with two edges and two vertices, respectively. It follows that skeleton and pertinent graphs are always biconnected once the parent edge is added.

A digraph G is planar if and only if the skeleton of each R-node in the SPQR-tree of G is planar. Any planar embedding $\mathcal{E}$ of G in which the reference edge e is incident to the outer face induces a *regular embedding* $\mathcal{E}_\mu$ of pert(μ), for each node μ of $\mathcal{T}$, that is, a planar embedding in which the poles of μ are incident to the outer face of $\mathcal{E}_\mu$. Symmetrically, by selecting regular embeddings for the skeletons of the nodes of $\mathcal{T}$, we can construct any embedding of G with edge e on the outer face, where the choices for the embeddings of the skeletons are all and only the (i) flips of the R-nodes and the (ii) permutations of the P-nodes.

The SPQR-tree $\mathcal{T}$ of a graph G with n vertices and m edges has m Q-nodes and $O(n)$ S-, P-, and R-nodes. Also, the total number of vertices of the skeletons stored at the nodes of $\mathcal{T}$ is $O(n)$. Finally, SPQR-trees can be constructed and handled efficiently. Namely, given a biconnected planar graph G, the SPQR-tree $\mathcal{T}$ of G can be computed in linear time [41, 42, 59, 63].

Partial 2-trees and Series-Parallel Digraphs. Recall that we have defined the connectivity of a digraph according to the connectivity of its underlying graph, i.e. ignoring the orientation of its edges. A *2-tree* is a digraph that can be obtained from an edge by repeatedly adding a new vertex connected to two adjacent vertices. Every 2-tree is planar and biconnected. A *partial 2-tree* is a subgraph of a 2-tree. A *series-parallel digraph* is a biconnected partial 2-tree. Equivalently, a *series-parallel digraph* can be defined as a biconnected planar digraph whose SPQR-tree only contains S-, P-, and Q-nodes; this is why we speak of *SPQ-trees* of series-parallel digraphs. A digraph is *outerplanar* if it can be embedded in the plane so that all its vertices are incident to a common face. Outerplanar digraphs are partial 2-trees. The SPQ-tree $\mathcal{T}$ of a biconnected outerplanar digraph rooted at any Q-node has the following property: Each P-node μ of $\mathcal{T}$ has exactly one S-node child (and at least one and at most two Q-node children, corresponding to opposite edges between the poles of μ).

Modality. Let G be a planar digraph and let $\mathcal{E}$ be an embedding of G. A pair of edges e_1, e_2 that appear consecutively in the circular order around a vertex v of G is *alternating* if they do not have the same orientation at v, i.e., they are not both incoming at or both outgoing from v. Also, we say that vertex v is *k-modal*, or that v has *modality k*, or that the *modality of v is k* in $\mathcal{E}$, if there exist exactly k alternating pairs of edges incident to v in $\mathcal{E}$. Clearly, the value k needs to be a non-negative even integer. An embedding of a digraph G is *k-modal*, if each vertex is at most k-modal.

The following observation follows immediately from the fact that in any embedding the number of alternating pairs at a vertex is bounded by its degree and must be even.

Observation 2.1. *In any embedding, the modality of a vertex v is at most $2\lfloor\frac{\deg(v)}{2}\rfloor$.*

By Observation 2.1, vertices of degree at most 5 are at most 4-modal. Therefore, determining the existence of a 4-modal embedding is a non-trivial task only if the digraph contains vertices of degree at least 6. In fact, we will determine a tight border of tractability for the problem, by showing that there exist instances of maximum degree Δ for every $\Delta \geq 7$ that cannot be treated efficiently, unless $P = NP$, while instances of maximum degree 6 can be tested efficiently.

We now define an auxiliary problem, called k-MaxModality (where k is a positive even integer), which will be useful to prove our algorithmic results. We denote the set of non-negative integers by $\mathbb{Z}^*$ and the set of non-negative even integers smaller than or equal to k as $\mathbb{E}_k^+ = \{b : b = 2a, b \leq k, a \in \mathbb{Z}^*\}$. Given a graph G, we call *maximum-modality function* an integer-valued function $m : V(G) \to \mathbb{E}_k^+$. We say that an embedding $\mathcal{E}$ of G *satisfies* m *at a vertex* v, if the modality of v in $\mathcal{E}$ is at most $m(v)$. Also, we say that an embedding $\mathcal{E}$ of G *satisfies* m, if it satisfies m at every vertex of G.

Problem: k-MaxModality

Input: A pair $\langle G, m\rangle$, where G is a digraph and m is a maximum-modality function.

Question: Is there an embedding $\mathcal{E}$ of G that *satisfies* m?

2.3 Implications on Hybrid Representations

A *flat clustered graph* (for short, *c-graph*) is a pair $\mathcal{C} = (G = (V, E), \mathcal{P} = (V_1, V_2, \ldots, V_c))$, where G is a graph and $\mathcal{P}$ is a partition of V into sets V_i, for $i = 1, \ldots, c$, called *clusters*[2]. An

[2]The more general notion of *clustered graph* is obtained by allowing the set of clusters to form a laminar set family, which is better described by a rooted tree $\mathcal{T}$ whose leaves are the vertices of G and whose every

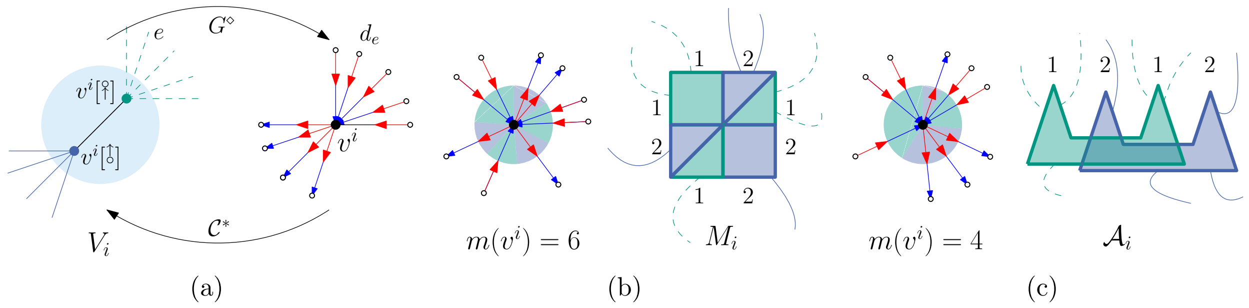


Figure 2.3: (a) Illustrations for the duality between the canonical digraph and the canonical c-graph. Correspondence (b) between 6-modal embeddings and planar NodeTrix representations, and (c) between 4-modal embeddings and clique-planar representations using 2-combs as geometric objects.

edge $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$ is an *intra-cluster edge*, if $i = j$, and is an *inter-cluster edge*, if $i \neq j$. The problem of visualizing such graphs so to effectively convey both the relation information encoded in the set E of edges of G and the hierarchical information given by the partition $\mathcal{P}$ of the clusters has attracted considerable research attention. As crossing-free graph drawings are universally considered more readable [87, 88], this effort has culminated in several notions of planarity for c-graphs. The most celebrated of such notions, introduced by Feng, Cohen, and Eades [50], goes by the name of CLUSTERED PLANARITY and asks for the existence of a *c-planar drawing* of a c-graph, that is, a planar drawing of G together with a representation of each cluster V_i as a region of the plane homeomorphic to a closed disk that contains the drawing of the subgraph of G induced by cluster V_i; additionally, clusters may not intersect each other and edges may cross the boundary of each cluster at most once. Alongside the classical notion of c-planar drawings, new hybrid models for the visualization of flat clustered networks (and corresponding planarity notions) have recently received considerable attention. In a *hybrid representation* of a graph different conventions are used to represent the dense and the sparse portions of the graph [9, 11, 37, 54, 61, 77, 102]. We present important implications of our results on some well-known models for hybrid-representations of c-graphs.

internal node μ represents the cluster containing the leaves of the subtree of $\mathcal{T}$ rooted at μ. However, since a flat clustering is more naturally described by a partition, rather then by a tree, we define c-graphs using $\mathcal{P}$ rather then $\mathcal{T}$.

Let $\mathcal{C}$ be a c-graph whose every cluster forms a clique of size at most 2, that is, each cluster contains at most two vertices connected by an intra-cluster edge. Starting from $\mathcal{C}$ we define an auxiliary digraph $G^{\diamond}$, called the *canonical digraph* for $\mathcal{C}$, as follows. Without loss of generality, assume that, for $i = 1, \dots, c$, each cluster V_i contains two vertices denoted as $v^i[⫯]$ and $v^i[⫰]$. The vertex set of $G^{\diamond}$ contains a vertex v^i, for $i = 1, 2, \dots, c$, and a dummy vertex d_e, for each inter-cluster edge $e \in E$. The edge set of $G^{\diamond}$ contains two directed edges, for each inter-cluster edge $e = (v^i_x, v^j_y) \in E$, with $x, y \in \{⫯, ⫰\}$ and $i \neq j$; namely, $E(G^{\diamond})$ contains (i) either the directed edges $v^i_x d_e$, if $x = ⫰$, or the directed edge $d_e v^i_x$, if $x = ⫯$, and (ii) either the directed edges $v^i_y d_e$, if $y = ⫰$, or the directed edge $d_e v^i_y$, if $y = ⫯$.

Now let $D = (V, E)$ be a digraph. We construct a c-graph $\mathcal{C}^* = (G^* = (V^*, E^*), \mathcal{P}^*)$ from D whose every cluster forms a clique of size at most 2, called the *canonical c-graph* for D, as follows. For each vertex $v^i \in V$, G^* contains two vertices $v^i[⫯]$ and $v^i[⫰]$, which form the cluster $V_i = \{v[⫯], v[⫰]\}$ in $\mathcal{P}^*$. For each (directed) edge $v^i v^j$ of D, G^* contains an (undirected) edge $(v^i[⫰], v^j[⫯])$; that is, each directed edge in E that is incoming (outgoing) at a vertex v^i and outgoing (incoming) at a vertex v^j corresponds to an inter-cluster edge in E^* incident to $v^i[⫯]$ (to $v^i[⫰]$) and to $v^j[⫰]$ (to $v^j[⫯]$). Finally, for each vertex $v^i \in V$, G^* contains an intra-cluster edge $(v^i[⫯], v^i[⫰])$. The canonical digraph and the canonical c-graph form dual concepts, as illustrated in Fig. 2.3a; the canonical c-graph of $G^{\diamond}$ is the original c-graph $\mathcal{C}$ (neglecting clusters originated by dummy vertices) and the canonical digraph of $\mathcal{C}^*$ is the original digraph D (suppressing dummy vertices).

NodeTrix Planarity. A *NodeTrix representation* of a c-graph $\mathcal{C} = (G, \mathcal{P})$ is a drawing of $\mathcal{C}$ such that: **(i)** Each cluster $V_i \in \mathcal{P}$ is represented as a symmetric adjacency matrix M_i (with $|V_i|$ rows and columns), drawn in the plane so that its boundary is a square Q_i with sides parallel to the coordinate axes. **(ii)** No two matrices intersect, that is, $Q_i \cap Q_j = \emptyset$, for all $1 \leq i < j \leq c$. **(iii)** Each intra-cluster edge is represented by the adjacency matrix M_i. **(iv)** Each inter-cluster edge (u, v) with $u \in V_i$ and $v \in V_j$ is represented as a simple Jordan arc connecting a point on the boundary of Q_i with a point on the boundary of Q_j, where the point on Q_i (on Q_j) belongs to the column or to the row of M_i (resp. of M_j) associated with u (resp. with v). A NodeTrix representation is *planar* if no inter-cluster edge intersects a matrix or another inter-cluster edge, except possibly at a common end-point; see Figs. 2.1b and 2.3b. The NodeTrix Planarity problem asks whether a c-graph admits a planar NodeTrix representation. NodeTrix Planarity has been proved NP-complete for c-graphs whose clusters have size larger than or equal to 5 [54].

We are ready to establish our main technical lemmas.

Lemma 2.1. *C-graph $\mathcal{C}$ is planar NodeTrix if and only if $G^\diamond$ admits a 6-modal embedding.*

Lemma 2.2. *Digraph D admits a 6-modal embedding if and only if $\mathcal{C}^*$ is planar NodeTrix.*

Proof for Lemmas 2.1 and 2.2. Let M_i be the matrix representing cluster $V_i = \{v^i[⫯], v^i[⫰]\}$. We have that, independently of which of the two possible permutations for the rows and columns of M_i is selected, the boundary of Q_i is partitioned into three maximal portions associated with $v^i[⫯]$ and three maximal portions associated with $v^i[⫰]$; that is, they form the pattern $[1, 2, 1, 2, 1, 2]$, see Fig. 2.3b. Therefore, any planar NodeTrix representation of $\mathcal{C}$ (of $\mathcal{C}^*$) can be turned into a 6-modal embedding of $G^\diamond$ (of D) via a local redrawing procedure which operates in the interior of Q_i; also, any 6-modal embedding of $G^\diamond$ (of D) can be turned into a planar NodeTrix representation of $\mathcal{C}$ ($\mathcal{C}^*$) via a local redrawing procedure which operates in a small disk centered at v_i that contains only v_i and intersects only edges incident to v_i.

Since $G^\diamond$ can be constructed in linear time from $\mathcal{C}$, Lemma 2.1 and the algorithm of Theorem 2.7 for solving k-Modality of directed partial 2-trees give us the following.

Theorem 2.1. NodeTrix Planarity *can be solved in linear time for flat clustered graphs whose clusters have size at most 2 and whose canonical digraph is a directed partial 2-tree.*

Note that (i) $\mathcal{C}^*$ can be constructed in polynomial time from D, (ii) $\mathcal{C}^*$ only contains clusters of size 2 (although clusters corresponding to vertices of D incident to incoming or outgoing edges only could be simplified into clusters of size 1), and (iii) each cluster $V_i \in \mathcal{P}^*$, with $v^i \in V(D)$, is incident to α inter clusters edges, where α is the degree of v^i in D. These properties and the fact that in Theorem 2.12 we prove the k-Modality problem to be NP-complete for digraphs of maximum degree $\Delta \geq k+3$ give us the following.

Theorem 2.2. NodeTrix Planarity *is NP-complete for flat clustered graphs whose clusters have size at most* 2*, even if each cluster is incident to at most* 9 *inter-cluster edges.*

We remark that the above NP-completeness result is best possible in terms of the size of clusters, as clusters of size 1 do not offer any advantage to avoid intersections between inter-cluster edges. Also, it solves [54, Open Problem (i)], which asks for the complexity of NodeTrix Planarity for c-graphs whose clusters have size between 2 and 5.

Clique Planarity. Hybrid representations have also been recently studied in the setting in which clusters are represented via intersections of geometric objects. In particular, Angelini *et al.* [9] introduced the following type of representations. Suppose that a c-graph $(G, \mathcal{P})$ is given, where $\mathcal{P}$ is *a set of cliques* that partition the vertex set of G. In an *intersection-link representation*, the vertices of G are represented by geometric objects that are translates of the same rectangle. Consider an edge (u, v) and let $R(u)$ and $R(v)$ be the rectangles representing u and v, respectively. If (u, v) is an intra-cluster edge (called *intersection-edge* in [9]), we represent it by drawing $R(u)$ and $R(v)$ so that they intersect, otherwise if (u, v) is an intra-cluster edge (called *link-edge* in [9]), we represent it by a Jordan arc connecting $R(u)$ and $R(v)$. A *clique-planar* representation is an intersection-link representation in which no inter-cluster edge intersects the interior of any rectangle or another inter-cluster edge, except possibly at a common end-point. The Clique Planarity problem asks whether a c-graph $(G, \mathcal{P})$ admits a clique-planar representation.

Angelini *et al.* proved the Clique Planarity problem to be NP-complete, when $\mathcal{P}$ contains a cluster V^* with $|V^*| \in O(|G|)$, and asked, in [9, Open Problem (2)], about the implications of using different geometric objects for representing vertices, rather than translates of the same rectangle. We address this question by considering k-combs as geometric objects, where a *k-comb* is the simple polygon with k spikes illustrated in Fig. 2.3c. We have the following.

Lemma 2.3. *C-graph $\mathcal{C}$ is a positive instance of* Clique Planarity *using k-combs as geometric objects if and only if $G^\diamond$ admits a $2k$-modal embedding.*

Lemma 2.4. *Digraph D admits an 4-modal embedding if and only if $\mathcal{C}^*$ is a positive instance of* Clique Planarity *using 2-combs as geometric objects.*

Proof for Lemmas 2.3 and 2.4. Let A_i be an arrangements of 2-combs representing cluster $V_i = \{v^i[\overset{\circ}{\uparrow}], v^i[\underset{\circ}{\uparrow}]\}$. We have that, the boundary of A_i is partitioned into at most two maximal portions associated with $v^i[\overset{\circ}{\uparrow}]$ and at most two maximal portions associated with $v^i[\underset{\circ}{\uparrow}]$; that is, they form the pattern $[1, 2, 1, 2]$, see Fig. 2.3c. Therefore, as for Lemmas 2.1 and 2.2, we can exploit a local redrawing procedure to transform a clique-planar representation of $\mathcal{C}$ (of $\mathcal{C}^*$) into a 4-modal embedding of $G^\diamond$ (of D), and vice versa.

Combining Lemma 2.3 and the algorithm of Theorem 2.7 gives us the following positive result.

Theorem 2.3. Clique Planarity *using r-combs, with $r \geq 1$, as geometric objects can be solved in linear time for flat clustered graphs whose clusters have size at most 2 and whose canonical digraph is a directed partial 2-tree.*

Finally, Lemma 2.4 and the discussion preceding Theorem 2.2 imply the following.

Theorem 2.4. Clique Planarity *using 2-combs as geometric objects is NP-complete, even for flat clustered graphs with clusters of size at most 2 each incident to at most 7 inter-cluster edges.*

Algorithm 1 Function TestSimplyConnected implements the reduction of Theorem 2.5 by exploiting function TestBiconnected to solve 4-MaxModality for biconnected instances. BinarySearch(f, β, m, v) exploits a binary search and function f to compute the minimum modality for cut-vertex v in an embedding of β satisfying m at every vertex different from v.

```
1: function TestSimplyConnected($(\langle G, m\rangle)$)
2:     if $G$ is biconnected then
3:         return TestBiconnected($\langle G, m\rangle$)
4:     $\beta \leftarrow$ leaf-block of $G$ with parent cut-vertex $v$
5:     $\ell \leftarrow$ BinarySearch(TestBiconnected,$\beta, m, v$)
6:     if $\ell \leq m(v)$ then
7:         $m(v) \leftarrow \min(m(v), m(v) - \ell + 2)$
8:         return TestSimplyConnected($\langle G^-_\beta, m\rangle$)
9:     return NO
```

2.4 Polynomial-time Algorithms

In this section, we present an algorithmic framework to devise efficient algorithms for the k-Modality problem for notable families of instances. First, in Section 2.4.1, we show how to efficiently reduce the k-Modality problem in connected digraphs to the k-MaxModality problem in biconnected digraphs. Then, in Section 2.4.2, we introduce preliminaries and definitions concerning SPQR-trees and k-modal embeddings of biconnected digraphs.

2.4.1 Simply-Connected Graphs

We first observe that the k-MaxModality problem is a generalization of the k-Modality problem. In fact, a directed graph $G = (V, E)$ admits a k-modal embedding if and only if the pair $\langle G, m\rangle$, with $m(v) = k, \forall v \in V(G)$, is a positive instance of the k-MaxModality problem.

Observation 2.2. *k-Modality reduces in linear time to k-MaxModality.*

Let $\langle G, m : V(G) \rightarrow \mathbb{E}^+_4\rangle$ be an instance of 4-MaxModality; also, let β be a leaf-block of the BC-tree $\mathcal{T}$ of G and let v be the parent cut-vertex of β in $\mathcal{T}$. We denote by G^-_β the subgraph of G induced by v and the vertices of G not in β, i.e., $G^-_\beta = G - (\beta - \{v\})$. Also, let $B(\mathcal{T})$ be the set of blocks in $\mathcal{T}$. We show that k-MaxModality (and k-Modality, by Observation 2.2) in connected digraphs is Turing reducible to k-MaxModality in biconnected digraphs.

Theorem 2.5. *Given a subroutine* TestBiconnected *that tests* k-MaxModality *for biconnected instances, there exists a procedure* TestSimplyConnected *that tests* k-MaxModality *for connected digraphs. Further, given an instance* $\langle G, m\rangle$ *of* k-MaxModality, *the runtime of* TestSimplyConnected($\langle G, m\rangle$) *is*

$$\mathcal{O}\big(|G| + \log k \sum_{\beta \in B(\mathcal{T})} r(\beta)\big),$$

where $r(\beta)$ *is the runtime of* TestBiconnected($\langle \beta, m\rangle$) *and* $\mathcal{T}$ *is the BC-tree of* G.

Proof. We present procedure TestSimplyConnected in Algorithm 1. The key idea in the algorithm is the following. Consider a leaf-block β of $\mathcal{T}$ with parent cut-vertex v. If $\mathcal{E}$ is an embedding of G, then it induces two embeddings, one for β and one for G^-_β, and v belongs to both subgraphs. If $\mathcal{E}$ satisfies m, then it holds that (i) both induced embeddings satisfy m and (ii) the sum of the modality of v in these embeddings is at most $m(v) + 2$. We show that in fact verifying properties (i) and (ii) is also sufficient to test if G admits an embedding that satisfies m.

The algorithm works recursively as follows. It selects a leaf-block β of $\mathcal{T}$ with parent cut-vertex v and computes the minimum modality ℓ at v for which β admits an embedding that satisfies m. To achieve this goal the algorithm exploits a binary search using subroutine TestBiconnected. Depending on the minimum modality at v it either rejects the instance if no embedding of β satisfying m exists, or sets $m(v) = \min(m(v), m(v) - \ell + 2)$ for G^-_β and recurs on it, otherwise.

To prove the correctness of the algorithm we simply need to show that any pair of embeddings $\mathcal{B}$ of β (computed at line 5) and $\mathcal{B}^-$ of G^-_β (computed at line 8) that satisfy m such that the modality of v in $\mathcal{B}$ is at most ℓ and the modality of v in $\mathcal{B}^-$ is at most $\min(m(v), m(v) - \ell + 2)$ can be composed together to obtain an embedding $\mathcal{E}$ of G that satisfies m.

By rerouting the edges of β to select a different outer face and (possibly) flipping the resulting embedding, we can assume that $\mathcal{B}$ is such that: 1. cut-vertex v lies on the outer face of $\mathcal{B}$, 2. if v is incident to both incoming and outgoing edges, then there exist both an incoming edge w_1 and an outgoing edge w_2 incident to v and to the outer face of $\mathcal{B}$ such that the edges incident to v different from w_1 and w_2 appear after w_1 (and before w_2) in the clockwise order around v.

We construct $\mathcal{E}$ by setting the rotation of every vertex as follows. Observe that, the modality of every vertex $u \neq v$ is at most $m(u)$, by hypothesis. So we set the rotation of each vertex different from v in $\mathcal{E}$ to the same as in $\mathcal{B}^-$, if it belongs to G^-_β, or in $\mathcal{B}$, if it belongs to β. What remains is setting the rotation of vertex v in such a way that the modality of v in $\mathcal{E}$ is at most $m(v)$.

It is useful to think that we are embedding $\mathcal{B}$ in a face of $\mathcal{H}$ incident to v. Ideally, the two edges incident to the face have a different orientation but if no such face exists then the modality of v in $\mathcal{H}$ is 0 and embedding in any face will satisfy m.

We first set the rotation of v in $\mathcal{E}$ in such a way that (i) the clockwise order of the edges incident to v and belonging to β (to G^-_β) is the same as determined by $\mathcal{B}$ (by $\mathcal{H}$) and (ii) the edges of β (of G^-_β) incident to v are consecutive around v. We then distinguish two cases which further constrain the ordering of the edges incident to v.

In the first case, at least one of β and G^-_β, say β, only contains edges with the same orientation incident to v, say incoming at v. Let e_1 be an incoming edge in G^-_β incident to v, or any edge in G^-_β incident to v if no incoming edge exists. Then, we additionally constrain the rotation at v so that an edge incident to v and to the outer face of $\mathcal{B}$ follows e_1 around v. Observe that, this corresponds to inserting a drawing of β whose embedding is $\mathcal{B}$ inside the face of $\mathcal{B}^-$ incident to e_1, and "merging" the two copies of v in β and G^-_β. The case in which β contains only outgoing edges incident to v is symmetric. As for the modality of v in $\mathcal{E}$, if the modality of v is 0 in both $\mathcal{B}$ and $\mathcal{B}^-$ then the modality of v in $\mathcal{E}$ is at most 2. Otherwise, the modality of v in $\mathcal{E}$ is the same as the modality γ of v in $\mathcal{B}^-$, as no alternation has been introduced when composing $\mathcal{B}$ and $\mathcal{B}^-$ to obtain $\mathcal{E}$. Therefore, since $\gamma \leq \min(m(v), m(v) - \ell + 2)$ (line 7) and $\ell = 0$, the modality of v in $\mathcal{E}$ is at most $m(v)$.

In the second case, both β and G^-_β contain incoming and outgoing edges incident to v; refer to Fig. 2.4. First, consider an alternating pair in $\mathcal{B}^-$ consisting of edges e_1 and e_2 of G^-_β incident to v (observe that such a pair always exists). Assume, without loss of generality, that e_1 is incoming at v and that e_2 is outgoing from v and e_1 precedes e_2 clockwise around v. Also consider similar alternating edges w_1 and w_2 of β incident to v.

Then, we additionally constrain the rotation at v so that e_1 precedes clockwise w_1 (w_2 precedes clockwise e_2) around v. Observe that, by construction, the modality of v in $\mathcal{E}$ is equal to $\gamma + \ell - 2$, where γ is the modality of v in $\mathcal{B}^-$ and ℓ is the modality of v in $\mathcal{B}$. This is due to the fact that, the above selection of embedding $\mathcal{B}$ avoids introducing an unnecessary alternation when composing $\mathcal{B}$ and $\mathcal{B}^-$ to obtain $\mathcal{E}$. Also, since $\ell \geq 2$, vertex v has modality $\gamma \leq m(v) - \ell + 2$ in $\mathcal{B}^-$. Thus, the modality of v in $\mathcal{E}$ is at most $m(v)$.

Finally, we discuss the running time of Algorithm 1. The non recursive work of the algorithm only consists of constant time operations and of at most $\lceil \log(k) \rceil$ calls to the subroutine TestBiconnected, while the recursive work consists of at most one call to the function TestSimplyConnected. Therefore, since the total number of calls to subroutine TestBiconnected is bounded by the number of blocks of G, which is $O(|\mathcal{T}|) = O(|G|)$ multiplied by $\log k$, the overall running time is $O(|G| + \log k \sum_{\beta \in B(\mathcal{T})} r(\beta))$. This concludes the proof.

□

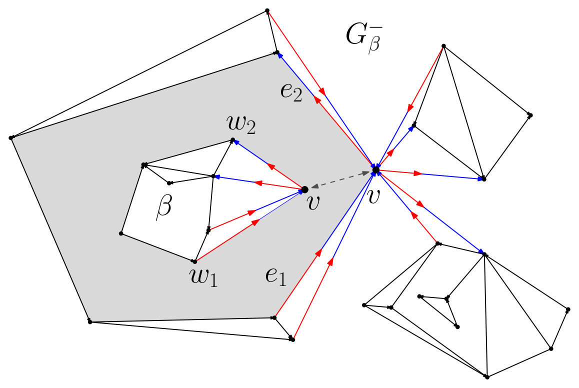


Figure 2.4: Illustration for the proof of Theorem 2.5. The embedding of G_β^- contains a face (in grey) with an alternating pair (e_1, e_2) incident to the cutvertex v; block β can be reconnected to v inside such a face without introducing unnecessary alternations.

2.4.2 Biconnected Graphs

Consider a pair $\langle G, m\rangle$ such that G is biconnected and let $\mathcal{E}$ be a planar embedding of G. Also, let $\mathcal{T}$ be the SPQR-tree of G rooted at an edge e of G incident to the outer face of $\mathcal{E}$. We will assume that the virtual edges of the skeletons of the nodes in $\mathcal{T}$ are oriented so that the extended skeleton of each node μ is a DAG with a single source u_μ and a single sink v_μ. This implies that the virtual edges belonging to the extended skeleton of a P-node have the same orientation, from u_μ to v_μ, and that the virtual edges of the skeleton of an S-node form a directed path from u_μ to v_μ. Let μ be a node of $\mathcal{T}$ and let $\mathcal{E}_\mu$ be the planar (regular) embedding of skel(μ) induced by $\mathcal{E}$. For an oriented edge $d = uv$ of skel(μ), the *left* and *right face* of d in $\mathcal{E}_\mu$ is the face of $\mathcal{E}_\mu$ seen to the left and to the right of d, respectively, when traversing this edges from u to v. We define the *outer left (right) face* of $\mathcal{E}_\mu$ as the left (right) face of the edge $u_\mu v_\mu$ in $\mathcal{E}_\mu$.

Embedding tuples. An *embedding tuple* (for short, *tuple*) is a 4-tuple $\langle \sigma_1, a, \sigma_2, b\rangle$, where $\sigma_1, \sigma_2 \in \{$⫰, ⫯$\}$ are orientations and $a, b \in \mathbb{N}$ are non-negative integers. Consider two tuples $t = \langle \sigma_1, a, \sigma_2, b\rangle$ and $t' = \langle \sigma_1', a', \sigma_2', b'\rangle$. We say that t *dominates* t', denoted as $t \preceq t'$, if $\sigma_1 = \sigma_1'$, $\sigma_2 = \sigma_2'$, $a \leq a'$, and $b \leq b'$. Also, we say that t and t' are *incompatible*, if none of them dominates the other. Since the relationship $\preceq$ is reflexive, antisymmetric,

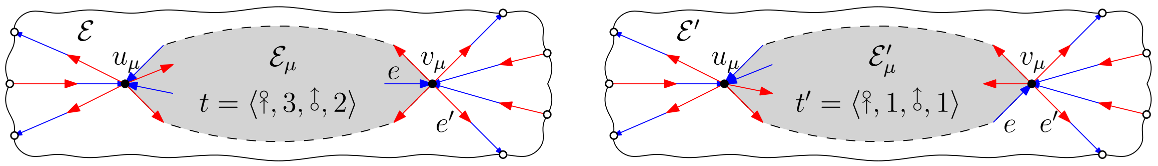


Figure 2.5: Illustration for the proof of Lemma 2.5. The parity of t and t' is the same at u_μ and different at v_μ; in particular, even if a new alternation is introduced between the pair (e, e') at v_μ, the different parity guarantees that the modality at v_μ does not increase from $\mathcal{E}$ to $\mathcal{E}'$.

and transitive, it defines a poset $(T, \preceq)$, where T is the set of embedding tuples. A subset $S \subseteq T$ is *succinct* or an *antichain*, if the tuples in S are pair-wise incompatible. Consider two subsets $S, S' \subseteq T$ of tuples. We say that S *dominates* S', denoted as $S \preceq S'$, if for any tuples $t' \in S'$ there exists at least one tuple $t \in S$ such that $t \preceq t'$. Also, S *reduces* S' if $S \preceq S'$ and $S \subseteq S'$. Finally, S is a *gist* of S', if S is succinct and reduces S'.

Let e_u and e_v be the edges of $\text{pert}(\mu)$ incident to the outer left face of $\mathcal{E}_\mu$ and to u_μ and v_μ, respectively, possibly $e_u = e_v$. Also, let a and b be non-negative integers. We say that the embedding $\mathcal{E}_\mu$ *realizes* tuple $\langle \sigma_1, a, \sigma_2, b\rangle$, if $\sigma_1 = \sigma(e_u, u_\mu)$, $\sigma_2 = \sigma(e_v, v_\mu)$, and a and b are the number of inner faces of $\mathcal{E}_\mu$ whose (two) edges incident to u_μ and to v_μ, respectively, form an alternating pair. A tuple $t = \langle \sigma_1, a, \sigma_2, b\rangle$ is *realizable by* μ, if there exists an embedding of $\text{pert}(\mu)$ satisfying m that realizes t, and *admissible*, if $a \leq m(u)$ and $b \leq m(v)$. A tuple is *good for* μ if it is both admissible and realizable by μ. We denote by $S(\mu)$ the gist of the set of good tuples for a node μ. Let e_μ be the virtual edge representing μ in the skeleton of the parent of μ in $\mathcal{T}$, with a small overload of notation, we also denote $S(\mu)$ by $S(e_\mu)$. For a tuple $t = \langle \sigma_1, a, \sigma_2, b\rangle \in S(e_\mu)$, where $e = u_\mu v_\mu$, the pair (σ_1, a) is the *embedding pair* of t at u_μ; likewise, the pair (σ_2, b) is the *embedding pair* of t at v_μ. We have the following substitution lemma.

Lemma 2.5. *Let $\mathcal{E}$ be a planar embedding of G satisfying m. Let μ be a node of $\mathcal{T}$ and let $\mathcal{E}_\mu$ be the embedding of* $\text{pert}(\mu)$ *induced by $\mathcal{E}$. Also, let $\mathcal{E}'_\mu \neq \mathcal{E}_\mu$ be an embedding of* $\text{pert}(\mu)$ *satisfying m. Then, G admits an embedding $\mathcal{E}'$ satisfying m in which the embedding of* $\text{pert}(\mu)$

is $\mathcal{E}'_\mu$, if $t' \preceq t$, where t and t' are the embedding tuples realized by $\mathcal{E}_\mu$ and by $\mathcal{E}'_\mu$, respectively.

Proof. We show how to construct a drawing Γ'_G of G satisfying m in which the embedding of pert(μ) is $\mathcal{E}'_\mu$; see Fig. 2.5. Let Γ_G be a drawing of G whose embedding is $\mathcal{E}$. Remove from Γ_G the drawing of all the vertices of pert(μ) different from u_μ and v_μ and the drawing of all the edges of pert(μ). Denote by f the face of the resulting embedded graph G^- that used to contain the removed vertices and edges. We obtain Γ'_G by inserting a drawing of pert(μ) whose embedding is $\mathcal{E}'_\mu$ in the interior of f so that vertices u_μ and v_μ are identified with their copies in G^-.

We claim that the embedding $\mathcal{E}'$ of Γ'_G satisfies m. First, the modality of each vertex of G not in pert(μ) is the same in $\mathcal{E}'$ as in $\mathcal{E}$. Second, the modality of each vertex in pert(μ) different from u_μ and v_μ is the same in $\mathcal{E}'$ as in $\mathcal{E}'_\mu$.

We only need to show that u_μ and v_μ satisfy m in $\mathcal{E}'$. We have that $t = \langle \sigma_1, a, \sigma_2, b \rangle$ and $t = \langle \sigma'_1, a', \sigma'_2, b' \rangle$. We show that the modality of u_μ in $\mathcal{E}'$ is smaller than or equal to the modality of u_μ in $\mathcal{E}$; analogous arguments hold for v_μ. We distinguish two cases. If a and a' have the same parity, as shown at vertex u in Fig. 2.5, then $\mathcal{E}'$ contains an alternating pair consisting of an edge in G' and of an edge in pert(μ) incident to u_μ only if $\mathcal{E}$ contains an alternating pair consisting of an edge in G' and of an edge in pert(μ) incident to u_μ. Therefore, since $a' \leq a$, the modality of u_μ in $\mathcal{E}'$ is smaller than or equal to the modality of u_μ in $\mathcal{E}$. Otherwise, if a and a' have the different parity, as shown at vertex v in Fig. 2.5, then $\mathcal{E}'$ may contain an alternating pair consisting of an edge in G' and of an edge in pert(μ) incident to u_μ and to the right outer face of pert(μ) even if $\mathcal{E}$ does not contain an alternating pair consisting of an edge in G' and of an edge in pert(μ) incident to u_μ. However, in this case, it holds that $a' < a$, therefore the modality of u_μ in $\mathcal{E}'$ is again smaller than or equal to the modality of u_μ in $\mathcal{E}$. □

Let $\mathcal{T}$ be the SPQR-tree $\mathcal{T}$ of G rooted at a reference edge e. In the remainder of the section, we show how to compute the gist $S(\mu)$ of the set of good tuples for μ, for each non-root node μ of $\mathcal{T}$. In the subsequent procedures to compute $S(\mu)$ for S-, P-, and R-nodes, we are not going to explicitly avoid set $S(\mu)$ to contain dominated tuples. In fact, this can always be done at the cost of an additive $O(k^2)$ factor in the running time, by maintaining an hash table that stores the tuples that have been constructed (possibly multiple times) by the procedures and by computing the gist of the constructed set as a final step.

Property 2.1. *For each node $\mu \in \mathcal{T}$, it holds that $|S(\mu)| \in O(k)$.*

Proof. By the definition of gist, any embedding pair (σ, a) has at most two tuples $t', t'' \in S(\mu)$ such that (σ, a) is the embedding pair of t' and t'' at u_μ; also, the embedding pairs (σ', a') of t' and (σ'', a'') of t'' at v_μ are such that $\sigma' \neq \sigma''$. Since there exist at most $2k$ realizable embedding pairs (σ, a) at u_μ (as $\sigma \in \{⫯, ⫰\}$, $a \in \{0, 1, \dots, k\}$, and the existence of tuple whose embedding pair at u_μ is $(\sigma, 0)$ implies that all tuples have such an embedding pair at u_μ), we have $|S(\mu)| \leq 4k$. □

If μ is a leaf Q-node in $\mathcal{T}$, then $S(\mu) = \{\langle \sigma(u_\mu v_\mu), 0, \sigma(u_\mu v_\mu), 0 \rangle\}$. If μ is an internal node of $\mathcal{T}$, we visit $\mathcal{T}$ bottom-up and compute the set $S(\mu)$ for μ assuming to have already computed the sets $S(\mu_1), \dots, S(\mu_k)$ for the children $\mu_1, \dots, \mu_k$ of μ (where μ_i is the child of μ corresponding to the edge e_i in skel(μ)). Let ρ be the unique child of the root of $\mathcal{T}$. Once the set $S(\rho)$ has been determined, we can efficiently decide whether G admits an embedding satisfying m in which the reference edge e is incident to the outer face by means of the following lemma.

Lemma 2.6. *Given $S(\rho)$, we can test whether G has an embedding that satisfies m in $O(k^2)$ time.*

Proof. Let $t = \langle \sigma_1, a, \sigma_2, b \rangle$ be a realizable tuple in $S(\rho)$ and let $\mathcal{E}_t$ be the embedding of G obtained by inserting the reference edge e in the outer face of a regular embedding $\mathcal{E}_\rho$ of

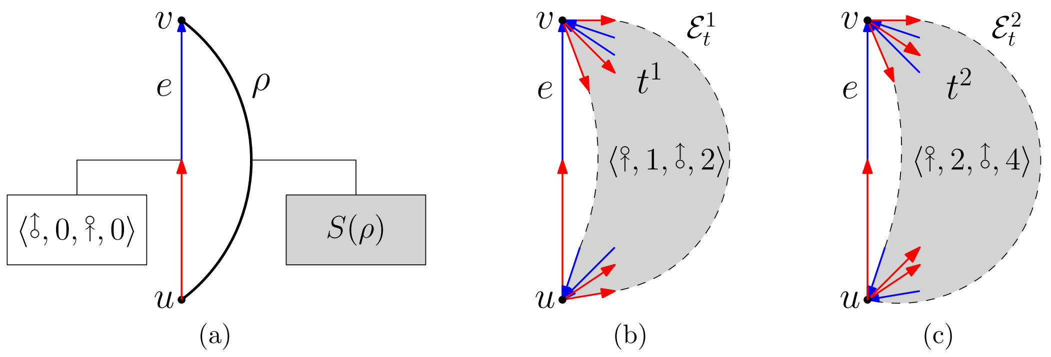


Figure 2.6: Illustrations for the proof of Claim 2.1. (a) The root edge e and ρ (where the orientation of ρ being arbitrary). (b) A 4-modal embedding of G, where u has 2 alternations and v has 4 alternations. (b) A 6-modal embedding of G, where u has 4 alternations and v has 6 alternations.

pert(ρ) realizing t that satisfies m. Let u and v be the poles of ρ. We have the following claim.

Claim 2.1. *Embedding $\mathcal{E}_t$ satisfies m if and only if:*

(i) (a) $m(u) \geq a+1$, if a is odd, or (b) $m(u) \geq a$, if a is even and $\sigma_1 = \sigma(e)$, or (c) $m(u) \geq a+2$, if a is even and $\sigma_1 \neq \sigma(e)$, and

(ii) (a) $m(v) \geq b+1$, if b is odd, or (b) $m(v) \geq b$, if b is even and $\sigma_2 = \sigma(e)$, or (c) $m(v) \geq b+2$, if b is even and $\sigma_2 \neq \sigma(e)$.

Proof. To prove the statement for vertex u, we just need to observe that, if a is odd, then the edges e'_u and e''_u incident to u and to the left and to the right outer face of $\mathcal{E}_\rho$, respectively, have opposite orientations, while if a if even, then these edges have the same orientation (in this case, possibly $e'_u = e''_u$). Also, since a is the number of alternations between edges incident to u and to the internal faces of $\mathcal{E}_\rho$, the modality at u in $\mathcal{E}_\rho$ is equal to $a+1$, if a is odd, while it is equal to a, otherwise. Therefore, since edge e appears between e'_u and e''_u in embedding $\mathcal{E}_t$ obtained from $\mathcal{E}_\rho$, we have that the number of alternations around u in $\mathcal{E}_t$ is the same as in $\mathcal{E}_\rho$, if a is odd or if a is even and $\sigma(e) = \sigma_1$, and it is equal to the modality of

u in $\mathcal{E}_\rho$ plus 2, if a is even and $\sigma(e) \neq \sigma_1$. Refer to Fig. 2.6. The proof of the statement for vertex v is analogous. □

By Claim 2.1, for each realizable tuple $t \in S(\rho)$, we can test whether embedding $\mathcal{E}_t$ satisfies m in constant time. Also, $|S(\rho)| \in O(k^2)$, by Property 2.1. Therefore, we can test in $O(k^2)$ time whether there exists an embedding of $\text{pert}(\rho)$ that can be extended to an embedding of G that satisfies m in which e is incident to the outer face. □

We remark that the choice of the reference edge does not affect the existence of a 4-modal embedding. In fact, a change of the outer face such that edge e is incident to such a face can always be performed while preserving the rotation at any vertex.

2.5 Partial 2-trees

In the following, we describe how to compute $S(\mu)$, if μ is an S-node (Lemma 2.7) and a P-node (Lemma 2.8) in $O(f(k)|\,\text{skel}(\mu)|)$ time, where f is a computable function.

Lemma 2.7. *Set $S(\mu)$ can be constructed in $O(k^2|\,\text{skel}(\mu)|)$ time for an S-node μ.*

Proof. Let μ be an S-node with skeleton $\text{skel}(\mu) = (e_1, e_2, \dots, e_h)$, where $e_i = u_{\mu_i} v_{\mu_i}$ is the i-th virtual edge of $\text{skel}(\mu)$. We define τ_i as the S-node obtained by the series composition of $\mu_1, \mu_2, \dots, \mu_i$. Clearly, $\text{pert}(\tau_i) \subseteq \text{pert}(\mu)$ and $\text{skel}(\tau_i) = (e_1, e_2, \dots, e_i)$. Initially $S(\tau_1) = S(\mu_1)$. We show how to construct $S(\tau_i)$, for $i = 2, \dots, h$. Since $\tau_k = \mu$, this gives us $S(\mu)$. The key idea is that when doing the series composition we only need to consider the embedding pairs at the shared vertex.

Consider two adjacent virtual edges $e_j = v_{j-1} v_j$ and $e_{j+1} = v_j v_{j+1}$ in $\text{skel}(\mu)$ sharing the internal vertex v_j, and let $t_1 = (\sigma_1', a, \sigma_2', b) \in S(\mu_j)$ and $t_2 = (\sigma_1'', c, \sigma_2'', d) \in S(\mu_{j+1})$. We define a function $g(\sigma_2', b, \sigma_1'', c)$ on the embedding pairs of t_1 and of t_2 at v_j which determines the modality at v_j when an embedding of μ_j realizing t_1 and an embedding of μ_{j+1} realizing t_2 are composed. The value of function $g(\cdot)$ can easily be computed in $O(1)$ time for any pair of tuples $t_1 \in S(\mu_j)$ and $t_2 \in S(\mu_{j+1})$.

For any $2 \leq j \leq h$, we obtain $S(\tau_j)$ from $S(\tau_{j-1})$ and $S(\mu_j)$ as follows. For each tuple $t_1 = (\sigma'_1, a, \sigma'_2, b) \in S(\tau_{j-1})$ and $t_2 = (\sigma''_1, c, \sigma''_2, d) \in S(\mu_j)$, we add tuple $\langle \sigma'_1, a, \sigma''_2, d \rangle$ to $S(\tau_j)$, if $g(\sigma'_2, b, \sigma''_1, c) \leq m(v_j)$. It is clear that the set $S(\tau_j)$ computed in the above fashion contains all and only all the tuples realizable by some embedding of $\text{pert}(\tau_j)$. Since both $|S(\tau_{j-1})|$ and $|S(\mu_j)|$ are in $O(k)$, by Property 2.1, and since for each pair of tuples (t_1, t_2) with $t_1 \in S(\tau_{j-1})$ and $t_2 \in S(\mu_j)$ we only perform constant-time operations, we have that $S(\tau_j)$ can be computed in $O(k^2)$ time (from $S(\tau_{j-1})$ and $S(\mu_j)$), for any $1 < j \leq h$. Therefore, the overall running time for computing $S(\mu)$ is $O(k^2 |\,\text{skel}(\mu)|)$. □

Lemma 2.8. *Set $S(\mu)$ can be constructed in $O((2k+4)!k^3 + |\,\text{skel}(\mu)|)$ time for a P-node μ.*

Proof. Let μ be a P-node with poles u_μ and v_μ, whose skeleton $\text{skel}(\mu)$ consists of h parallel virtual edges $e_1, e_2, \ldots, e_h$. Consider two children μ_1 and μ_2 of μ such that: (i) all the edges incident to u_μ in $\text{pert}(\mu_1)$ and in $\text{pert}(\mu_2)$ have the same orientation σ_u and (ii) all the edges incident to v_μ in $\text{pert}(\mu_1)$ and in $\text{pert}(\mu_2)$ have the same orientation σ_v. Clearly, $S(\mu_1) = S(\mu_2) = \{\langle \sigma_u, 0, \sigma_v, 0 \rangle\}$. We have the following claim.

Claim 2.2. *Let τ' be the P-node such that* $\text{skel}(\tau') = \text{skel}(\mu) - e_2$. *Then,* $S(\tau') = S(\mu)$.

Proof. Let $\mu_1, \ldots, \mu_h$ be the children of μ associated with $e_1, \ldots, e_h$, respectively.

We prove that a tuple t belongs to $S(\tau')$ if and only if it belongs to $S(\mu)$.

Let t be an embedding tuple in $S(\tau')$, and let $\mathcal{E}'$ be an embedding of $\text{pert}(\tau')$ realizing t. The embedding $\mathcal{E}'$ is defined by a permutation $\pi_{\tau'}$ of the virtual edges of $\text{skel}(\tau')$ and by a choice for an embedding of the pertinent graphs of all the children of τ'. We construct an embedding $\mathcal{E}$ of pert(μ) realizing t, by selecting a permutation π_μ of the virtual edges of skel(μ) and an embedding of the pertinent graphs of all the children of μ, as follows. First, we initialize $\pi_\mu = \pi_{\tau'}$, and then we insert the virtual edge e_2 in π_μ so that it immediately follows e_1. Second, we set the embedding of the pertinent graphs of the children of μ that are also children of τ' to the one they have in $\mathcal{E}'$, and we set the embedding of pert(μ_2) to be any embedding of such a pertinent graph that satisfies m. Clearly, the resulting embedding of pert(μ) satisfies m. Also, since the edges incident to u_μ (resp., to v_μ) have the same orientation both in pert(μ_1) and in pert(μ_2) at u_μ (resp., at v_μ), and since such edges appear consecutively around u_μ (resp., around v_μ), by construction, we have that the embedding type of $\mathcal{E}$ is the same at the one of $\mathcal{E}'$. This concludes the proof that $t \in S(\mu)$.

Let $t = \langle \sigma_1, a, \sigma_2, b \rangle$ be an embedding tuple in $S(\mu)$, and let $\mathcal{E}$ be an embedding of pert(μ) realizing t. The embedding $\mathcal{E}$ is defined by a permutation π_μ of the virtual edges of skel(μ) and by a choice for an embedding of the pertinent graphs of all the children of μ.

First, we show that we may assume that e_1 immediately precedes e_2 in π_μ. Suppose, without loss of generality by a possible renaming of such virtual edges, that e_1 precedes e_2 in π_μ, and that e_1 and e_2 are not consecutive in π_μ. Let $\mathcal{E}^*$ be the embedding of pert(μ) obtained by moving e_2 right after e_1 in π_μ, while keeping unchanged the embedding of each of the pertinent graphs of the children of μ. Clearly, the embedding tuple t^* of $\mathcal{E}^*$ is of the form $\langle \sigma_1, c, \sigma_2, d \rangle$, as the edges incident to the left outer face of $\mathcal{E}^*$ and to u_μ (resp., and to v_μ) are the same edges incident to the left outer face of $\mathcal{E}$ and to u_μ (resp., and to v_μ). Also, $c \leq a$ and $d \leq b$ holds, since the edges incident to u_μ (resp., to v_μ) have the same orientation both in pert(μ_1) and in pert(μ_2) at u_μ (resp., at v_μ). Further, neither $c < a$ nor $d < b$ holds, as otherwise t^* would dominate t, which contradicts $t \in S(\mu)$. Therefore, we have $t^* = t$.

Then, by the above assumption, we obtain an embedding $\mathcal{E}'$ of pert(τ') with the same embedding type as the one of $\mathcal{E}$, by simply restricting $\mathcal{E}$ to $\bigcup_{i \neq 2}$ pert(μ_i). This concludes the proof that $t \in S(\tau')$ and the proof of the claim. $\square$

By Claim 2.2, we can exploit the preprocessing step that precedes the claim to reduce in $O(|\operatorname{skel}(\mu)|)$ time the computation of $S(\mu)$ to the computation of $S(\tau)$, where τ is a P-node whose skeleton consists of at most $2k+4$ virtual edges of $\operatorname{skel}(\mu)$. The P-node τ contains at most four children each of which has the property that its pertinent graph contains only edges with the same orientation at u_μ and only edges with the same orientation at v_μ. Also, every other child of τ contributes with at least one alternating pair of edges incident to u_μ or to v_μ in any planar embedding of $\operatorname{pert}(\tau)$. Therefore, in order for $\operatorname{pert}(\tau)$ to admit an embedding that satisfies m, there must exist at most $2k$ such children of μ (and of τ). Thus, if $\operatorname{skel}(\tau)$ contains more that $2k+4$ virtual edges after the preprocessing described above, then we can immediately determine that $S(\mu) = S(\tau) = \emptyset$. Otherwise, τ contains at most $2k+4$ virtual edges and we construct $S(\tau)$ as follows.

For a permutation π of the virtual edges of $\operatorname{pert}(\tau)$, let τ_i^π be the P-node obtained by restricting τ to the first i virtual edges in π. Set the embedding of $\operatorname{skel}(\tau_i^\pi)$ so that the virtual edges of $\operatorname{skel}(\tau_i^\pi)$ are ordered according to π. Then, in a fashion similar to the S-node case, we can compute $S(\tau_i^\pi)$ for the given embedding of $\operatorname{skel}(\tau_i^\pi)$ by combining $S(\tau_{i-1}^\pi)$ and $S(e_i)$ in $O(k^2)$ time (recall that both these sets have size $O(k)$, by Property 2.1). Clearly, for any fixed π, we can compute $S(\tau_\pi^h)$ in $O(k^3)$ time. Thus, by performing the above computation for all the $(2k+4)!$ possible permutations for the virtual edges of $\operatorname{pert}(\tau)$, we can construct $S(\tau)$ in $O((2k+4)!k^3 + |\operatorname{skel}(\mu)|)$ time. □

Altogether, Lemmas 2.7 and 2.8 yield the following main result.

Theorem 2.6. k-MaxModality *can be solved in* $O((2k+4)!k^3n)$ *for series-parallel digraphs.*

Observation 2.2, Theorem 2.5, and Theorem 2.6 immediately imply the following.

Corollary 2.1. k-Modality *can be solved in* $O(((2k+4)!k^3 \log k)n)$ *for directed partial* 2-*trees.*

Due to the special algorithmic framework we are employing, we can however turn the multiplicative $O(\log k)$ factor in the running time into an additive $O(k)$ factor by modifying Algorithm 1 as follows. When considering a cut-vertex v (line 4), we will execute "only once" the function TestBiconnected (line 5) by rooting the SPQ-tree at a Q-node η corresponding to an edge incident to v. This will allow us to compute the minimum modality for cut-vertex v in an embedding that satisfies m, by simply scanning the set $S(\eta)$, which takes $O(k)$ time by Property 2.1, rather than by exploiting a logarithmic number of calls to TestBiconnected.

Theorem 2.7. k-Modality *can be solved in* $O((2k+4)!k^3n)$ *for directed partial* 2*-trees.*

2.6 A Linear-time Algorithm for 4-MaxModality when $\Delta \leq 6$

In this section, we show that in the special case when $k = 4$ and G has maximum degree $\Delta \leq 6$, it is possible to compute the set $S(\mu)$ when μ is an R-node in linear time in the size of skel(μ).

Our strategy to compute $S(\mu)$ is as follows. We select a single tuple from the admissible set of each virtual edge incident to u_μ and v_μ, in every possible way. Each selection determines a "*candidate tuple*" t for $S(\mu)$. First, we check if t is admissible at both u and v. Second, we restrict the tuples of the edges incident to the poles to only the tuples that form t and check if there is a way of satisfying m at the (inner) vertices of skel(μ). If both the poles and the inner vertices are satisfiable, then we add t to $S(\mu)$. Since the degrees of the poles are bounded, there is at most a constant number of candidate tuples which must be checked. The complexity lies in this check.

We now formally describe how to compute $S(\mu)$. First, for each virtual edge e_i of skel(μ) incident to the poles of μ, we select a tuple t_i from $S(\mu_i)$. That is, we allow only embedding realizing t_i for the pertinent graph of child μ_i. Let $T_u = [t_{u,1}, t_{u,2}, \dots, t_{u,\ell}]$ be the list of tuples selected for the virtual edges $e_{u,1}, e_{u,2}, \dots, e_{u,\ell}$ of skel(μ) incident to u_μ and let $T_v = [t_{v,1}, t_{v,2}, \dots, t_{v,h}]$ be the list of tuples selected for the virtual edges $e_{v,1}, e_{v,2}, \dots, e_{v,h}$ of skel(μ) incident to v_μ. Without loss of generality, we assume virtual edges $e_{u,1}$ and $e_{v,1}$ are the virtual edges of skel(μ) incident to u_μ and to v_μ, respectively, and to the left outer face of the unique (up to a flip) embedding of skel(μ); possibly $e_{u,1} = e_{v,1}$.

Each pair of lists T_u and T_v yields a candidate tuple $t = \langle \sigma_1, a, \sigma_2, b \rangle$ for μ, where σ_1 is the orientation of $t_{u,1}$ at u, a is the number of alternations at u determined by T_u, σ_2 is the orientation of $t_{v,1}$ at v, and b is the number of alternations at v determined by T_v. Observe that the same tuple t might originate from different pairs of lists for u_μ and v_μ. The tuples selected to construct T_u and in T_v allow for an embedding of pert(μ) realizing tuple t *if and only if*:

Condition 1: tuple t satisfies m at u_μ and at v_μ, and

Condition 2: it is possible to select tuples for each of the remaining virtual edges of skel(μ) that satisfy m at every internal vertex of skel(μ).

Let $\mathcal{P}(\mu)$ be the set of candidate tuples for μ constructed as described above. By Property 2.1, the admissible set of any virtual edge has at most $O(1)$ tuples and there are at most 5 virtual edges in skel(μ) incident to each pole of μ (since $\Delta \leq 6$), so $|\mathcal{P}(\mu)| \in O(1)$. Thus, we can easily filter out the candidate tuples that do not satisfy Condition 1 in total constant time. Therefore, in the following we assume that all the candidate tuples in set $\mathcal{P}(\mu)$ satisfy such a condition.

In the remainder of this section, for each pair of lists T_u and T_v yielding a tuple $t \in \mathcal{P}(\mu)$, we will show how to test Condition 2 for μ in linear time. This and the fact that $|\mathcal{P}(\mu)| \in O(1)$ imply the following.

Lemma 2.9. *Set $S(\mu)$ can be constructed in $O(|\operatorname{skel}(\mu)|)$ time for an R-node μ, if $\Delta \leq 6$.*

Altogether, Lemmas 2.7, 2.8 and 2.9 yield the following main result.

Lemma 2.10. 4-MaxModality *can be solved in linear time for biconnected digraphs with* $\Delta \leq 6$.

Observation 2.2, Theorem 2.5, and Lemma 2.10 immediately imply the following.

Theorem 2.8. 4-Modality *can be solved linear time for digraphs with* $\Delta \leq 6$.

To prove Lemma 2.9, we show how to solve the following auxiliary problem for special instances.

Problem: 4-MaxSkelModality

Input: A triple $\langle G = (V, E), \mathcal{S} = \{S(e_1), \ldots, S(e_{|E|})\}, m\rangle$ where G is an embedded directed graph, each $S(e_i)$ is a set containing embedding tuples for the virtual edge $e_i \in E$, and $m : V \to \mathbb{E}_4^+$ is the *maximum-modality function*.

Question: Can we select a tuple from each set $S(e_i)$ in such a way that the modality at each vertex $v \in V$ is at most $m(v)$?

For each pair of lists T_u and T_v yielding a candidate tuple in $\mathcal{P}(\mu)$, we will construct an instance $I_\mu(T_u, T_v) = (G, \mathcal{S}, m)$ of 4-MaxSkelModality as follows. 1. We set $G = \text{skel}(\mu)$ and we fix the embedding of G to be equal to the unique regular embedding of skel(μ); 2. for each virtual edge $e_{u,i}$ incident to u_μ, with $i = 1, \ldots, \ell$, we set $S(e_{u,i}) = \{t_{u,i}\}$; for each virtual edge $e_{v,j}$ incident to v_μ, with $j = 1, \ldots, h$, we set $S(e_{v,j}) = \{t_{v,j}\}$; for each of the remaining virtual edges e_d of skel(μ), we set $S(e_d) = S(\mu_d)$; finally, 3. the maximum-modality function of I_μ coincides with m.

Clearly, $I_\mu(T_u, T_v)$ is a positive instance of 4-MaxSkelModality if and only if, given the constrains imposed by the tuples in T_u and in T_v, there exists a selection of tuples for the edges of G not incident to u_μ or v_μ that satisfies m at all the internal vertices of G, i.e., Condition 2 holds.

Let v be a vertex of G and let e be an edge in $E(v)$, we denote by $A_v(e)$ the maximum number of alternations at v over all the tuples in $S(e)$.

Definition 2.1 (Good instances)**.** *An instance of* 4-MaxSkelModality *is* good *if, for any vertex v in G, it holds $\sum_{e \in E(v)}(A_v(e)+1) \leq 6$.*

Note that, for each edge e in $ske(\mu)$ incident to a vertex v, the pertinent graph H_e of the child of μ associate with e contributes at least $A_v(e)+1$ edges to $d_{H_e}(v)$. Thus, we have $\sum_{e \in E(v)}(A_v(e)+1) \leq \sum_{e \in E(v)} d_{H_e}(v) \leq 6$. Therefore, instance $I_\mu(T_u, T_v)$ is good. Although 4-MaxSkelModality turns out to be NP-complete in general (Theorem 2.12), we are now going to show the following main positive result.

Theorem 2.9. 4-MaxSkelModality *is linear-time solvable for good instances.*

The outline of the linear-time algorithm to decide whether a good instance $I = \langle G = (V,E), \mathcal{S} = \{S(e_1), \ldots, S(e_{|E|})\}, m\rangle$ of 4-MaxSkelModality is a positive instance is a follows; see also Algorithm 2.

- We process I by means of a set of *reduction rules* applied locally at the vertices of G and their incident edges. Each of these rules, if applicable, either detects that the instance I is a negative instance or transforms it into an equivalent smaller instance $I' = \langle G', \mathcal{S}', m'\rangle$. Each rule can be applied when specific conditions are satisfied at the considered vertex. A rule may additionally set a vertex as *marked*. Any marked vertex v has the main property that **any selection** of tuples from the admissible sets of the edges incident to v satisfies m' at v.

- Let I^* be the instance of 4-MaxSkelModality obtained when no reduction rule may be further applied. We prove that instance I^* has a special structure that allows us to reduce the problem of testing whether I^* is a positive instance of 4-MaxSkelModality to that of verifying the NAE-satisfiability of a constrained instance of NAESAT, in fact, of Planar NAESAT. Since Planar NAESAT is in P [82], this immediately implies that 4-MaxSkelModality is also in P. However, in Section 2.7, by strengthening a result of Porschen *et al.* [86], we are able to show that the constructed instances of NAESAT are always satisfiable and that a satisfying NAE-truth assignment can be computed in linear time.

Consider an unmarked vertex v of G. We provide three reduction rules that, if applicable, turn a good instance I into an equivalent smaller good instance I'. Each rule consists of a *condition* for the rule to be applied and of a *reduction* describing how instance I' is obtained from I.

We further assume the rules to be applied according to the *order of priority* given below (see Algorithm 2). Notice that as a side-effect of applying a reduction rule, a neighbor of v in G' may now satisfy the condition of a reduction rule.

We now clarify what we mean by a "smaller" instance. Let $U(G)$ be the set of unmarked vertices of G. Let $\phi(I) = \sum_{v\in U(G)} \sum_{e\in E(v)} |S(e)|$ and let $\xi(I) = 32 \cdot |E(G)|$. Let $\pi(I) = \phi(I) + \xi(I)$ be a function that associates an integer to instance I. We say that $\phi(I)$ is the *potential* of I. Observe that, $\pi(I)$ is always positive and $\pi(I) \in O(|V(G)|)$. The latter is due to the fact that (i) $\phi(I) \in O(|V(G)|)$, since G has bounded degree and since $|S(e)| \in O(1)$ for any edge $e \in E(G)$, by Property 2.1, and that (ii) $\xi(I) \in O(|V(G)|)$, since $O(|E(G)|) \in O(|V(G)|)$ due to the fact that G is planar. We shall say that an instance I' is *smaller* than an instance I if $\pi(I') < \pi(I)$.

Let u of G be a vertex and let $e_1, \dots, e_h$ be the edges incident to u. By selecting a single tuple from each set $S(e_i)$, we generate a *combination of tuples* for u. Since the rotation of u is fixed, each combination determines a specific modality at v. Thus, it is natural to talk of a *satisfying combination*, if m is satisfied by the combination at u, and of an *unsatisfying* combination, otherwise Let $N_u(e) \subseteq S(e)$ be the set containing all the tuples of e that do not appear in any satisfying combination for u, where e is an edge incident to u.

Algorithm 2 Procedure ReduceInstance

```
1: procedure ReduceInstance($I = \langle G, \mathcal{S}, m \rangle$)
2:     $\mathcal{Q} = [\,]$                    ▷ $\mathcal{Q}$ is the queue of vertices to be processed
3:     $\mathcal{V} \leftarrow V(G)$
4:     Set vertices in $\mathcal{V}$ to unmarked
5:     while $\mathcal{V} \neq \emptyset$ do
6:         $u \leftarrow$ extract a vertex from $\mathcal{V}$
7:         $\mathcal{Q} \leftarrow \mathcal{Q} \cup u$
8:         while $\mathcal{Q} \neq \emptyset$ do
9:             $v \leftarrow$ extract a vertex from $\mathcal{Q}$
10:            for $i = 1, 2, 3$ do          ▷ Rule 1 may reject $I$; Rule 2 sets $v$ as marked
11:                Apply, if possible, Rule $i$ at $v$
12:            if Rules 1 or 3 have been applied then     ▷ Only Rules 1 and 3 modify $I$
13:                $\mathcal{U}_v \leftarrow$ unmarked neighbors of $v$
14:                $\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{U}_v$
15:                $\mathcal{V} \leftarrow \mathcal{V} \setminus \mathcal{U}_v$
```

▷ Rule 1:

Condition: Let $e_1, \ldots, e_h$ be the edges of G incident to v. There exists $i \in \{1, \ldots, h\}$ such that $N_v(e_i) \neq \emptyset$.

Reduction: If $N_v(e_i) = S(e_i)$, for any $i \in \{1, \ldots, h\}$, then reject I. Otherwise, initialize $I' = I$ and set $S'(e_i) = S(e_i) \setminus N_v(e_i)$, for $i = 1, \ldots, h$. —*End of* Rule 1

It is clear that, since no tuple $t \in N_v(e_i)$ appears in any valid solution for I, removing the tuples in $N_v(e_i)$ from $S(e_i)$ yields an equivalent instance I'. Also, for an unmarked vertex v, we can compute I' from I in constant time. This is due to the fact that since v has maximum degree 6 in G and since, by Property 2.1, each edge incident to v contains a constant number of tuples, it is possible to test whether a tuple $t \in S(e_i)$ belongs to $N_v(e_i)$ in constant time. Finally, $\pi(I') < \pi(I)$, since $\xi(I') = \xi(I)$ and $\phi(I') \leq \phi(I) - |N_v(e_i)|$.

▷ Rule 2:

Condition: Every combination of tuples for v satisfies m at v.

Reduction: We initialize $I' = I$ and mark v. —*End of* RULE 2

Clearly, I' is equivalent to I. Observe that, verifying the Condition of RULE 2 at v coincides with verifying that $N_v(e_i) = \emptyset$, for each edge e_i incident to v. The fact that this can be done in constant time can be proved analogously to RULE 1. Also, $\pi(I') < \pi(I)$, since $\xi(I') = \xi(I)$ and $\phi(I') = \phi(I) - \sum_{e \in E(v)} |S(e_i)|$.

If after applying, if possible, RULE 1 and 2, vertex v remains unmarked, then there must be at least two edges incident to v whose admissible sets contain a tuple that participates in both a satisfying and an unsatisfying combination of tuples for v.

$\triangleright$ RULE 3:

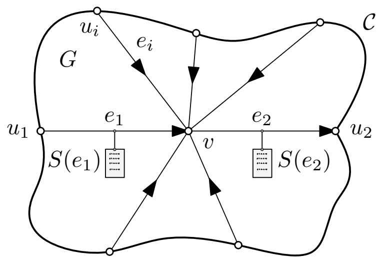
Condition: For each edge $e_i = u_i v$ incident to v, except for two edges e_1 and e_2, the variety of e_i at v is 1. That is, only e_1 and e_2 have variety at v larger than 1.

Reduction: We obtain instance I' from I as follows. For simplicity of description we will consider the edges incident to v to be oriented in the following manner: $e_1 = u_1 v$, $e_2 = v u_2$, and $e_i = u_i v$, for each edge $e_i \in E(v)$ with $i \neq \{1, 2\}$; the other orientations can be treated similarly. Refer to Fig. 2.7.

We initialize G' to the embedded directed graph obtained from G by removing v and its incident edges. We set $S'(e) = S(e)$, for each edge $e \in E(G')$, and we set $m'(u) = m(u)$, for each vertex $u \in V(G')$.

Let $\mathcal{C}$ be the cycle of G' whose boundary used to contain vertex v and its incident edges, and no other edges; refer to Fig. 2.7a.

1. For each edge e_i incident to v in G with $i \neq \{1, 2\}$, we insert a marked vertex v_i inside $\mathcal{C}$, set $m(v_i) = m(v)$, and introduce a new edge $e'_i = u_i v_i$ with $S'(e'_i) = S(e_i)$.



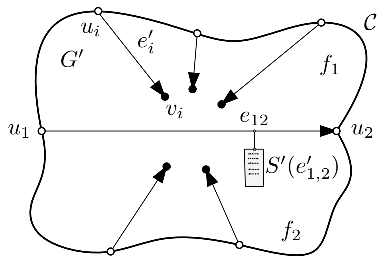


Figure 2.7: Illustration for the transformation of Rule 3. (a) Part of I in the interior of the cycle $\mathcal{C}$ containing vertex v, and (b) part of I' in the interior of cycle $\mathcal{C}$.

2. Then, we embed a new edge $e'_{1,2} = u_1u_2$ inside $\mathcal{C}$, thus splitting the interior of $\mathcal{C}$ into two faces f_1 and f_2 of G'; refer to Fig. 2.7b.
3. We add the following tuples to $S'(e'_{1,2})$. For each tuple $t_1 = \langle \sigma_1, a, \sigma_2, b\rangle \in S(e_1)$ and for each tuple $t_2 = \langle \sigma_3, c, \sigma_4, d\rangle \in S(e_2)$, we test whether the embedding pair of t_1 at v and the embedding pair of t_2 at v together with the (unique) embedding pairs at v of the remaining edges incident to v form a satisfying combination of tuples for v. If this is the case, then we add tuple $\langle \sigma_1, a, \sigma_4, d\rangle$ to $S'(e'_{1,2})$.
4. We set $S'(e'_{1,2})$ to its gist, by removing all dominated tuples.

—*End of* Rule 3

The reduction of Rule 3 can clearly be performed in constant time, since v has bounded degree and since $|S(e_1)|, |S(e_2)| \in O(1)$, by Property 2.1. Also, since vertices v_i, with $i \neq 1, 2$, are marked, $\phi(I') \leq \phi(I) - \sum_{e_i \in E(v)} |S(e_i)| + 2|S(e_{1,2})|$. The subtractive term of the previous formula is due to the fact $e_1, e_2 \notin E(G')$ and that each edge u_iv in G, with $i \neq \{1, 2\}$, corresponds to an edge u_iv_i in G' and v_i is marked. The last additive term instead is due to the fact that, if both u_1 and u_2 are unmarked in I (and hence in I'), then they both contribute $|S(e_{1,2})|$ to $\phi(I')$. Further, since $|E(G')| = |E(G)| - 1$, we have $\xi(I') = \xi(I) - 32$. Finally, $|S'(e'_{1,2})| \leq 16$, by (the proof of) Property 2.1. Thus, $\pi(I') < \pi(I)$. Next, we now show the correctness of Rule 3.

Lemma 2.11. *Let I' be the instance obtained from instance $I = \langle G, \mathcal{S}, m \rangle$ by applying the transformation of* Rule *3 to some vertex v of G. Then, instances I' and I are equivalent.*

Proof. ($I \Rightarrow I'$) Suppose that I is a positive instance of 4-MaxSkelModality, that is, there exist tuples $t(e) \in S(e)$, for each edge e in $E(G)$, that satisfy m at every vertex in $V(G)$. We show that I' is also a positive instance, by selecting a tuple $t'(e) \in S'(e)$, for each edge e in $E(G')$, such that m' is satisfied at every vertex in $V(G')$.

First, for each edge $e \in E(G) \cap E(G')$, we set $t'(e) = t(e)$. Observe that, this selection of tuples already satisfies m' at every vertex of $V(G')$ different from u_i, where u_i is a neighbor of v in G. Then, for each edge $u_iv \in E(G)$, with $i \neq 1, 2$, we set $t'(u_iv_i) = t(u_iv)$. It is easy to see that the constructed assignment satisfies now m' at each vertex $u_i \in V(G')$, with $i \neq 1, 2$, and that m' is trivially satisfied at v_i, with $i \neq 1, 2$, since each of these vertices is only incident to edge u_iv_i. Thus, we only need to show that there exists a tuple in $S'(e'_{1,2})$ for edge $e'_{1,2}$ that allows us to satisfy m' at u_1 and u_2. We set $t'(e'_{1,2}) = \langle \sigma_1, a, \sigma_3, c \rangle$, if $t_1 = \langle \sigma_1, a, \cdot, \cdot \rangle$ and $t_2 = \langle \cdot, \cdot, \sigma_4, d \rangle$ are the tuples of u_1v and vu_2 in the solution of I. Note that, tuple $t'(e'_{1,2}) \in S'(e'_{1,2})$, by construction, since t_1 and t_2 appear in a valid solution for I (and thus in a satisfying combination of tuples for v in I). To see that m' is satisfied at u_1 and u_2, it is sufficient to observe that (i) the embedding pair of edge u_1v at u_1 and the embedding pair of edge vu_2 at u_2 in I are the same as the embedding pair of edge u_1u_2 at u_1 and at u_2, respectively, in I', and that (ii) all the remaining edges incident to u_1 and to u_2 belong to both G and to G' and therefore contribute with the same embedding pair at u_1 and u_2 in I and I'.

($I' \Rightarrow I$) Suppose now that I' is a positive instance of 4-MaxSkelModality, that is, there exists tuples $t'(e) \in S'(e)$, for every e in $E(G')$, that satisfy m' at each vertex in $V(G')$. We show that I is also a positive instance, by selecting a tuple $t(e) \in S(e)$, for each edge e in $E(G)$, such that m is satisfied at each vertex in $V(G)$.

First, for each edge $e \in E(G) \cap E(G')$, we set $t(e) = t'(e)$. Similarly to the previous direction, this selection of tuples satisfies m at every vertex in $V(G)$ different from v or not adjacent to v. Then, we set $t(u_iv) = t'(u_iv_i)$, if $i \neq 1, 2$. It is easy to see that the constructed assignment satisfies now m at each vertex $u_i \neq v \in V(G)$, with $i \neq 1, 2$. Observe that, u_1v and vu_2 are the only remaining edges whose tuples have not been selected and that tuples $t(u_iv)$, with $i \neq 1, 2$, determine constraints on such a selection. Nevertheless, for each $i \neq 1, 2$, all the tuples in $S(u_iv)$ impose the same embedding constrains at v, as each of them contributes with a unique embedding pair (σ_i, a_i) at v. Observe that, for each tuple $t'(e'_{1,2}) = \langle \sigma_1, a, \sigma_3, c \rangle$ in $S'(e'_{1,2})$, there exists (at least) a pair of tuples $t_1 \in S(u_1v)$ and $t_2 S(vu_2)$ such that $t_1 = \langle \sigma_1, a, \sigma_2, b \rangle$ and $t_2 = \langle \sigma_3, c, \sigma_4, d \rangle$ satisfy m at v together with pairs (σ_i, a_i), with $i \neq 1, 2$. It follows that, by setting $t(u_1v) = t_1$ and $t(vu_2) = t_2$ we satisfy m at v, by the definition of $S'(e'_{1,2})$. Also, m is satisfied at u_1 and at u_2 since the embedding pair of edge u_1v at u_1 and the embedding pair of edge vu_2 at u_2 in I is the same as the embedding pair of edge u_1u_2 at u_1 and at u_2 in I', respectively, and since all the remaining edges incident to u_1 and to u_2 belong to both G and to G' and therefore contribute with the same embedding pair at u_1 and u_2 in I and I'. This concludes the proof of the lemma. □

We remark that each instance I' obtained from I by applying any of the reductions of Rules 1, 2, and 3 is also a good instance. This is trivial for Rules 1 and 2. In fact, for every edge $e \in G'$ ($e \in G$), we either have that $S'(e) \subset S(e)$ (Rule 1) or $S'(e) = S(e)$ (Rule 2). As for Rule 3, this follows from the fact that (i) $A'_{v_i}(u_iv_i) = A_v(e_i)$, for every edge $e_i = u_iv$ with $i \neq 1, 2$, and (ii) $A'_{u_1}(e_{1,2}) = A_{u_1}(e_1)$ and $A'_{u_2}(e_{1,2}) = A_{u_2}(e_2)$; where $A'_x(e)$ denotes the maximum number of alternations at x in a tuple in $S'(e)$, with e being an edge incident to x.

Let $I^* = \langle G^*, \mathcal{S}^*, m^* \rangle$ be the good instance, equivalent to I, produced by applying a maximal sequence of reduction rules to I. We say that I^* is *irreducible*. The fact that (i) each of the transformations of Rules 1, 2, and 3 can be performed in constant time and decreases $\pi(I)$ and that (ii) $\pi(I) \in O(|G|)$ immediately imply that we can construct instance I^* in

$O(|G|^2)$ time. However, by processing the vertices in a guided way as described in the listing of Algorithm 2, we can achieve an $O(|G|)$-time speed-up. The key idea here is that of maintaining a queue that contains the unmarked neighbors of vertices to which rules that modify the structure of the instance have been previously applied, namely, RULES 1 and 3 (Lines $12 - 14$ of the algorithm). Only when the queue is emptied and there exist unvisited (unmarked) vertices, then an unvisited vertex is added to the queue (Line 9). The running time is thus bounded by the number of times the queue is filled with $O(1)$ new vertices (Line 9 and 14); recall that $|N(v)| \in O(1)$ for each vertex v at each step of the algorithm. Since, the instruction at Line 9 is performed at most $|V(G)|$ times and the instruction at Line 14 is performed at most $\pi(I)$ times, we get the claimed linear speed-up. We formalize this in the following.

Lemma 2.12. *Given a good instance $I = \langle G, \mathcal{S}, m \rangle$, Algorithm 2 either detects that I is negative or returns an irreducible good instance $I^* = \langle G^*, \mathcal{S}^*, m^* \rangle$ in $O(|G|)$ time equivalent to I.*

The following lemma will prove useful.

Lemma 2.13. *For each unmarked vertex $v \in V(G^*)$, it holds that: (i) v has degree 3, (ii) $m^*(v) = 4$, and (iii) there exist tuples $t_1, t_2 \in S^*(e)$ such that the embedding pair of t_1 and of t_2 at v are $(\overset{\circ}{\uparrow}, 1)$ and $(\underset{\circ}{\uparrow}, 1)$, respectively, for each edge e incident to v.*

Proof. Let v be an unmarked vertex of G^*. Since I^* is irreducible, none of the conditions of Rules 1, 2, and 3 apply at v. We denote by $A^*_x(e)$ the maximum number of alternations of the embedding tuples of $S^*(e)$, where x is a vertex incident to the edge e.

We first show that $|E^*(v)| = 3$ by contradiction. Suppose first that $A^*_v(e) = 0$ for all the edges incident in $E^*(v)$, then the variety of these edges at v is 1 and there is only one combination of tuples at v. Therefore, either RULE 1 would have rejected the instance, or RULE 2 would have marked v, contradicting to the fact that I^* is irreducible. Suppose now that $E^*(v)$

contains exactly one edge e with $A^*_v(e) > 0$, then each tuple of $S^*(e)$ participates in a single combination. It follows that either one of the combinations is unsatisfying and Rule 1 would have applied, or they are all satisfying and Rule 2 would have marked v, contradicting to the fact that I^* is irreducible. Finally, suppose that $E^*(v)$ contains exactly two edges e_1 and e_2 with $A^*_v(e_1) > 0$ and $A^*_v(e_2) > 0$, then the rest of the edges of $E^*(v)$ have variety 1 at v so Rule 3 would have applied. As I^* is irreducible, however, this yields a contradiction. Therefore, $E^*(v)$ must contains at least 3 edges e_1, e_2, e_3 with $A^*_v(e_1), A^*_v(e_2), A^*_v(e_3) \geq 1$. Furthermore, since I^* is a good instance $\sum_{e \in E^*(v)}(A^*_v(e) + 1) \leq 6$, so there can be at most 3 such edges, proving Property (i) of the statement.

The arguments above also imply that $A^*_v(e_1) = A^*_v(e_2) = A^*_v(e_3) = 1$. This, there must be at least 3 alternations at v, and since v is still unmarked and the maximum-modality is even, $m^*(v)$ must be 4, proving Property (ii) of the statement.

Finally, we prove Property (iii) of the statement. First, using the same arguments used to prove Property (i), we have that if all or all but one of these edges have variety 1 at v then Rule 1 or Rule 2 apply. Likewise, if exactly one of these edges has variety 1 at v, then the other two edges have variety greater than 1 at v and Rule 3 applies. Thus, all three edges must have a variety of at least 2 at v. In fact, as they contribute with exactly one alternation at v, by Property (ii), they must have variety exactly 2. Since $(\overset{\circ}{\uparrow}, 1)$ and $(\underset{\circ}{\uparrow}, 1)$ are the only two embedding pairs with a single alternation, this proves Property (iii). □

Our next and final tool is the following, quite surprising, result.

Lemma 2.14. *Any irreducible good instance I^* is a positive instance.*

Theorem 2.9 immediately follows from Lemma 2.14. Section 2.7 presents a key result on the NAE-satisfiability problem of certain CNF-formulas, which we then exploit in Section 2.8 where we prove Lemma 2.14.

2.7 NAE-Satisfiability of CNF($\leq$2)-Formulas

A *literal* is either a boolean variable x (*positive literal*) or a negated boolean variable $\overline{x}$ (*negative literal*). A *CNF formula* is a propositional formula consisting of a conjunction of *clauses*, i.e., disjunctions of literals. For the sake of succinctness, we also denote each formula $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_k$ by the set $\phi = \{c_1, c_2, \ldots, c_k\}$ of its clauses, and each clause $c_j = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_h$ by the set $c_j = \{\ell_1, \ldots, \ell_h\}$ of its literals.

An instance of the Not-all-Equal Satisfiability (NAESAT) problem consists of a CNF formula $\phi = \{c_1, \ldots, c_k\}$ defined on the set $\mathcal{X}_\phi = \{x_1, \ldots, x_n\}$ of variables. The problem asks for the existence of a NAE-*truth assignment* for ϕ, i.e., a truth assignment for the variables in $\mathcal{X}_\phi$ such that each clause in ϕ contains both a true and a false literal. If such an assignment exists, then we say that the formula ϕ is NAE-*satisfiable*. The NAESAT problem is known to be NP-complete [90] even when each clause contains at most three literals (NAE-3-SAT).

The *variableclause graph* G_ϕ of a CNF formula ϕ is the undirected bipartite graph whose vertices are the variables in $\mathcal{X}_\phi$ and the clauses of ϕ, and whose edges represent the membership of a variable in a clause; see, e.g., Fig. 2.8. Clearly, G_ϕ has size linear in the size of ϕ. We denote the set of variables and clauses of G_ϕ by $Var(G_\phi) = \mathcal{X}_\phi$ and by $Cl(G_\phi)$, respectively. The Planar NAESAT problem is the restriction of NAESAT to instances whose variableclause graph is planar. Planar NAESAT can be solved efficiently[3], by means of a linear-time reduction to the MaxCut problem in planar graphs [82], for which an $O(n^{1.5} \log n)$ algorithm exists [91]. We denote a NAE-truth assignment for the variables of G_ϕ by means of a function $A : Var(G_\phi) \rightarrow \{\texttt{true}, \texttt{false}\}$, and the opposite NAE-truth assignment by the function $\overline{A} : Var(G_\phi) \rightarrow \{\texttt{true}, \texttt{false}\}$, where $\overline{A}(x) = \neg A(x)$, for each variable $x \in Var(G_\phi)$.

[3]In [82] Moret presented a reduction for instances of Planar NAESAT in which each clause contains exactly three literals (Planar NAE-3-SAT). However, the same reduction also implies the existence of a polynomial-time algorithm for general instances of Planar NAESAT with the same running time, as a linear-time reduction from NAE-3-SAT to NAESAT, which also preserves the planarity of the variable-clause graph, is known [93].

Observe that, if A is a NAE-truth assignment, then so is $\overline{A}$. We now present two simple observations concerning the variable-clause graph G_ϕ, which will be exploited in the remainder.

Observation 2.3. *A CNF-formula ϕ is* NAE-*satisfiable if and only if each of the CNF-formulas corresponding to the connected components of G_ϕ is* NAE-*satisfiable.*

Observation 2.3 will allow us, in some of the proofs of the technical lemmas that follow, to assume that graph G_ϕ is connected.

Observation 2.4. *A clause containing both the positive literal x and the negative literal $\overline{x}$, for some variable $x \in Var(G_\phi)$, is always* NAE-*satisfiable.*

By Observation 2.4, in the following we will always consider formulas ϕ whose clauses contain literals corresponding to different variables. In particular, this implies that G_ϕ is **simple**.

Using the notation of [86], we say that a CNF formula ϕ is a *$CNF(2)$-formula* if each variable appears in exactly two clauses of ϕ, i.e., $\deg(x) = 2$, for each variable $x \in Var(G_\phi)$; similarly, a CNF formula ϕ is a *$CNF(\leq 2)$-formula* if each variable appears in at most two clauses of ϕ, i.e., $\deg(x) \leq 2$, for each variable $x \in Var(G_\phi)$. Further, we say that a CNF formula ϕ is a *$CNF^*(\leq 2)$-formula* if each variable appears in at most two clauses of ϕ and there exists at least a *free variable*, that is, a variable that appears (negated or unnegated) in a single clause. Clearly, every $CNF(\leq 2)$-formula is either a $CNF(2)$-formula or a $CNF^*(\leq 2)$-formula.

The next observation immediately follows from the fact that the variable-clause graph G_ϕ of a $CNF(2)$-formula ϕ has minimum degree $\delta(G_\phi) = 2$.

Observation 2.5. *The variable-clause graph G_ϕ of a $CNF(2)$-formula ϕ contains at least a non-trivial block; variable and clause vertices are yellow and blue circles, respectively.*

It is easy to see that there exist $CNF(2)$-formulas that do not admit any NAE-truth assignment even when G_ϕ is a cycle. For example, the simple formula $\phi = \{c_1 = \{x, y\}, c_2 =$

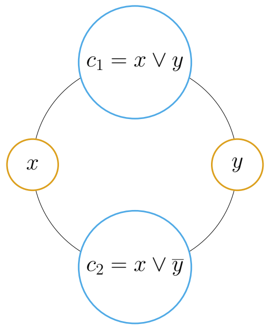


Figure 2.8: The variable-clause graph of a $CNF(2)$-formula that is not NAE-satisfiable.

$\{\overline{y}, x\}\}$ whose variable-clause graph G_ϕ is the cycle (x, c_1, y, c_2, x) illustrated in Fig. 2.8 is not NAE-satisfiable. However, for this special class of instances, once a truth assignment for a variable has been decided, then the decision for all the remaining variables is completely fixed and, in fact, in this case NAESAT can be trivially decided in linear time. In the remainder of the section, we prove the rather surprising result that the only $CNF(2)$-formulas that may not admit a NAE-truth assignment are exactly those whose variable-clause graph contains at least a connected component isomorphic to a simple cycle. Our result improves on the result of Porschen et al. [86], who presented a previous linear-time algorithm to test whether a $CNF(2)$*-formula* is NAE-satisfiable.

The rest of the section is devoted to the proof of the following main result.

Theorem 2.10. *Given a* $CNF(\leq 2)$ *formula* ϕ*, it is possible to compute a* NAE*-truth assignment for* ϕ*, if any, in linear time. Also, if* G_ϕ *does not contains a component isomorphic to a simple cycle, then* ϕ *is always* NAE*-satisfiable*

Concerning Theorem 2.10, we remark that when G_ϕ is connected, than G_ϕ may contain a cycle component only if ϕ is a $CNF(2)$ formula.

2.7.1 Proof of Theorem 2.10

Let ϕ be a $CNF(\leq 2)$-formula. Without loss of generality, by Observation 2.3, we are going to assume that G_ϕ is connected. We have that ϕ is either a $CNF^*(\leq 2)$-formula or a $CNF(2)$-formula. In the former case, we show in Lemma 2.15 that ϕ always admits a NAE-truth assignment, which can also be computed in linear time. In the latter case, we further distinguish two cases based on whether the variable-clause graph G_ϕ is a cycle or not. If G_ϕ is a cycle, then we can easily verify whether ϕ admits a NAE-truth assignment in linear time and compute it, if any, as discussed before. If G_ϕ is a not a cycle, then let β^* be a non-trivial block of G_ϕ, which exists by Observation 2.5. Also, let c^* be a clause in β^* such that $\deg(c^*) \geq 3$, which exists since G_ϕ is not a cycle. Let $\phi^- = \phi \setminus c^*$. Clearly, ϕ^- is $CNF^*(\leq 2)$-formula, as all the variables incident to c^* are free in ϕ^-. Therefore, by Lemma 2.15, ϕ^- admits a NAE-truth assignment, which can be computed in linear time. Finally, in Lemma 2.16, we show that if ϕ^- admits a NAE-truth assignment, than so does ϕ and that a NAE-truth assignment for ϕ can be computed in linear time given any NAE-truth assignment for ϕ^-. This concludes the proof of the theorem.

We now present the two lemmas for the proof of Theorem 2.10.

Lemma 2.15. *Let ϕ be a $CNF^*(\leq 2)$-formula whose variable-clause graph G_ϕ is connected. Then, a* NAE*-truth assignment for ϕ always exists and can be computed in linear time.*

Proof. We will prove the statement by *strong induction* on the number $n = |Cl(G_\phi)|$ of clauses in ϕ. At the end of the proof, we will discuss how the proof can be used to define an algorithm that computes a NAE-truth assignment for ϕ in linear time.

In the following, for a literal l, we denote by $var(l)$ the corresponding variable.

Base case. If $n = 1$, then $\phi = \{c\}$ is a CNF formula containing a single clause c; necessarily, clause c contains at least two literals and all the variables corresponding to the literals in

c are free. Let $l^* \in c$ be a distinguished literal and assume, without loss of generality, that l^* is a positive literal, that is, $l^* = var(l^*)$. We write $c = (l^* \vee l_1 \vee l_2 \vee \cdots \vee l_k)$ with $k \geq 1$. We define a NAE-truth assignment A for ϕ as follows. For $i = 1, \ldots, k$, we set $A(var(l_i))$ to either true or false arbitrarily. Then, if the clause $\phi' = (l_1 \vee l_2 \vee \cdots \vee l_k)$ is satisfied by A, we set $A(var(l_i)) = \texttt{false}$, otherwise we set $A(var(l_i)) = \texttt{true}$. Thus, by construction, clause c contains both a true and a false literal, and thus it is NAE-satisfied.

Inductive case. Let ϕ be a $CNF^*(\leq 2)$-formula with $n+1$ clauses. Suppose that *every* $CNF^*(\leq 2)$-formula with n clauses is NAE-satisfiable. We will show that ϕ is also NAE-satisfiable.

Let x be a free variable of ϕ, and let $c \in \phi$ be the clause containing a literal l^* corresponding to x. As in the base case, without loss of generality, we assume that l^* is a positive literal. Let $\phi' = \phi - \{c\}$. Since x is free in ϕ, there exists no clause in ϕ' containing either the positive literal x or the negated literal $\overline{x}$.

The variable-clause graph $G_{\phi'}$ of ϕ' consist of one or more connected components $G_{\phi'_1}$, $G_{\phi'_2}$, $\ldots$, $G_{\phi'_h}$, with $h \geq 1$, corresponding to clauses $\phi'_1, \phi'_2, \ldots, \phi'_h$, respectively, each containing at most n clauses.

We have that each clause ϕ'_i contains at least a free variable, for $i = 1, \ldots, h$. In fact, all the variables that are incident to c and to a clause of ϕ'_i G_ϕ are incident to exactly a clause in ϕ'_i, and therefore such variables are free in ϕ'_i; refer to Fig. 2.9.

Let A'_i be a NAE-truth assignment for ϕ'_i, which exists, by our inductive hypothesis, since ϕ'_i contains at least a free variable and at most n clauses. We construct a truth assignment A' for ϕ^- as follows. For each variable $y \in Var(G_{\phi'})$, we set $A'(y) = A'_i(y)$, if $v \in Var(G_{\phi'_i})$. Note that, this uniquely determines a truth assignment for all the variables of ϕ', since $Var(G_{\phi'}) = \bigcup_{i=1}^{h} Var(G_{\phi'_i})$ and since $Var(G_{\phi'_i}) \cap Var(G_{\phi'_j}) = \emptyset$ for any $1 \leq i < j \leq h$. Clearly, A' is a NAE-truth assignment for ϕ'. However, the truth value defined by A' for

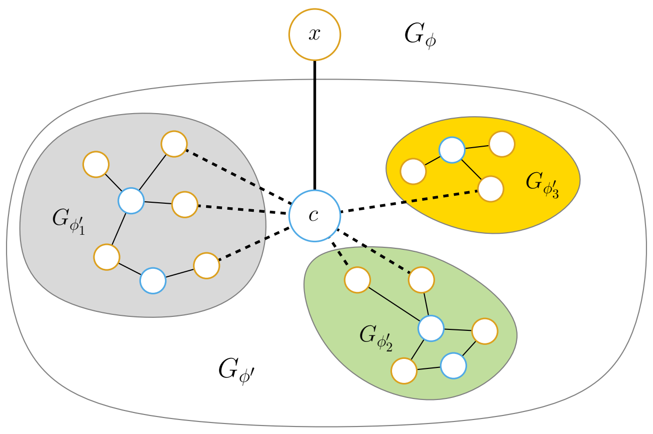


Figure 2.9: Illustration for the inductive case in the proof of Lemma 2.15; variable and clause vertices are yellow and blue circles, respectively. The graph $G_{\phi'}$ corresponding to the formula $\phi' = \phi \setminus \{c\}$ consists of three connected components $G_{\phi'_1}$, $G_{\phi'_2}$, and $G_{\phi'_3}$. The free variables created by the removal of c are the ones incident to the dashed edges.

the literals in c different from x might be such that c contains only literals with the same truth value. Nonetheless, since $x \notin Var(\phi')$, as in the base case, we can extend A' to a NAE-truth assignment for ϕ, by appropriately assigning a truth value to x so that c is also NAE-satisfied. This concludes the proof that a NAE-truth assignment for ϕ always exists.

We conclude the proof by discussing how it can be turned into a linear-time algorithm to actually construct a NAE-truth assignment for ϕ. In the following, we denote an instance of the NAESAT problem, i.e., a CNF formula, by the corresponding variable-clause graph.

The inductive arguments described above immediately yield a divide-and-conquer recursive algorithm for this task: Find a free variable x, remove x and its unique incident clause c to split the instance G_ϕ into the smaller instances $G_{\phi'_1}, G_{\phi'_2}, \ldots, G_{\phi'_h}$, apply recursion to solve them, and finally recover a solution for G_ϕ from the solutions for the smaller instances as described in the inductive case. Clearly, all the non-recursive work can be performed in time linear in the size of G_ϕ. Therefore, this immediately gives us a quadratic-time algorithm. In order to get linear time, however, we can improve this procedure as follows. First, there is no need to actively search for free variables in each instance $G_{\phi'_i}$ on which recursion must be applied. In fact, as described in the proof of the inductive case, each instance $G_{\phi'_i}$ contains at least a free variable incident to the removed clause c and, thus, such a variable can be passed as a parameter to the recursive call. Second, removing x and c, splitting the instance, and recovering a global solution from the solutions for $G_{\phi'_1}, G_{\phi'_2}, \ldots, G_{\phi'_h}$ can clearly be done in time proportional to the degree of c in G_ϕ. The two previous arguments clarify that the non-recursive work of the algorithm can in fact be done in linear time. However, for the algorithm to be actually linear, we need one more resource. In particular, we cannot afford to compute all the $G_{\phi'_i}$'s explicitly, as this would require time linear to the size of $G_{\phi'}$. Instead, we exploit the free variables incident to c as the "entrance points" for the instances $G_{\phi'_i}$'s and keep track of which of them has already been solved recursively. To this aim, we modify the algorithm so that is also labels the vertices as `processed` whenever the instance containing them has already been dealt with, and invoke recursion only on free variables incident to c that are not labeled as `processed`. In this way, we avoid actually constructing each instance and we never apply recursion on the same instance twice. This concludes the proof. $\square$

Recall that $\phi^- = \phi \setminus c^*$. We have the following.

Lemma 2.16. *Any* NAE*-truth assignment for* ϕ^- *can be extended in linear time to a* NAE*-truth assignment for* ϕ*.*

Proof. Let $A^- : Var(G_{\phi^-}) \rightarrow \{\texttt{true}, \texttt{false}\}$ be a NAE-truth assignment for ϕ^-. We show how to construct a NAE-truth assignment $A : Var(G_\phi) \rightarrow \{\texttt{true}, \texttt{false}\}$ for ϕ. We distinguish two cases based on whether clause c^* is a cut-vertex of G_ϕ (**Case 1**) or it is not (**Case 2**).

We use the following notation. Let $H_1, \ldots, H_k$ be connected subgraphs of the variable-clause graph H_ϕ of a formula CNF ϕ. Clearly, each variable of ϕ belongs to exactly one of the H_i's, that is, it holds (i) $Var(H_\phi) = \bigcup_i^k Var(H_i)$ and (ii) $Var(H_i) \cap Var(H_j) = \emptyset$ with $1 \leq i < j \leq k$. Also, for $i = 1, \ldots, k$, let ϕ_i be the CNF formula corresponding to H_i and let $A_i : Var(H_i) \rightarrow \{\texttt{true}, \texttt{false}\}$ be a truth assignment for ϕ_i. We denote by $A = \bigcup_{i=1}^k A_i^-$

the truth assignment for ϕ obtained by setting $A(x) = A_i(x)$, for each variable $x \in Var(G_\phi)$, where i is the unique index such that $x \in Var(H_i)$.

Proof for Case 1. If c^* is a cut-vertex of G_ϕ, then G_{ϕ^-} is disconnected. Let $G_{\phi_1^-}$, $G_{\phi_2^-}$, $\ldots$, $G_{\phi_k^-}$, with $k \geq 2$, be the connected components of G_{ϕ^-} and let ϕ_i^- be the CNF formula corresponding to $G_{\phi_i^-}$, for $i = 1, \ldots, k$. Observe that, each variable in $Var(G_\phi)$ belongs to exactly one of the $G_{\phi_i^-}$'s. Let A_i^- be the truth assignment obtained by restricting A^- to the variables in $Var(G_{\phi_i^-})$, for $i = 1, \ldots, k$. Clearly, A_i^- (and $\overline{A_i^-}$) is a NAE-truth assignment for ϕ_i^-.

Let $\ell_1 \in c^*$ be a literal corresponding to a variable x_1 of $G_{\phi_1^-}$ and let $\ell_2 \in c^*$ be a literal corresponding to a variable x_2 of $G_{\phi_2^-}$. If $A_1^-(x_1)$ and $A_2^-(x_2)$ are such that ℓ_1 and ℓ_2 have different truth values, then $A = \bigcup_{i=1}^{k} A_i^-$ is a NAE-truth assignment for ϕ. Otherwise, $A = \overline{A_1^-} \cup \bigcup_{i=2}^{k} A_i^-$ is a NAE-truth assignment for ϕ.

Proof for Case 2. If c^* is not a cut-vertex of G_ϕ, then G_{ϕ^-} is connected. Let β^* be the block of G_ϕ to which c^* belongs and let $\mathcal{C} = (x_1, c_1, x_2, c_2, \ldots, x_m, c_m = c^*)$ be a cycle belonging to β^* passing through c^*, where $m + 1 = 1$, the c_i's are clauses, and the x_i's are variables. Observe that, by construction, $Var(G_{\phi^-}) = Var(G_\phi)$. We initialize $A = A^-$. If clause c^* is NAE-satisfied by A, then A is a NAE-truth assignment for G_ϕ and we are done. Otherwise, A is such that all the literals of c^* have the same truth value, say `true`. We then modify A as follows. For $i = 1, \ldots, m$, we set $prev_i = c_{i-1}$, $var_i = x_i$, and $next_i = c_i$, where $i - 1 = m$ if $i = 1$. Note that, since ϕ is a $CNF(2)$-formula, clauses $prev_i$ and $next_i$ are the only two vertices of G_ϕ the variable var_i is incident to. For $i = 1, \ldots, m$, we perform the following operations. First, if clause $prev_i$ is not NAE-satisfied, then we set $A(var_i) = \overline{A(var_i)}$. Observe that, this modification only affects the two clauses incident to var_i, that is, $prev_i$ and $next_i$. In particular, $prev_i$ is now NAE-satisfied. If $next_i$ is NAE-satisfied after this step, then we are done and A is a NAE-truth assignment for G_ϕ. Otherwise, we (increment i and) repeat the previous steps. It might be the case that all the variables of cycle $\mathcal{C}$ be changed their value, that is, we reach $i = m$. The fact that A is a NAE-truth assignment also in this case is due to the fact that, when $i = m$, the literals in $next_m = c^*$ corresponding to variables var_m and var_1 are both false, by construction, that clause c^* used to contain three or more `true` literals, and that the value of these remaining literals has not changed.

As for the running time, in **Case 1** we just need to negate the value in A_1^- of the variables in $G_{\phi_1}^-$, which takes $O(|G_{\phi_1}^-|) \in O(|G_\phi|)$ time; whereas **Case 2** requires a simple visit of $\mathcal{C}$, which takes $O(m) \in O(|\beta^*|) \in O(|G_\phi|)$ time. This concludes the proof of the lemma. $\square$

2.8 Proof of Lemma 2.14

This section is devoted to proving that an irreducible good instance I^* produced by Algorithm 2 is always a positive instance. Although we assumed that G was connected, due to the transformation done by Rule 3, G^* may now contain several disconnected components. However, if G^* is disconnected, then I^* is a positive instance if and only if each "subinstance" of I^* associated with each connected component of G^* is a positive instance. Therefore, without loss of generality, in the following we will assume that G^* is connected. Moreover, if every vertex of G^* is marked, then I^* is trivially positive. In fact, if a vertex v is marked (that is, Rule 2 has been applied at v), then any selection of a tuple from the admissible sets of the edges incident to v satisfies m^* at v. Therefore, we can construct, in linear time, a solution for I^* by arbitrarily selecting a tuple from the admissible set of each edge of G^*. Thus, in the following, we will further assume that G^* contains unmarked vertices.

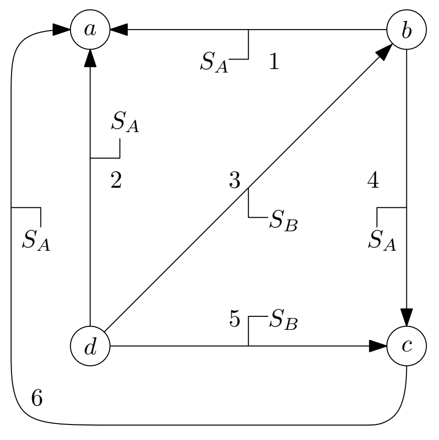
We say that a pair (t_1, t_2) of tuples in $S^*(e)$ is a *mirrored pair for* e if $t_1 = (\sigma_1, a, \sigma_2, b)$ and $t_2 = (\overline{\sigma_1}, a, \overline{\sigma_2}, b)$, with $\sigma_1, \sigma_2 \in \{⫯, ⫰\}$; observe that a and b need to be odd integers. The following observation is a consequence of Property (iii) of Lemma 2.13, if there are only two elements in the set, and of Property (iii) of Lemma 2.13 and the pigeon hole principle, if there are more.

Observation 2.6. *Let $e = uv$ be an edge of G^* whose endpoints are both unmarked. There exists a mirrored pair (t_1, t_2) for e in $S^*(e)$ such that $t_1 = (\sigma_1, 1, \sigma_2, 1)$ and $t_2 = (\overline{\sigma_1}, 1, \overline{\sigma_2}, 1)$, with $\sigma_1, \sigma_2 \in \{\overset{\circ}{\uparrow}, \underset{\circ}{\uparrow}\}$.*

We are now ready to prove Lemma 2.14. We first process instance I^*, and set $S^*(e) = \{t_1, t_2\}$ for each edge e in G^* with both endpoints unmarked, where (t_1, t_2) is a mirrored pair for e, which exists by Observation 2.6. Observe that this corresponds to arbitrarily selecting a mirrored pair for e and removing all tuples from $S^*(e)$ except for the two tuples belonging to the selected pair for e. Likewise, for each edge e with a single unmarked endpoint, we constrain its admissible set $S^*(e)$ to only two tuples with embedding pairs $(\overset{\circ}{\uparrow}, 1)$ and $(\underset{\circ}{\uparrow}, 1)$, respectively, at the unmarked endpoint of e. By Property (iii) of Lemma 2.13 such tuples must exist, and as the other endpoint of e is marked, m remains satisfiable at it. Although this might seem unwary, we will show that instance I^* is still a positive instance after the above pruning. For simplicity of notation, we continue to call such a pruned instance I^*.

2.8.1 Reduction to NAESAT

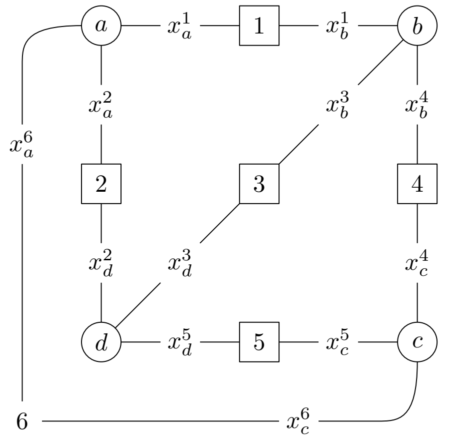
Before proceeding, we recall important properties of I^* that we are going to exploit (see also Lemma 2.12). Applying the reduction rules (and pruning) has produced a good irreducible instance $I^* = \langle G^*, \mathcal{S}^*, m^* \rangle$, with unmarked vertices (at least one, by our initial assumption) and perhaps some marked vertices. If a vertex v is marked then any combination of tuples selected from the admissible sets of its incident edges will satisfy $m^*(v)$, while if v is unmarked then it has degree 3 in G^*, $m^*(v) = 4$, and the "pruned" admissible sets of all its incident edges have each exactly 2 embedding pairs at v, namely, the two embeddings pairs $\{(\overset{\circ}{\uparrow}, 1), (\underset{\circ}{\uparrow}, 1)\}$. In particular, if an edge has two unmarked endpoints then its admissible set is a mirrored pair. We are going to show that I^* is a positive instance by creating an instance of NAESAT that



(a)

$\boxed{1}$	$x_b^1 \vee \overline{x_a^1}$	ⓐ	$x_a^1 \vee x_a^2 \vee x_a^6$
$\boxed{2}$	$x_d^2 \vee \overline{x_a^2}$	ⓑ	$\overline{x_b^1} \vee x_b^3 \vee \overline{x_b^4}$
$\boxed{3}$	$x_d^3 \vee x_b^3$	ⓒ	$x_c^4 \vee x_c^5 \vee \overline{x_c^6}$
$\boxed{4}$	$x_b^4 \vee \overline{x_c^4}$	ⓓ	$\overline{x_d^2} \vee \overline{x_d^3} \vee \overline{x_d^5}$
$\boxed{5}$	$x_d^5 \vee x_c^5$		
$\boxed{6}$	$x_c^6 \vee \overline{x_a^6}$		

(b)



(c)

$\boxed{1}$	$x_b^1 = T$	$x_a^1 = T$	$\langle \overset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1 \rangle$
$\boxed{2}$	$x_d^2 = T$	$x_a^2 = T$	$\langle \overset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1 \rangle$
$\boxed{3}$	$x_d^3 = F$	$x_b^3 = T$	$\langle \underset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1 \rangle$
$\boxed{4}$	$x_b^4 = T$	$x_c^4 = T$	$\langle \overset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1 \rangle$
$\boxed{5}$	$x_d^5 = T$	$x_c^5 = F$	$\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1 \rangle$
$\boxed{6}$	$x_c^6 = F$	$x_a^6 = F$	$\langle \underset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1 \rangle$

(d)

Figure 2.10: A complete example for the NAESAT reduction for the proof of Theorem 2.9. (a) An irreducible good instance I^*. (b) The $CNF(\leq 2)$-formula ϕ produced from I^*. (c) The variable-clause graph G_ϕ of ϕ. (d) A NAE-truth assignment for ϕ and the corresponding selection of tuples.

is satisfiable if and only if I^* is a positive instance. In particular, we are going to create CNF formulas in which each variable appears in at most two clauses (called $CNF(\leq 2)$-formulas in Section 2.8.1) whose variable-clause graph contains no component isomorphic to a simple cycle. Lemma 2.14 then follows, since such CNF formulas are always NAE-satisfiable, by Theorem 2.10. We provide a complete example for the reduction of a good instance in Fig. 2.10.

Let v be an unmarked vertex, let e be an edge incident to v, and let $p = (\sigma, 1)$ be an embedding pair of e at v. We define the *normalized embedding pair* $\hat{p}$ *for* p as $\hat{p} = (\sigma, 1)$, if e is incoming at v, or as $\hat{p} = (\overline{\sigma}, 1)$, if e is outgoing from v. This normalization allows us to disregard the orientation of the edges of G^* in what follows.

Lemma 2.17. *Let $v \in V^*$ be an unmarked vertex with incident edges e_1, e_2, e_3. Let p_1, p_2, p_3 be the embedding pairs at v of some combination of tuples $T = (t_1 \in S^*(e_1), t_2 \in S^*(e_2), t_3 \in S^*(e_3))$. Then, T is an unsatisfying combination of tuples at v if and only if $\hat{p}_1 = \hat{p}_2 = \hat{p}_3$.*

Proof. Each embedding pair contributes with one internal alternation to the modality at v. We prove the statement by arguing about the effect that the selection of tuples in T has on the total modality at v.

Since any two edges $e', e'' \in \{e_1, e_2, e_3\}$ are consecutive around v, having the same normalized embedding pair for e' and e'' introduces an additional alternation at v between e' and e''. In particular, since e_1, e_2, and e_3 are pairwise consecutive around v, if T contains tuples having the same normalized embedding pair at v, then this introduces 3 such alternations and therefore, since $m^*(v) = 4$, we have that T is an unsatisfying combination of tuples for v.

On the other hand, if T is such that $\hat{p}_1 \neq \hat{p}_2$ and $\hat{p}_1 \neq \hat{p}_3$, that is, not all the tuples in T have the same normalized embedding pair at v. Then, no additional alternation is formed by the pairs $(\hat{p}_1, \hat{p}_2)$ and $(\hat{p}_1, \hat{p}_3)$, while, as in the previous case, exactly one alternation is provided by the pair $(\hat{p}_2, \hat{p}_3)$. This concludes the proof that a combination of tuples satisfies $m^*(v)$ if and only if the normalized embedding pairs of the tuples at v are **not all equal**. $\square$

We now describe how to obtain a CNF-formula ϕ from I^* that is NAE-satisfiable if and only if I^* is a positive instance. The formula ϕ is defined on a set $\mathcal{X}$ of boolean variables. Set $\mathcal{X}$ contains a variable for each incidence between an edge and an unmarked vertex, whose two truth values are in one-to-one correspondence with the two possible embedding pairs $(\overset{\circ}{\uparrow}, 1)$ and $(\underset{\circ}{\uparrow}, 1)$ of the edge at the vertex. The formula ϕ contains two types of (NAESAT) clauses.

Edge clauses (square vertices in Fig. 2.10) ensure that the two variables associated with the two incidences of the same edge with its (unmarked) endpoints assume (simultaneously) only truth values that correctly represent one of the two tuples in the admissible set of the edge.

Vertex clauses (circle vertices in Fig. 2.10) ensure that a combination of embedding pairs at an unmarked vertex v of G^*, determined by the truth values of the variables associated with the edges incident to v, satisfies $m^*(v)$.

We construct ϕ, by processing each edge and vertex of G^* as follows. Initialize $\phi = \emptyset$.

Edges. For each edge $e = uv \in E^*$, we update ϕ as follows.

- If both u and v are marked, any tuple selected from $S(e)$ will satisfy m^* at both vertices, so we these edges do not contribute to the construction of ϕ.
- If only one of u and v is marked, say u is marked and v is unmarked, then we add a boolean variable x_v^e to $\mathcal{X}$ (associated with the incidence between e and v).
- If both u and v are unmarked, then we add two boolean variables x_u^e and x_v^e to $\mathcal{X}$. Note that, in this case, by Observation 2.6, the admissible set $S^*(e)$ of e contains a mirrored pair, that is, either $S^*(e) = S_A$ or $S^*(e) = S_B$, where $S_A = \{\langle ⫯, 1, ⫯, 1\rangle, \langle ⫰, 1, ⫰, 1\rangle\}$ and $S_B = \{\langle ⫰, 1, ⫯, 1\rangle, \langle ⫯, 1, ⫰, 1\rangle\}$. If $S^*(e) = S_A$, then we add to ϕ an edge clause $c_e = (x_u^e \vee \overline{x_v^e})$. Otherwise, if $S^*(e) = S_B$, we add to ϕ an edge clause $c_e = (x_u^e \vee x_v^e)$.

Vertices. For each vertex $v \in V^*$, we update ϕ as follows.

- If v is marked, it has no representation in ϕ, and we do nothing.
- If v is unmarked, we add to ϕ a vertex clause c_v defined as follows. Let e_1, e_2, and e_3 be the three edges of G^* incident to v. For $i = 1, 2, 3$, let x_v^i be the variable associate

with the incidence of edge e_i and v. For $i = 1, 2, 3$, we add to c_v the following literals: If e_i is incoming at v, then x_v^i appears as a positive literal in c_v; otherwise, if e_i is outgoing from v, then x_v^i appears as a negative literal in c_v. For example, if the all these edges are incoming at v then $c_v = (x_v^1 \vee x_v^2 \vee x_v^3)$, if only e_2 is outgoing from v then $c_v = (x_v^1 \vee \overline{x_v^2} \vee x_v^3)$.

This concludes the construction of ϕ, which can clearly be performed in linear time by visiting all the vertices and edges of G^*. In the next lemma, we show the correctness of the above reduction.

Lemma 2.18. *Let I^* be an irreducible good instance and let ϕ be the CNF-formula constructed from I^* as described above. Then, I^* is a positive instance if and only if ϕ is* NAE*-satisfiable.*

Proof. ($\phi \Rightarrow I^*$) Suppose that ϕ is NAE-satisfiable, then there exists a truth assignment $A : \mathcal{X} \rightarrow \{\texttt{true}, \texttt{false}\}$ for the set $\mathcal{X}$ of variables that NAE-satisfies ϕ. We show how to create a selection of tuples for each edge in E^* that satisfies m^* at every vertex starting from A as follows.

First, if both the endvertices of an edge are marked, then we select any tuple from its admissible set.

Second, for each edge $e = uv$ of G^*, we select a tuple t_e whose embedding pair at u is $(\overset{\circ}{\uparrow}, 1)$ if and only if $A(x_u^e) = \texttt{true}$ and whose embedding pair at v is $(\overset{\circ}{\uparrow}, 1)$ if and only if $A(x_v^e) = \texttt{true}$. See Fig. 2.10d for an example. We claim that such a tuple must exist. In fact, if e has only one unmarked endpoint, then Lemma 2.13 guarantees it. Otherwise, if both endpoints are unmarked, then e has a mirrored pair, by Observation 2.6, and its admissible set $S^*(e)$ corresponds to either S_A or to S_B. If $S^*(e) = S_A$, then because $c_e = (x_u^e \vee \overline{x_v^e})$ and c_e is NAE-satisfied by A, then $A(x_v^e) = A(x_u^e)$, thus tuple $t_e \in S_A$. Otherwise, if $S^*(e) = S_B$, then because $c_e = (x_u^e \vee x_v^e)$ and c_e is NAE-satisfied by A, then $A(x_v^e) \neq A(x_u^e)$, thus tuple $t_e \in S_B$.

We now show that our selection satisfies m^* at every vertex. Clearly, m^* is satisfied at every marked vertex. To show that the selection also satisfies m^* at every unmarked vertex, we prove that for each such a vertex not of all of its incident edges have been assigned tuples with the same normalized embedding pair (Lemma 2.17). By construction, a variable x_v^e appears in a variable clause c_v as a positive literal, if e is incoming to v, and as a negative literal, if e is outgoing from v. In particular, the edges incident to a vertex v have been selected the same normalized embedding pair if and only if the literals corresponding to these edges in c_v have the same truth value. Since the truth assignment NAE-satisfies c_v, one of literal truth values must be different from the other truth values, and likewise one of the normalized embedding pairs at v for the selected tuples must be different from the other normalized embedding pairs, so by Lemma 2.17, $m^*(v)$ must be satisfied. Therefore, since the selection of tuples satisfies m^* at every vertex, I^* is a positive instance.

($I^* \Rightarrow \phi$) Suppose that I^* is a positive instance, then there exists a selection of tuples that satisfies m^* at every vertex. For one such selection, we construct a truth assignment $A : \mathcal{X} \to \{\texttt{true}, \texttt{false}\}$ for ϕ as follows. For each edge e incident to an unmarked vertex v, we set $A(x_v^e) = \texttt{true}$ if and only if the embedding pair for e at v in the selection is (⫯, 1). We will show that A is a NAE-truth assignment for ϕ.

First consider the edge clauses. Let e_c be an edge clause of an edge $e = uv$ and let t_e be its selected tuple. Since the two endpoints of e are unmarked in G^*, its admissible set $S^*(e)$ is a mirrored pair, either S_A or S_B. If $S^*(e) = S_A$, then the embedding pairs of t at u and at v must be the same. Thus, in the truth assignment A, we have $A(x_u^e) = A(x_v^e)$. It follows that $c_e = (x_u^e \vee \overline{x_v^e})$ is NAE-satisfied by A. If $S^*(e) = S_B$, then the embedding tuples of t at u and at v must be different. Thus, in the truth assignment A, we have that $A(x_u^e) \neq A(x_v^e)$. It follows that $c_e = (x_u^e \vee x_v^e)$ is NAE-satisfied.

Next consider the variable clauses. For an unmarked vertex v, since the selected tuples satisfies $m^*(v) = 4$ (Lemma 2.13), we have that the normalized embedding pairs of the edges incident to v must not all be the same (Lemma 2.17). Recall that, a variable x_v^e appears in a variable clause c_v as a positive literal, if e is incoming to v, and as a negative literal, if e is outgoing from v. In particular, since v is not incident to edges with the same normalized embedding pairs, the literals corresponding to these edges do not have the same truth value in c_v , which implies that c_v is NAE-satisfied by A. This concludes the proof that A is a NAE-truth assignment for ϕ. □

Recall that the variable-clause graph G_ϕ of ϕ is the undirected bipartite graph whose vertices are the variables and clauses of ϕ and in which there is an edge between a variable vertex and a clause vertex if and only if the variable appears in the clause. Notably, the planarity of G^* implies that the variable-clause graph of the formula ϕ constructed from I^* is also planar, which in turn implies that the NAE-satisfiability of ϕ can be tested in polynomial-time by [82]. In what follows, however, we show that ϕ has the properties required by Theorem 2.10, which allow us to construct a NAE-truth assignment for ϕ and determined that I^* is a positive instance in linear time.

If I^* has marked nodes, then G_ϕ may be disconnected. By Observation 2.3, each of the connected components if G_ϕ can be handled separately, so in the following we will assume that G_ϕ is connected.

Lemma 2.19. *Let I^* be an irreducible instance, let ϕ be the CNF-formula obtained from I^* and let G_ϕ be the variable-clause graph of ϕ. Then it holds that:*

1. *ϕ is a CNF(≤ 2)-formula, that is, each variables appears in at most 2 clauses,*
2. *G_ϕ is simple, and*
3. *G_ϕ is not isomorphic to a cycle graph.*

Proof. Fist, by construction, each variable is associated with the incidence of an edge and an unmarked vertex, and appears in at most two clauses, namely, a vertex clause and, possibly, and edge clause. Therefore, formula ϕ is a $CNF(\leq 2)$ formula.

Since G^* is simple it has no loops. Thus, we have that no variable in ϕ appears twice in a vertex clause. Therefore, G_ϕ is also simple.

Finally, for a graph to be a isomorphic to a cycle graph every vertex must have degree 2. However, since there is at least one unmarked vertex in G^*, by assumption, formula ϕ has at least one vertex clause, which contains three literals. Therefore, graph G_ϕ contains at least one vertex of degree 3, and therefore it is not a cycle graph. $\square$

We have shown that finding selection of tuples that satisfies m^* for I^* can be reduced in linear time to the problem of finding a NAE-satisfying truth assignment for a $CNF(\leq 2)$-formula ϕ. Since ϕ fulfills the requirements of Theorem 2.10 it is always NAE-satisfiable. Thus, I^* is always a positive instance. We remark that this also means that we do not need to actually compute a NAE-truth assignment for ϕ to prove it is a positive instance, that is, if Algorithm 2 does not reject the instance, then such an instance is always positive.

2.9 NP-Completeness for $\mathbf{\Delta \geq k+3}$

In this section, we show that the k-Modality problem does not admit a polynomial-time algorithm for all planar digraphs with maximum degree $\Delta \geq k+3, k \geq 4$, unless $P = NP$. Clearly, the k-Modality problem is in NP, since it is possible to verify in polynomial time whether a given embedding is k-modal, by inspecting the rotation at each vertex. We first present a reduction for the k-Modality problem when $k = 4$ and $\Delta = 7$, and then extend it to $k \geq 4$ and $\Delta \geq k+3$.

In order to prove the NP-hardness, we reduce from 4-Bounded Planar 3-Connected 3-SAT, a restricted version of Planar 3-SAT which is known to be NP-complete [76].

Problem: 4-Bounded Planar 3-Connected 3-SAT

Input: A CNF-formula $\phi = \{c_1, \dots, c_m\}$ defined on a set $\mathcal{X}_\phi = \{x_1, \dots, x_n\}$ of variables, where every variable appears in at most 4 clauses and the vertex-clause graph G_ϕ of ϕ is planar and vertex 3-connected.

Question: Is there a truth assignment for the variables in $\mathcal{X}_\phi$ such that each clause in ϕ is satisfied?

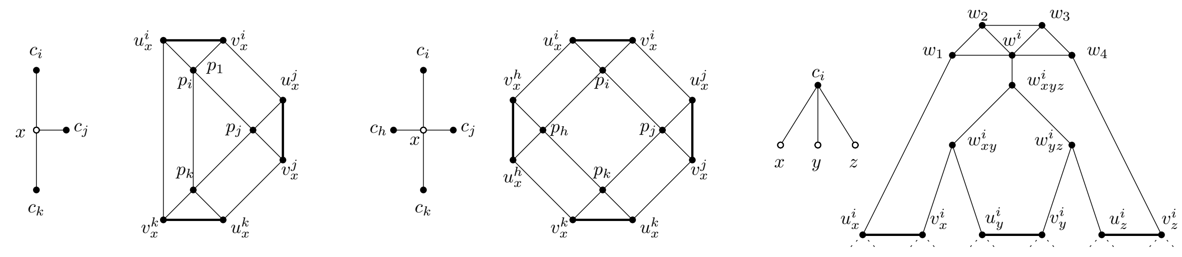
Whitney [101] proved that any 3-connected planar graph admits a unique combinatorial embedding (up to a flip). Therefore, we have the following simple observation.

Observation 2.7. *G_ϕ has a unique combinatorial embedding $\mathcal{E}_\phi$ (up to a flip).*

We show how to construct, in polynomial time, an instance $G = (V, E)$ of 4-Modality starting from the planar embedding $\mathcal{E}_\phi$ of G_ϕ in two steps. First, we transform G_ϕ into an embedded undirected graph F_ϕ, which we call the *frame* of G. Then, we replace every (undirected) edge in F_ϕ with one of several different gadgets consisting of directed edges. The resulting planar digraph G has a 4-modal embedding if and only if the formula ϕ is satisfiable. Further, our reduction generates a digraph with maximum degree $\Delta = 7$.

2.9.1 The Frame $\mathbf{F}_\phi$

We first describe how to create F_ϕ from a planar embedding $\mathcal{E}_\phi$ of the vertex-clause graph G_ϕ. Note that, since G_ϕ is vertex 3-connected and every variable appears in at most 4 clauses, it follows that every variable vertex in $Var(G_\phi)$ has degree 3 or degree 4. To obtain F_ϕ, we will exploit special plane graphs, called *variable* and *clause frames*; refer to Fig. 2.11. More formally, we construct F_ϕ in two steps as follows.



(a) (b) (c)

Figure 2.11: Replacements for the frame graph F_ϕ. (a) The variable frame for a degree-3 variable vertex. (b) FThe variable frame for a degree-4 variable vertex. (c) The clause frame for a clause vertex c incident to the variable vertices x, y, and z. The merging edges are thick.

– First, for each variable vertex $x \in Var(G_\phi)$ incident to the clause vertices c_i, c_j, and c_k, we replace x in $\mathcal{E}_\phi$ with a copy of the variable frame depicted in Fig. 2.11a; similarly, for each variable vertex $x \in Var(G_\phi)$ incident to the clause vertices c_i, c_j, c_k, and c_h, we replace x in $\mathcal{E}_\phi$ with a copy of the variable frame depicted in Fig. 2.11b. Also, for each clause vertex c_i in $Cl(G_\phi)$ incident to the variable vertices x, y, and z, we replace c_i in $\mathcal{E}_\phi$ with a copy of the clause frame depicted in Fig. 2.11c. Note that, for each (x, c_i) of G_ϕ, where x is a variable vertex and c_i is a clause vertex, the variable frame for x and the clause frame for c_i both contain the edge (u_x^i, v_x^i), called *merging edge*.

  All the variable and clause frames are embedded in the plane in a small disk centered at the variable vertex and at the clause vertex they stem from, respectively, in $\mathcal{E}_\phi$. By selecting the radii of such disks so that they do not overlap with each other, we have that all the variable and clause frames lie in the outer face of each other.

– Then, we identify the pairing merging edges in each variable and clause frame. By adhering to the planar embedding $\mathcal{E}_\phi$, the resulting embedded graph F_ϕ is also planar; refer to Fig. 2.12.

The following observations follows from Observation 2.7 and from the fact that the variable and clause frames become 3-connected if the endpoints of their merging edges are joined to a

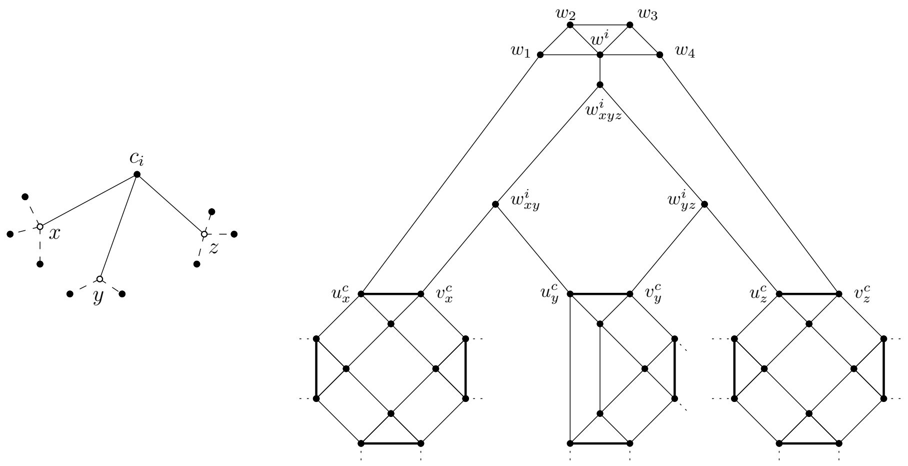


Figure 2.12: Illustration for the construction of F_ϕ from $\mathcal{E}_\phi$ focuses on the clause vertex c_i incident to the variable vertices x, y, and z. The variables x and z participate in 4 clauses, while the variable y participates in only 3 clauses.

new vertex placed in their outer face.

Observation 2.8. *F_ϕ is 3-vertex-connected and, thus, it has a unique combinatorial embedding $\mathcal{E}_F$ (up to a flip).*

2.9.2 The Graph G

After constructing F_ϕ, we transform it into the final digraph G by replacing every edge (u, v) of F_ϕ with either a directed edge or a diamond component with poles u and v. *Diamond components* are digraphs that come in two types, called D_A and D_B; each original endpoint in $V(F_\phi)$ of the edge that a diamond component replaces is a *pole* of the component.

$\mathbf{D_A}$ The edges are oriented to form a directed acyclic graph where the two non-pole vertices a and b are the only sink and source vertices, respectively; see Fig. 2.13a.

$\mathbf{D_B}$ The edges in the outer face form a directed cycle (a, u, b, v), the edge joining the vertices a and b is oriented from a to b so that a directed cycle (a, b, v, a) is formed; see Fig. 2.13b.

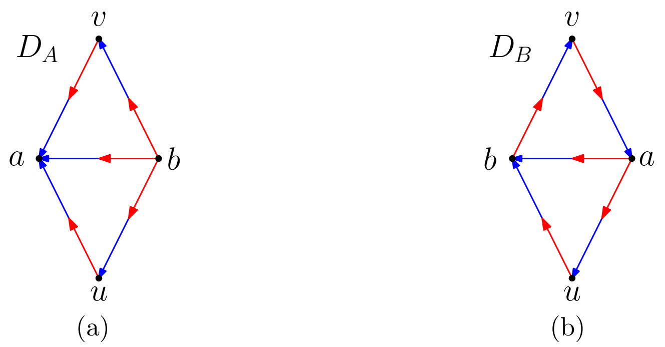


Figure 2.13: Illustrations of the diamond components. (a) The diamond component D_A; in the shown embedding of D_A, the component has a negative flip at u and a positive flip at v. (b) The diamond component D_B; ; in the shown embedding, the component has a negative flip at both u and v.

Note that each diamond component has only two possible combinatorial embeddings in which the poles lie on the outer face (this is due to the fact that adding the edge connecting the poles of a diamond components results in a 3-connected graph). These embeddings can be obtained one from the other, by a flip of the embedding.

Let $x \in \{u, v\}$ be a pole of a diamond component D, then we shall say that D has a *positive (negative) flip* at x in an embedding of G, if the incoming edge of D immediately precedes the outgoing edge of D incident to x in the clockwise (counterclockwise) ordering of these edges around x. We have the following.

Observation 2.9. *Let D be a diamond component planarly embedded such that its poles u and v lie on the outer face. Then, the following hold:*

1. *If $D = D_A$, then D has a positive flip at u if and only if it has a negative flip at v.*
2. *If $D = D_B$, then D has a positive flip at u if and only if it has a positive flip at v.*

For instance, the diamond component D_A depicted in Fig. 2.13a has a positive flip at v and a negative flip at u, while the diamond component D_B depicted in Fig. 2.13b has a negative flip at both u and v.

Next, we will exploit the diamond components to turn the variable and clause frames in F_ϕ into digraphs, called *variable* and *clause gadgets*, respectively, so to obtain the digraph G.

Variable Gadget. The *variable gadget* for a variable vertex x incident to clauses $c_1, c_2, \ldots, c_l$, $3 \leq l \leq 4$, is the digraph obtained from the variable frame for x as follows; refer to Fig. 2.14 for an example when $l = 4$.

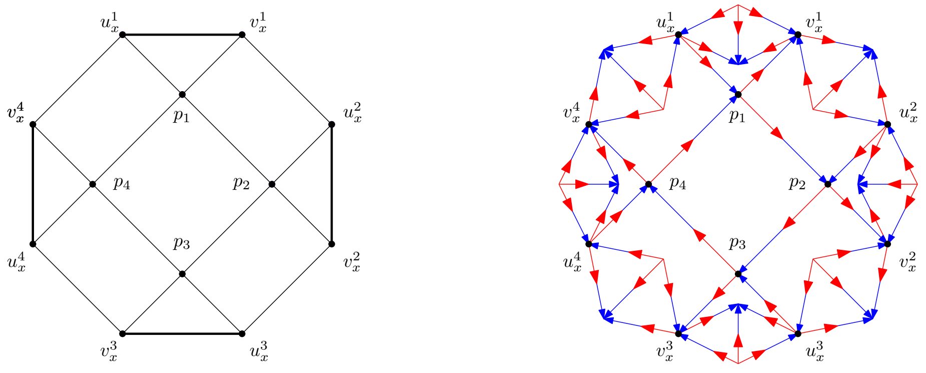


Figure 2.14: Illustration for the construction of digraph G from F_ϕ: The replacement to obtain the variable gadget (b) from the variable frame (a) in the case of a variable x incident to 4 clauses c_1, c_2, c_3, and c_4.

1. For $i = 1, \ldots, \ell$, the merging edge (u_x^i, v_x^i) is replaced with a D_A diamond component.
2. For $i = 1, \ldots, \ell$, the edge (v_x^i, u_x^{i+1}) is replaced with a D_A diamond component (where $\ell + 1 = 1$).
3. For $i = 1, \ldots, \ell$, the edge (p_i, p_{i+1}) is replaced with the directed edge $p_i p_{i+1}$ (where $\ell + 1 = 1$).
4. For $i = 1, \ldots, \ell$, the edge (u_x^i, p_i) is replaced with the directed edge $u_x^i p_i$ and the edge (v_x^i, p_i) is replaced with the directed edge $p_i v_x^i$.

The *clause gadget* for a clause $c_i = (l_x \vee l_y \vee l_z)$, where l_x, l_y, and l_z are literals for the variables

x, y, and z, respectively, is the digraph obtained from the clause frame for c_i as follow; refer to Fig. 2.15.

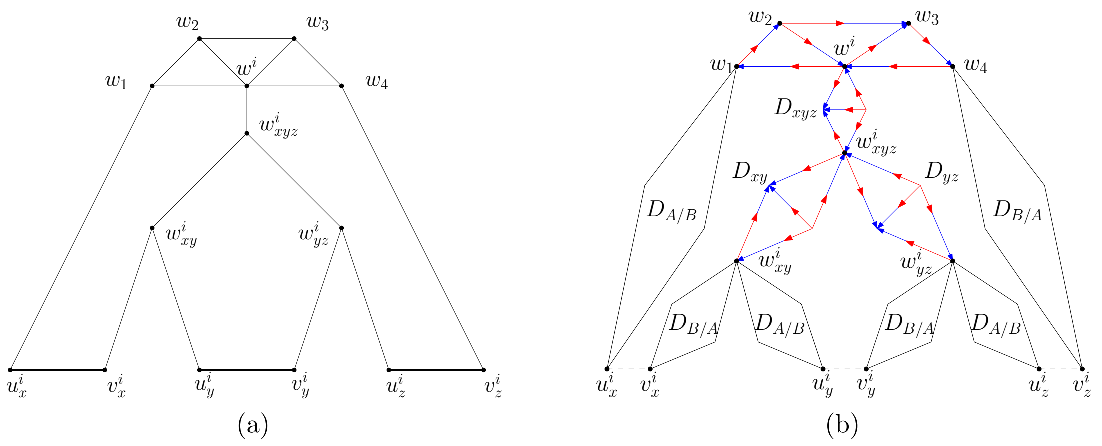


Figure 2.15: Illustration for the construction of digraph G from F_ϕ: The replacement to obtain the clause gadget (b) from the clause frame (a) for a clause c_i containing literals corresponding to the variables x, y, and z. Diamond components labeled with $D_{A/B}$ and $D_{B/A}$ may be either D_A or D_B components, depending on whether the variable to whose merging edge the components are incident to appears as a positive or a negative literal in c_i.

1. Replace the edges (w^i_{xy}, w^i_{xyz}), (w^i_{yz}, w^i_{xyz}), and (w^i_{xyz}, w^i) with a D_A diamond component D_{xy}, D_{yz}, and D_{xyz}, respectively;
2. Replace the edge (w_1, w^i) with the directed edge $w^i w_1$, the edge (w_2, w^i) with the directed edge $w_2 w^i$, the edge (w_3, w^i) with the directed edge $w^i w_3$, and the edge (w_4, w^i) with the directed edge $w_4 w^i$.
3. Replace the path (w_1, w_2, w_3, w_4) with the directed path (w_1, w_2, w_3, w_4)
4. If l_x is a positive (negative) literal, then replace the edge (u^i_x, w_1) with a D_A (D_B) diamond component and replace the edge (v^i_x, w^i_{xy}) with a D_B (D_A) diamond component.
5. If l_y is a positive (negative) literal, then replace the edge (u^i_y, w^i_{xy}) with a D_A (D_B) diamond component, and replace the edge (v^i_y, w^i_{yz}) with a D_B (D_A) diamond component.
6. If l_z is a positive (negative) literal, then replace the edge (u^i_z, w^i_{yz}) with a D_A (D_B) diamond component, and replace the edge (v^i_z, w_4) with a D_B (D_A) diamond component.

The construction of the digraph G described above can be clearly performed in polynomial time. Also, G has maximum degree 7. In fact, the only degree-7 vertices are the vertices u_x^i and v_x^i of the variable gadgets, for each variable x participating in a clause c_i. Also, all the vertices w^i, w^i_{xyz}, w^i_{xy}, and w^i_{yz}, for each clause c_i defined on literals for the variables x, y, and z, have degree 6. Since all the remaining vertices of G have degree smaller than or equal to 5, it follows from Observation 2.1 that the modality at these vertices is always at most 4. Therefore, the only embedding choices in a 4-modal embedding of G are those that happen at the vertices of degree greater than or equal to 6 listed above.

In the following, we show that G admits a 4-modal embedding if and only if ϕ is satisfiable.

Since F_ϕ is 3-connected and since we replaced each edge of F_ϕ with either a directed edge or a diamond component (which becomes 3-connected, if the poles are joined by an edge), we can easily see that G is biconnected and that the SPQR-tree $\mathcal{T}$ of G (considered as undirected) is, in fact, an RQ-tree $\mathcal{T}$. In particular, $\mathcal{T}$ contains an R-node whose skeleton is F_ϕ and it contains an R-node, for each diamond component, where both the skeleton and the pertinent graph of these R-nodes are isomorphic to the corresponding diamond component. Also, the pertinent graph of each D_A diamond component admits the embedding tuples $S_A = \{\langle ⫯, 1, ⫯, 1\rangle, \langle ⫰, 1, ⫰, 1\rangle\}$, while the pertinent graph of each D_B diamond component admits the embedding tuples $S_B = \{\langle ⫰, 1, ⫯, 1\rangle, \langle ⫯, 1, ⫰, 1\rangle\}$. (Note that, since these embedding tuples are symmetric, we do not need to assign an orientation to the virtual edges of the nodes of $\mathcal{T}$ for these tuples to be well-defined). Furthermore, fixing the flip of the R-node whose skeleton is F_ϕ does not affect the possibility that G admits a 4-modal embedding (as the flip of a 4-modal embedding is still 4-modal). Therefore, we have the following.

Observation 2.10. *The only possible embedding choices for G are the flips of its diamond components.*

By Observation 2.10, the modality of each vertex in an embedding of G is only determined by the flip of the diamond components incident to it.

Next, we prove two key lemmas concerning the embeddings of the variable gadgets and of the clause gadgets in a 4-modal embedding of G.

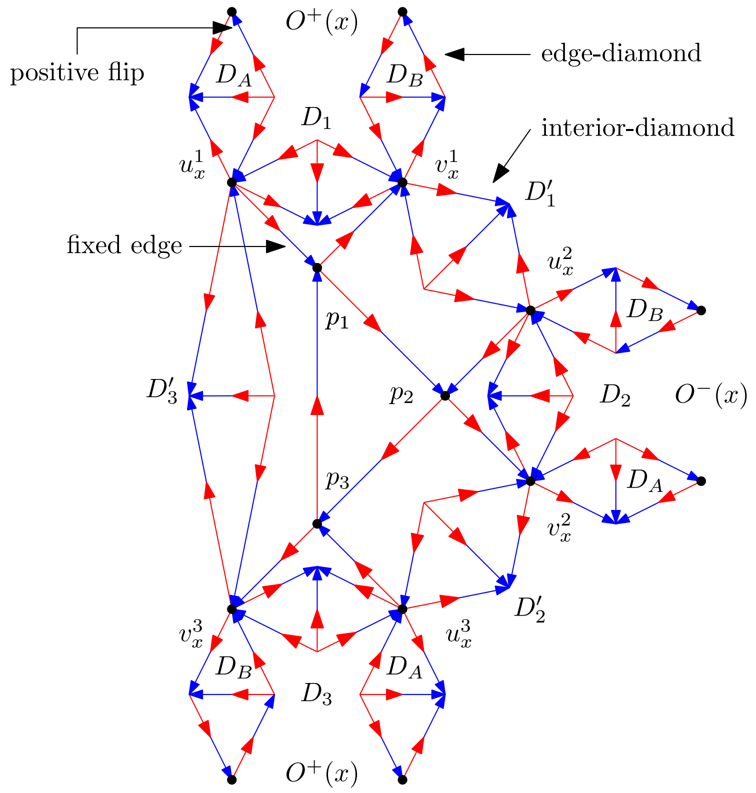


Figure 2.16: Illustration of the extended variable gadget E_x of a variable $x \in \mathcal{X}_\phi$ that participates to the three clause c_1, c_2, and c_3 of ϕ. The variable x appears as a positive literal in c_1 and c_3, and as a negative literal in c_2. The embedding of the E_x is True, as the edge-diamonds incident to the vertices u_x^1 and u_x^3 have a positive flip at their clause pole, and the edge-diamond incident to the vertex u_x^2 has a negative flip at its clause pole.

For each variable $x \in \mathcal{X}_\phi$ whose literals participate to the set of clauses $C(x) = c_1, \ldots, c_l$, $l \in \{3, 4\}$, let the *extended variable gadget* E_x of x be the subgraph of G composed of the variable gadget for x and the diamond components belonging to the clause gadgets for c_i, with $i \in \{1, 2, 3, 4\}$, that are incident to G_x; refer to Fig. 2.16. We call *edge-diamonds* of x the diamond components belonging to E_x and not to the variable gadget for x. We denote the set of edge-diamonds for x as $O(x)$. Also, the pole of each such a diamond component D belonging to the variable gadget is the *variable pole* of D, while the pole in a clause gadget is the *clause pole* of D. We define $C^+(x) \subseteq C(x)$ as the set of clauses where x appears as a positive literal and $C^-(x_i) \subseteq C(x)$ as the set of clauses where x appears as a negative literal. We define $O^+(x) \subseteq O(x)$ as the set of edge-diamonds that belong to clause gadgets for clauses in $C^+(x)$, and $O^-(x_i) \subseteq O(x_i)$ as the set of edge-diamonds that belong to clause gadgets for clauses in $C^-(x_i)$.

Let $\mathcal{H}$ be a planar embedding of G and let $\mathcal{H}_x$ be the embedding of E_x induced by $\mathcal{H}$. Note that, the embedding $\mathcal{H}_x$ is completely determined by the flips of the diamond components belonging to E_x. We say that $\mathcal{H}_x$ is *True* (*False*), if the flip of every edge-diamond in $O^+(x)$ is positive (negative) at its clause pole and the flip of every edge-diamond in $O^-(x)$ is negative (positive) at its clause pole. For instance, the embedding of the extended variable gadget depicted in Fig. 2.16 is True; in fact, the edge-diamonds incident to the vertex u_x^1 and u_x^3 have a positive flip at their clause poles and x participates as a positive literal in both c_1 and c_3, while the edge-diamond incident to the vertex u_x^2 has a negative flip at its clause pole and x participates as a negative literal in c_2.

Lemma 2.20. *Let $\mathcal{E}$ be a 4-modal embedding of G and let $\mathcal{E}_x$ be the embedding of E_x induced by $\mathcal{E}$. Then, the embedding $\mathcal{E}_x$ is either True or False.*

Proof. Consider the vertices u_x^i and v_x^i of E_x. Each of these vertices is a pole of three diamond components. Two of these diamond components are D_A diamond components also belonging to the variable gadget for x; we call them *interior-diamonds* of x. The third diamond component is either a D_A or a D_B diamond component which is an edge-diamond for x. Moreover, each vertex u_x^i and v_x^i is the endpoint of an edge whose other endpoint is p_i, we call *fixed edge* such an edge.

First, we will prove that the embedding $\mathcal{E}_x$ of E_x induced by $\mathcal{E}$ is completely determined by the choice of a flip for any of its interior-diamonds; refer to Fig. 2.16. Namely, let D_i denote the interior-diamond components with poles u_x^i and v_x^i and let D'_i denote the interior-diamond components with poles v_x^i and u_x^{i+1}, with $l+1=1$. We will show that a choice of a flip for D_i completely determines the flip of all the interior-diamonds of x and all the edge-diamonds of x.

The poles of the interior-diamond component D_i are incident to a fixed edge, either the edge $u_x^i p_i$ and the edge $p_i v_x^i$. Also, in any planar embedding of G, and thus in $\mathcal{E}$, we have that in the rotation at u_x^i all the edges belonging to the edge-diamond component incident to u_x^i are consecutive and precede the edges of D_i, all the edges of D_i are consecutive and precede the edge $u_x^i p_i$, and all the edges of D'_{i-1} are consecutive and follow the edge $u_x^i p_i$. Analogously, in $\mathcal{E}$, we have that in the rotation at v_x^i all the edges belonging to the edge-diamond component incident to v_x^i are consecutive and precede the edges of D'_i, all the edges of D'_i are consecutive and precede the edge $p_i v_x^i$, and all the edges of D_i are consecutive and follow the edge $p_i v_x^i$.

Since the fixed edges incident to each interior-diamond D have a different orientation at each pole of D and the interior-diamonds are D_A diamond components, at least one of the fixed edges must form an alternation with with an edge of D in $\mathcal{E}$. Which of the two fixed edge has an alteration, depends on the flip of D.

Suppose that D_i has a negative flip at v_x^i; the case in which D_i has a positive flip at v_x^i is symmetric. Then, the edge $p_i v_x^i$ form an alternating pair at v_x^i with an edge of D_i. Since all the three diamond components incident to v_x^i already contribute with an alternating pair of edges, we have that the flip of in a 4-modal embedding of G the flips of these diamond components is fixed. Namely, the edge-diamond incident to v_x^i has a positive flip at v_x^i and the diamond component D'_i has a negative flip at v_x^i. Since, the edge of the interior-diamond component D'_i can not form an additional alternation with the fixed edge $p_i v_x^i$, it must form a alternation with the fixed edge $p_{i+1} u_x^{i+1}$. Therefore, the embedding choice for D_i propagates throughout all the diamond components of E_x.

Altogether, we have proved that E_x admits exactly two possible 4-modal embeddings. Next, we will prove that one of them is True and that the other is False.

Let D_i be the internal diamond of x with poles u_x^i and v_x^i and let D'_i be the internal diamonds of x with poles u_x^i and u_x^{i+1}, where $l + 1 = 1$. We will prove that $\mathcal{E}_x$ is True if and only if the flip of the D_i's at u_i is positive. Since each edge-diamond is incident to at least a diamond D_i, with $i \leq l$, and since all the diamonds D_i, with $i \leq l$, have the same flip at their u_i poles, the statement follows.

By the arguments above, we have that all the diamonds D_i have the same, positive or negative flip, at their poles v_x^i, and that all the diamonds D'_i have the same, positive or negative flip, at their poles v_x^i. Also, a diamond D'_i has a positive flip at v_x^i if and only if the diamond D_i has a positive flip at v_x^i.

Suppose now that a diamond D_i has positive flip at u_x^i, and thus a negative flip at v_x^i. Then, the edge-diamond incident to v_x^i has a positive flip at v_x^i. We have the following two cases. If $c_i \in C^+(x)$, then the edge-diamond incident to v_x^i is a D_B diamond component and therefore it has a positive flip at its clause pole. Symmetrically, if $c_i \in C^-(x)$, then the edge-diamond incident to v_x^i is a D_A diamond component and therefore it has a negative flip at its clause pole. Finally, for $i = 1, \dots, l$, the edge-diamonds incident to u_i and v_i have distinct types, either D_A or D_B. This implies that such diamond components have the same flip at their clause vertex, and concludes that proof that $\mathcal{E}_x$ is either True or False. □

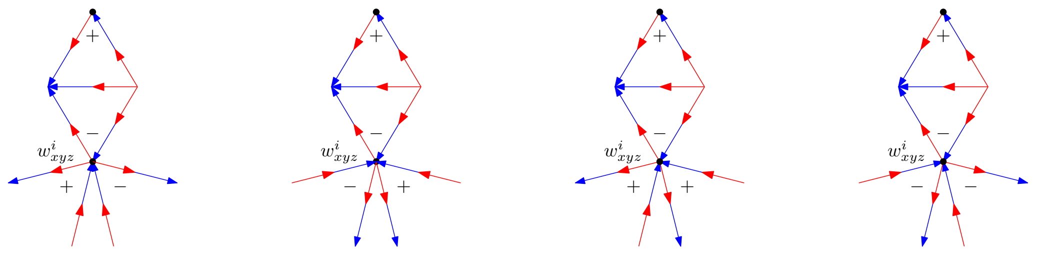
Lemma 2.21. *In any 4-modal embedding $\mathcal{E}$ of G, for each clause c_i of ϕ, the clause gadget for c_i contains at least one edge-diamond with a positive flip at w_{xy}^i or at w_{yz}^i.*

Proof. First, observe that the D_A diamond component D_{xyz} must have a positive flip at w^i in $\mathcal{E}$; see Fig. 2.15. In fact, since the rotation of the remaining edges incident to w^i is fixed, and since every two consecutive such edges form an alternating pair at w^i, a negative flip of D_{xyz} at w^i would imply a modality equal to 6 at this vertex.

However, the flip of D_{xyz} also affects the modality at w^i_{xyz}. We use this property to prove the statement in two steps.

(i) For the modality at w^i_{xyz} to be 4 when D_{xyz} has a positive flip at w^i, we shall show that at least one of D_{xy} or D_{yz} must have a positive flip at w^i_{xyz}.

(ii) Moreover, for one of D_{xy} or D_{yz} to have a positive flip at w^i_{xyz}, we shall show that at least one of the edge-diamonds incident to w^i_{xy} or to w^i_{yz} must have a positive flip at w^i_{xy} or at w^i_{yz}, respectively.



(a) $m(w) = 4$ (b) $m(w) = 4$ (c) $m(w) = 4$ (d) $m(w) = 6$

Figure 2.17: Illustration for the situation at the vertex w^i_{xyz}.

(i) The modality at w^i_{xyz} is determined by the flips of D_{xy}, D_{yz} and D_{xyz}. Since D_{xyz} has a positive flip at w^i, by hypothesis, it has a negative flip at w^i_{xyz}; see Fig. 2.17.

Note that, if D_{xy} and D_{yz} have the same flip at w^i_{xyz}, then there is an alternation at w^i_{xyz} between an edge of D_{xy} and an edge of D_{yz}. On the other hand, if they have different flips at w^i_{xyz}, then there is no alternation.

If both D_{xy} and D_{yz} have different flips at w^i_{xyz}, then one of them has to be positive, and **(i)** trivially holds. Note that, in this case, the modality at w^i_{xyz} is 4 in $\mathcal{E}$, independently of which of D_{xy} and D_{yz} has a positive flip at w^i_{xyz}; refer to Figs. 2.17a and 2.17b. This is indeed the case as D_{xy}, D_{yz}, and D_{xyz} each contribute with one pair of alternating edges at w^i_{xyz}, however no edge in D_{xy} forms an alternating pair with an edge of D_{yz} at w^i_{xyz}, and one edge of D_{xyz} incident to w^i_{xyz} forms an alternating pair at w^i_{xyz} with an edge of D_{xy} if and only if the other edge of D_{xyz} incident to w^i_{xyz} does not form an alternating pair at w^i_{xyz} with an edge of D_{yz}.

On the other hand, if both D_{xy} and D_{yz} have the same flip at w^i_{xyz}, then there are already at least 4 alternations at w^i_{xyz}. Namely, one formed by the edges of each diamond component incident to w^i_{xyz}, and one between D_{xy} and D_{yz}. Therefore, since the flip of D_{xyz} is negative at w^i_{xyz}, we have that for the modality at w^i_{xyz} to be 4, the diamond components D_{xy} and D_{yz} must both have a positive flip at w^i_{xyz}; refer to Figs. 2.17c and 2.17d.

(ii) We now show that at least one of the edge-diamonds incident to w^i_{xy} or to w^i_{yz} must have a positive flip at w^i_{xy} or at w^i_{yz}, respectively. By **(i)**, we know that at least one of D_{xy} or D_{yz} must have a positive flip at w^i_{xyz}. We show that if D_{xy} has a positive flip at w^i_{xyz}, then one of the edge-diamonds incident to w^i_{xy} must have a positive flip at w^i_{xy}; the proof that if D_{yz} has a positive flip at w^i_{xyz}, then one of the edge-diamonds incident to w^i_{yz} must have a positive flip at w^i_{yz} is symmetric. The proof is identical to the one for **(i)**; here, the diamond component D_{xy} plays the role of D_{xyz} in the proof of **(i)** and the two edge-diamonds incident to w^i_{xy} play the role of D_{xy} and D_{yz} in the proof of **(i)**. In particular, since the flip of D_{xy} at w^i_{xyz} is positive, it is negative at w^i_{xy}, and therefore a negative flip of both its two incident edge-diamonds at w^i_{xy} would imply a modality of 6 at w^i_{xy} (analogously to the case in Fig. 2.17d for **(i)**). This concludes the proof. □

Theorem 2.11. 4-Modality *is NP-complete even for biconnected digraphs of maximum degree* 7.

Proof. We have already shown that the digraph G can be constructed in polynomial time from the formula ϕ, and that G is biconnected and has maximum degree 7. We now prove that G admits a 4-modal embedding $\mathcal{E}$ if and only if ϕ is satisfiable.

($\phi \Rightarrow \mathcal{E}$) Suppose that ϕ is satisfiable, i.e., there exists a truth assignment $A : \mathcal{X}_\phi \rightarrow \{True, False\}$ for the variables in $\mathcal{X}_\phi$ satisfying each clause in ϕ. Recall that, by Observation 2.10, the only embedding choices that affect the modality of an embedding of G are the flips of its diamond components. Next, we show how to select such flips, based on the truth assignment A, so that the resulting embedding $\mathcal{E}$ of G is 4-modal.

For each variable $x \in \mathcal{X}_\phi$, we set the embedding of the extended variable gadget E_x of x to be $A(x)$ in $\mathcal{E}$, i.e., we set the embedding of E_x to be True if and only if $A(x) = True$. Thus, we have set the flips of all the diamond components of E_x. Note that, in a True or False embedding of E_x the modality of all the vertices of E_x, except for the clause poles, is at most 4; in fact, the modality at the clause poles of the edge-diamonds of E_x also depends on the embedding choices in the rest of G. The only diamond components of G whose flips have not yet been fixed are the three diamond components incident to the vertices w^i_{xyz}'s of the clause gadgets contained in G. In the following, for each clause c_i of ϕ, we show how to set the flip of its diamond components D_{xy}, D_{yz}, and D_{xyz} incident to w^i_{xyz} in such a way that the modality at w^i_{xy}, w^i_{yz}, and w^i_{xyz} is 4 in $\mathcal{E}$.

Since the truth assignment satisfies ϕ, at least one the edge-diamonds of the clause gadget for c_i has a positive flip at its corresponding clause pole. Assume, without loss of generality, that one of the two edge-diamonds incident to w^i_{xy} has a positive flip at w^i_{xy}. Then, the flip of D_{xy} can be selected so that it is negative at w^i_{xy}, while ensuring that the modality at w^i_{xy} is 4 in $\mathcal{E}$. In fact, the modality at w^i_{xy} is 4 as long as the the three diamond components incident to w^i_{xy} have not all the same flip at w^i_{xy}. Further, since the flip of D_{xy} is negative at w^i_{xy}, it is positive at w^i_{xyz}. Also, since the flip of D_{xyz} must be set to be positive at w^i, in order for the modality at w^i to be at most 4, the modality of D_{xyz} at w^i_{xyz} is negative. Therefore, since the modality at w^i_{xyz} is 4 as long as the the three diamond components incident to w^i_{xyz} have not all the same flip at w^i_{xyz}, we immediately get that (i) the modality of w^i_{xyz} is 4 in $\mathcal{E}$, independently of the flip of D_{yz}, and that (ii) we can select a flip of D_{yz} so that the modality of w^i_{yz} is 4 in $\mathcal{E}$, by simply ensuring that the flip of D_{yz} at w^i_{yz} is different from that of at least one of the two edge-diamonds incident to w^i_{yz}. This concludes the proof that the flips

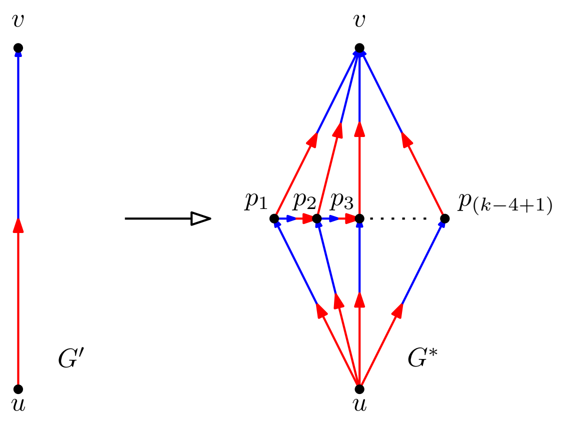


Figure 2.18: Illustration of the thickening gadget $T(u, v)$ for the proof of Theorem 2.12.

of the diamond components of G can be selected so that the resulting embedding $\mathcal{E}$ of G is 4-modal.

$(\mathcal{E} \Rightarrow \phi)$ Suppose that G admits a 4-modal embedding $\mathcal{E}$. We show how to infer from $\mathcal{E}$ a truth assignment $A : \mathcal{X}_\phi \to \{True, False\}$ for the variables $\mathcal{X}_\phi$ of ϕ that satisfies each clause of ϕ. By Lemma 2.20, the embedding $\mathcal{E}_x$ of each extended variable gadget E_x induced by $\mathcal{E}$ is either True or False, for every variable $x \in \mathcal{X}_\phi$. We set $A(x)$ to be $True$ if and only if the embedding $\mathcal{E}_x$ of E_x is True.

Next, we show that A satisfies each clause of ϕ. Let c_i be a clause of ϕ. By Lemma 2.21, at least one edge-diamond incident to either w^i_{xy} or w^i_{yz} must have a positive flip at w^i_{xy} or w^i_{yz}, respectively. Assume, without loss of generality, that an edge-diamond D incident to w^i_{xy} has a positive flip at w^i_{xy}. Let x be the variable whose extended variable gadget E_x contains the edge-diamond D. Clearly, x participates to c_i either as a positive literal or as a negative literal. In the former case, since $D \in O^+(x)$ and since w^i_{xy} is the clause pole of D, we have that $\mathcal{E}_x$ is True, and thus c_i is satisfied by $A(x) = True$. In the latter case, since $D \in O^-(x)$ and since w^i_{xy} is the clause pole of D, we have that $\mathcal{E}_x$ is False, and thus c_i is satisfied by $A(x) = False$. This concludes the proof of the theorem. □

In the following, we show how to extend the proof of Theorem 2.11 to $k > 4$ and $\Delta \geq k + 3$.

For any $k > 4$, let the *thickening gadget* $T(u, v)$ be the digraph defined as follows; refer to Fig. 2.18. The vertex set of $T(u, v)$ consists of the vertices u, v and of the vertices p_i, for $i = 1, \dots, k-4+1$. The edge set of $T(u, v)$ consists of the directed edges up_i and p_iv, for $i = 1, \dots, k-4+1$, and of the directed edges p_ip_{i+1}, for $i = 1, \dots, k-4$.

First, we perform the same reduction as in the proof of Theorem 2.11. Let G be the resulting digraph. Second, we replace each edge uv of G with the three directed edges uu', $u'v'$, and $v'v$, that is, we replace the edge with a directed path of length 3 between its endpoints. Let G' be the resulting digraph. Finally, for every vertex v in $V(G)$, we select an edge e of G' incident to it. If $e = uv$, we replace e with $T(u, v)$, otherwise we replace e with $T(v, u)$. Let G^* be the resulting digraph.

Clearly, the construction of G^* can be performed in polynomial time and G^* is biconnected (since G is biconnected). Further, $\Delta(G') = \Delta(G)$ and the thickening gadgets have degree $k - 4 + 1$ at their endpoints. Moreover, no two thickening edges are adjacent, as any two vertices of $V(G)$ have distance at least 3 in G'. Therefore, $\Delta(G^*) = \Delta(G) + k - 4 = k + 3$.

We claim that G^* admits a k-modal embedding if and only if G admits a 4-modal embedding. The key property of the thickening gadgets is that, in any k-modal embedding of G^*, they contribute with $k-4$ unavoidable alternations to the modality of the vertex of $V(G)$ they are incident to. Moreover, since the thickening gadgets are 3-connected, in any planar embedding $\mathcal{E}^*$ of G^*, the edges up_1, p_1v, $up_{(k-4+1)}$, and $p_{(k-4+1)}v$ of a thickening gadget $T(u, v)$ bound a face of the embedding $\mathcal{E}'$ of $T(u, v)$ induced by $\mathcal{E}^*$. Therefore, in $\mathcal{E}^*$, the number of alternations at a vertex v of $V(G)$ that are not those determined by pairs of edges of a thickening gadget incident to v, is independent of which of the two possible embeddings of the thickening gadget is selected (as both such embeddings offer edges with the same orientation at u and at v at their interface with the rest of the graph). Therefore, we can obtain a drawing Γ^* of G^* determining a k-modal embedding from a drawing Γ' of G' determining a 4-modal embedding, by replacing the drawing of an edge (u, v) of G' in Γ' with any planar drawing of $T(u, v)$

with u and v on its outer face. Clearly, the drawing Γ^* has maximum modality equal to k, as no two thickening gadgets are incident to a common vertex and since each thickening gadget contributes with $k-4$ pairs of alternating edges at its endvertices. Symmetrically, we can obtain a drawing Γ' of G' determining a 4-modal embedding from a drawing Γ^* of G^* determining a k-modal embedding, by replacing the drawing of a thickening gadget $T(u,v)$ of G^* in Γ^* with a drawing of the edge uv that resembles the drawing in Γ^* of the path (u, p_1, v) of $T(u,v)$. The fact that G' admits a 4-modal embedding if and only if G admits one, then implies that G^* has a k-modal embedding if and only if G has a 4-modal embedding, which in turn yields the following.

Theorem 2.12. k-Modality *is NP-complete with* $k \geq 4$ *even for biconnected digraphs of maximum degree* $\Delta \geq k+3$.

2.10 k-Modal Embeddings of Outerplanar Digraphs

In this section, we consider k-modal embeddings of outerplanar digraphs.

It is easy to see that any directed tree always admits a bimodal embedding. In fact, it suffices to observe that, when applied to trees, the simple reduction presented in Section 2.1 from the problem of testing for the existence of a bimodal embedding to the one of testing the planarity of a suitably-defined auxiliary graph, produces again trees, and that trees are planar. We thus have the following.

Fact 2.1. *Any directed tree has a bimodal embedding.*

On the other hand, we can prove that this result does not extend further to the class of outerplanar graphs, which includes the trees. For instance, Fig. 2.19 provides an example of an outerplanar graph $\mathcal{O}$, which is also simple and biconnected, that does not admit any bimodal embedding.

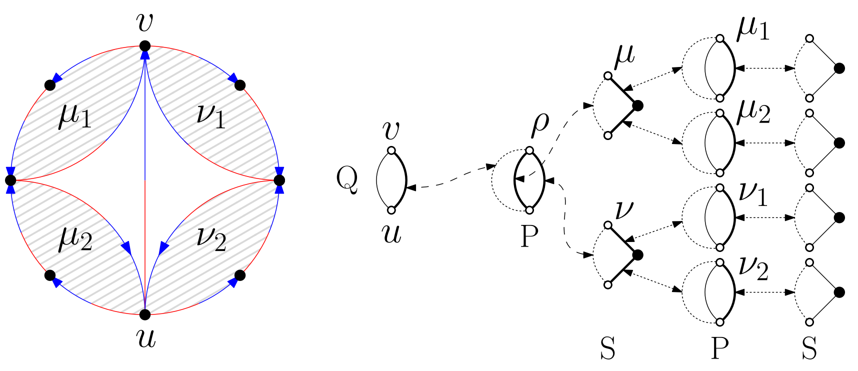


Figure 2.19: A simple biconnected outerplanar digraph $\mathcal{O}$ that does not admit a bimodal embedding.

Theorem 2.13. *There exists an infinity family $\mathcal{F}$ of simple biconnected outerplanar digraphs that are not bimodal.*

Proof. First, we prove that the digraph $\mathcal{O}$ depicted in Fig. 2.19 does not admit any bimodal embedding. Consider the SPQR-tree $\mathcal{T}$ of $\mathcal{O}$ rooted at edge (u, v). We can easily see that $S(\nu_1) = S(\nu_2) = S(\mu_1) = S(\mu_2) = \{\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle, \langle \underset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1\rangle\}$. The parent node of nodes ν_1 and ν_2 is the S-node ν. We have that $S(\nu) = \{\langle \overset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1\rangle, \langle \underset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle\}$. This is due to the fact that each good embedding of pert(ν_1) and pert(ν_2) already contributes with an alternation at their shared vertex and that, in order for such a vertex to be bimodal in an embedding of pert(ν), the good embeddings of the pertinent graphs of ν_1 and ν_2 have to be combined so not to create any additional alternation at the vertex. The proof that $S(\mu) = \{\langle \overset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1\rangle, \langle \underset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle\}$ is symmetric. Further, the parent node of nodes ν and μ is the P-node ρ. We have that $S(\rho) = \{\langle \overset{\circ}{\uparrow}, 2, \overset{\circ}{\uparrow}, 2\rangle, \langle \underset{\circ}{\uparrow}, 2, \underset{\circ}{\uparrow}, 2\rangle\}$. This is due to the fact that each good embedding of pert(ν) and pert(μ) contributes with an alternation at the poles u and v of ρ and that, in order for the such vertices to be bimodal in an embedding of pert(ρ), the good embeddings of the pertinent graphs of ν and μ have to be combined so not to create any additional alternation at u and v. Finally, since inserting edge uv in any of the good embeddings of pert(ρ) so to construct an embedding of $\mathcal{O}$ would create an additional alternation at either u or v, we have that $\mathcal{O}$ does not have any bimodal embedding.

The graphs in $\mathcal{F}$ are recursively defined as follows. For each integer $i \geq 0$, family $\mathcal{F}$ contains a graph G_i on $8+i$ vertices. Graph G_0 coincides with $\mathcal{O}$. Observe that, $\mathcal{O}$ is simple, outerplanar, and biconnected. For $i > 0$, let vz be any edge of G_i that is incident to the outerface of any outerplanar embedding of G_i. Graph G_{i+1} is constructed from G_i by introducing a new vertex x and edges vx and xz. It is easy to see that G_{i+1} is simple, outerplanar, and biconnected, if G_i is simple, outerplanar, and biconnected, which holds by induction, and that G_{i+1} does not admits a bimodal embedding as it contains $G_0 = \mathcal{O}$ as a subgraph. □

In view of the above negative result, we consider k-modal embeddings of outerplanar digraphs with $k > 2$. We remark that the next algorithmic results hold even for outerplanar digraphs containing multi-edges.

Lemma 2.22. *Let G be a biconnected outerplanar digraph and let c be a vertex of G. Then, G admits a 4-modal embedding in which c is at most bimodal.*

Proof. Let $\mathcal{T}$ be the SPQR-tree of G. By the next claim, we assume that $\mathcal{T}$ is rooted at a Q-node ω that corresponds to an edge e incident to c and that is adjacent to an S-node σ in $\mathcal{T}$.

Claim 2.3. *There exists an edge of G incident to c whose corresponding Q-node is adjacent to an S-node of $\mathcal{T}$.*

Proof. Consider an outerplanar embedding $\mathcal{E}$ of G and let (u, c) be and edge of G incident to c and to the outer face f_∞ of $\mathcal{E}$. Observe that, since G is biconnected, f_∞ is a simple cycle containing edge (u, c) and passing through all the vertices of G. We prove that (u, c) is not the intra-pole edge of any P-node of $\mathcal{T}$, which in turn implies the statement. Suppose, for a contradiction, that vertices u and c are the poles of some P-node θ of $\mathcal{T}$. Then, let μ_1 and μ_2 be the two S-nodes adjacent to θ in $\mathcal{T}$, and let ω be the Q-node adjacent to θ corresponding to edge (u, c). Clearly, u and v are also the poles of μ_1, μ_2, and ω. Assume, for the sake

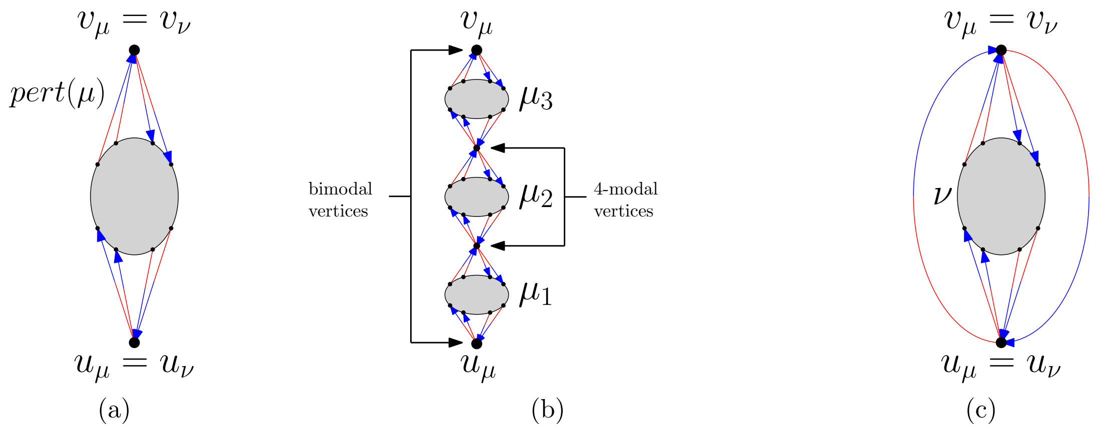


Figure 2.20: Illustrations for the proof of Claim 2.4. (a) An embedding of the pertinent graph of a node μ realizing tuple $\langle$⫯$, 1,$⫰$, 1\rangle$. (b) An S-node μ with three children μ_1, μ_2, and μ_3. Combining the embeddings of the pertinent graphs of the children of μ yields new 4-modal vertices (shared by the different children). (c) An embedding of the pertinent graph of a P-node μ that realizes tuple $\langle$⫯$, 1,$⫰$, 1\rangle$; the three intra-pole edges $u_\mu v_\mu$ and the two intra-pole edges $v_\mu u_\mu$ are embedded in the left and in the right outer face of an embedding of the pertinent graph of the S-node child ν of μ that realizes tuple $\langle$⫯$, 1,$⫰$, 1\rangle$.

of properly defining the pertinent graphs of the nodes of $\mathcal{T}$, that $\mathcal{T}$ is rooted at ω. Since (u, c) belongs to f_∞, a vertex of the pertinent graph of either μ_1 or μ_2 different from their poles must lie in the interior of f_∞. We thus obtain a contradiction to the fact that $\mathcal{E}$ is outerplanar. □

The proof of the lemma is based on the following claim.

Claim 2.4. *Let μ be a non-root node of $\mathcal{T}$. Then,* pert(μ) *admits a 4-modal embedding $\mathcal{E}_\mu$ realizing either tuple $\langle$⫯$, a,$ ⫰$, b\rangle$ or $\langle$⫰$, a,$ ⫯$, b\rangle$, with $a, b \leq 1$; see Fig. 2.20a.*

Proof. The proof is based on a bottom-up traversal of $\mathcal{T}$; when considering a non-root node μ of $\mathcal{T}$ with children $\mu_1, \dots, \mu_k$, we show that pert(μ) admits an embedding satisfying the conditions of the claim, assuming that the pertinent graphs pert$(\mu_1), \dots,$ pert(μ_k) of $\mu_1, \dots, \mu_k$ also satisfy such conditions.

Q-nodes. Suppose that μ is a Q-node with poles $\{u_\mu, v_\mu\}$. If μ corresponds to the directed edge $u_\mu v_\mu$, then pert(μ) admits a unique embedding, which realizes tuple $\langle$⫰$, 0, \in, 0\rangle$; otherwise, if μ corresponds to the directed edge $v_\mu u_\mu$, then pert(μ) admits a unique embedding, which realizes tuple $\langle \overset{\circ}{\uparrow}, 0, \underset{\circ}{\uparrow}, 0\rangle$.

S-nodes. Suppose that μ is a Q-node and let $(u_1 = u_\mu, \dots, u_k = v_\mu)$ be the path representing the skeleton of μ. Also, let μ_i be the children of μ corresponding to the virtual edge (u_i, u_{i+1}) of skel(μ), for $i = 1, \dots, k-1$. Suppose that pert(μ_1) and pert(μ_2) contain edges with different orientations incident to u_μ and to v_μ, respectively; the other cases being simpler.

We show how to construct an embedding $\mathcal{E}_\mu$ of pert(μ) realizing tuple $\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle$; refer to Fig. 2.20b. We obtain $\mathcal{E}_\mu$ by selecting for each child μ_i of μ an embedding $\mathcal{E}_{\mu_i}$ realizing tuple $\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle$; observe that such embeddings exist by hypothesis. Clearly, $\mathcal{E}_\mu$ realizes the embedding tuple $\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle$, as the embedding pair of pert(μ_1) at μ is $(\overset{\circ}{\uparrow}, 1)$ and the embedding pair of pert(μ_k) at v_μ is $(\underset{\circ}{\uparrow}, 1)$, by construction. We now show that $\mathcal{E}_\mu$ is 4-modal at any vertex v of pert(μ). If $v \in \{u_1 = u_\mu, u_k = v_\mu\}$, then, by the above arguments, v is bimodal in $\mathcal{E}_\mu$. If $v \notin \{u_2, \dots, u_{k-1}\}$, then v is an internal node of the pertinent graph of some children μ_i of μ, and thus v is 4-modal in $\mathcal{E}_\mu$ since it is 4-modal in $\mathcal{E}_{\mu_i}$ by hypothesis. Finally, if $v = u_i$, with $i \in \{2, \dots, k-1\}$, then v is at most 4-modal in v as it is at most bimodal in both $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$ by hypothesis. An embedding of pert(μ) realizing tuple $\langle \underset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1\rangle$ can be obtained as a horizontal flip of $\mathcal{E}_\mu$.

P-nodes. Suppose now that μ is a P-node. Recall that, since the unique child σ of the root Q-node ω is an S-node and since G is outerplanar, node μ has exactly one non-Q-node child ν. Suppose that pert(ν) contains edges with different orientations incident to $u_\mu = u_\nu$ and to $v_\mu = v_\nu$, respectively; the other cases being simpler.

We show how to construct an embedding $\mathcal{E}_\mu$ of pert(μ) realizing tuple $\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle$; refer to to Fig. 2.20c. We obtain $\mathcal{E}_\mu$ by selecting an embedding $\mathcal{E}_\nu$ of pert(ν) realizing tuple $\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle$ (such embeddings exist by hypothesis) and by embedding each directed edge e corresponding to a Q-node child in the left or in the right face of $\mathcal{E}_\nu$ depending on whether $e = u_\mu v_\mu$ or

$e = v_\mu u_\mu$, respectively. Clearly, $\mathcal{E}_\mu$ realizes the embedding tuple $\langle \overset{\circ}{\uparrow}, 1, \underset{\circ}{\uparrow}, 1\rangle$, as the embedding pair of $\text{pert}(\mu_1)$ at μ is $(\overset{\circ}{\uparrow}, 1)$ and the embedding pair of $\text{pert}(\mu_k)$ at v_μ is $(\underset{\circ}{\uparrow}, 1)$, by construction. Also, vertices u_μ and v_μ are bimodal in $\mathcal{E}_\mu$ and any internal vertex of $\text{pert}(\mu)$ is also an internal vertex of $\text{pert}(\nu)$ and thus it is 4-modal in $\mathcal{E}_\mu$ since it is 4-modal in $\mathcal{E}_\nu$. As for the S-node case, an embedding of $\text{pert}(\mu)$ realizing tuple $\langle \underset{\circ}{\uparrow}, 1, \overset{\circ}{\uparrow}, 1\rangle$ can be obtained as a horizontal flip of $\mathcal{E}_\mu$. This concludes the proof of the claim. □

By Claim 2.4, we have that $\text{pert}\,\sigma$ admits a 4-modal embedding $\mathcal{E}_\mu$ that has at most an alternation at both its poles. Thus, by placing edge e in the outer face of $\mathcal{E}_\mu$, we obtain a 4-modal embedding of G that is bimodal at the poles of ω, one of which is c. This concludes the proof of the lemma. □

We are now ready to present the main result of the section.

Theorem 2.14. *Every outerplanar multigraph admits a* 4*-modal embedding.*

To prove Theorem 2.14, we are going to need the following simple observation, which follows from the proof of Theorem 2.5.

Observation 2.11. *For any modality $k \geq 2$, every digraph whose blocks admit a k-modal embedding in which the modality of a specified vertex is at most* 2 *also admits a k-modal embedding.*

Proof of Theorem 2.14. Lemma 2.22 and Observation 2.11 allow us to prove Theorem 2.14 as follows. Let G be an outerplanar multigraph. For each block β_i of G, by Lemma 2.22, we can compute a 4-modal embedding of β_i in which the modality of the parent cut-vertex c_i of β_i is at most 2. Then, by Observation 2.11, all such embeddings can be composed together to obtain a 4-modal embedding of G. This concludes the proof.

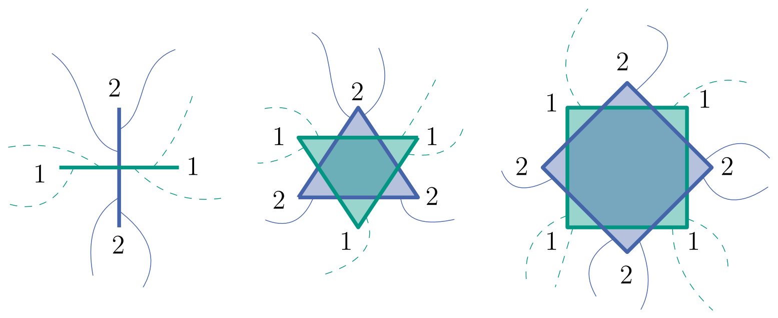


Figure 2.21: Arrangements or pairs of rotated convex k-gons are in one to one correspondence with $2k$-modal embeddings.

2.11 Conclusions and Open Problems

In this chapter, we studied the complexity of the k-Modality problem, with special emphasis on $k = 4$. We provided complexity, algorithmic, and combinatorial results. Our main algorithmic contribution for $k = 4$ and $\Delta \leq 6$ leverages an elegant connection with the NAE-satisfiability of special CNF formulas, whose study allowed us to strengthen a result in [86].

Moreover, we showed notable applications of the previous results to some new interesting embedding problems for clustered networks, some of which solve open problems in this area [9, 54]. In this side of the spectrum, it is work pointing out that the connection between modal embeddings and intersection-link representations can be exploited to show NP-hardness results and polynomial-time algorithms analogous to those proved in Theorem 2.4 and Theorem 2.3, respectively. In particular, by allowing the geometric objects that represent vertices to be *rotations* of the same convex k-gon, we can construct arrangements of pairs of geometric objects such that, traversing the boundary of the arrangements clockwise, we encounter each object k times along such a boundary; Fig. 2.21 illustrates the example of segments, triangles, and squares, which correspond to 4-, 6- and 8-modal embeddings, respectively. The above observation allows to extend the positive and negative results mentioned above also to this extended setting for Clique Planarity.

Several interesting open questions arise from our research: **Q1:** What is the relationship between k-modality and maximum degree? In particular, is there a non-trivial function f such that, given any digraph G of maximum degree Δ, deciding whether G admits an $f(\Delta)$-modal embedding can be done in polynomial-time? **Q2:** Is there a polynomial-time algorithm for computing k-modal embeddings of partial 2-trees, whose running time does not depend on k? **Q3:** It is easy to see that any directed tree always admits a 2-modal embedding. In Section 2.10, we showed that this result does not extend to every outerplanar digraph. Despite this, we proved that every outerplanar digraph is 4-modal. Along this line, are there relevant families of planar digraphs that always admit a k-modal embedding, for some non-negative even integer k?

Chapter 3

Minimum Width Drawings of Phylogenetic Trees

3.1 Introduction

A *phylogenetic tree* is a rooted tree that represents evolutionary relationships among a group of organisms. The branching represents how species are believed to have evolved from common ancestors and the vertical height difference represents genetic difference or, in “clock trees”, time estimates for when the species diverged. In this chapter, we study the algorithmic complexity of producing minimum-width orthogonal upward drawings of phylogenetic trees. We consider vertices as extending horizontally and edges as extending vertically, making edge-length equal to vertical distance.[1] See Fig. 3.1.

Given a phylogenetic tree, T, with a length, L_e, defined for each edge, $e \in T$, the *min-width orthogonal phylogenetic tree* drawing problem is to produce an upward planar orthogonal

[1]To consider a node-link representation with 1-bend edges, edge-lengths does not coincide with vertical distance so special considerations must be taken.

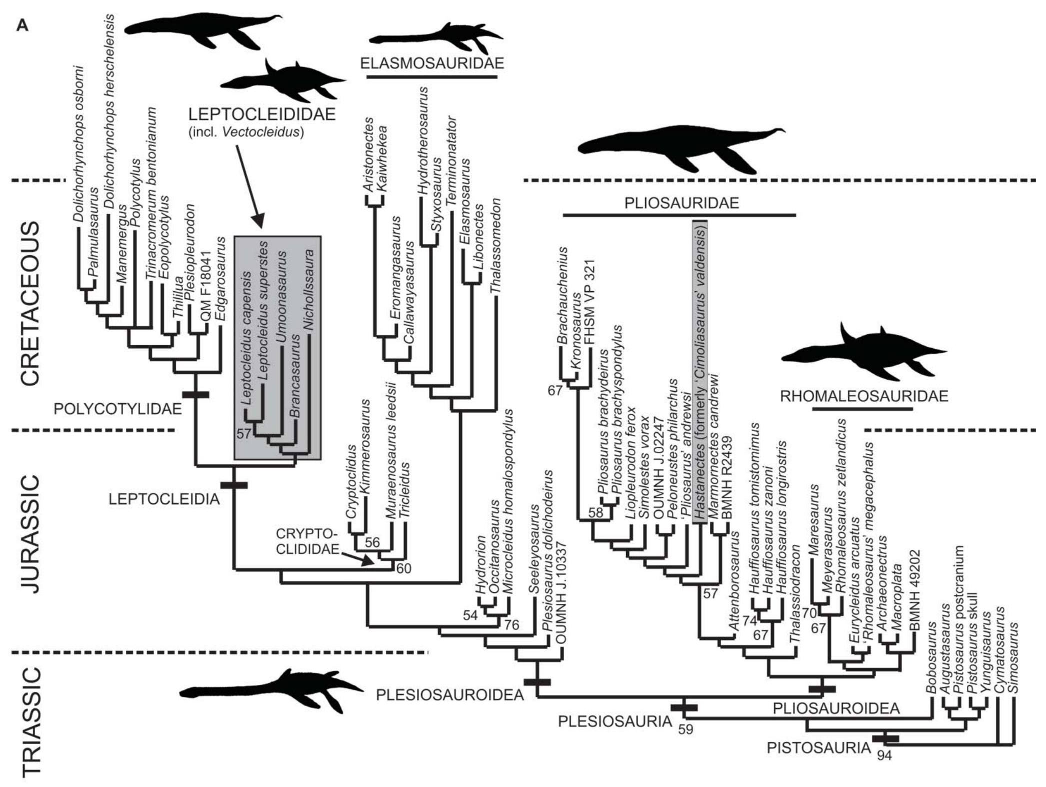


Figure 3.1: An orthogonal upward drawing of a phylogenetic tree, from [25].

drawing of T that satisfies each edge-length constraint (so the drawn vertical length of each edge e is L_e)[2] and minimizes the width of the drawing of T. The motivation for this problem is to optimize the area of an orthogonal drawing of a phylogenetic tree, since the height of such a drawing is fixed by the sum of lengths of the edges on a longest root-to-leaf path. From an algorithmic complexity perspective, note that this problem is trivial in clock trees since all leaves represent non-extinct species, implying the edge length sum for every root-to-leaf path is the same and all the leaves should be drawn on the same level in this case. Thus, we are interested in the general case, as shown in Fig. 3.1, for this implies that width improvement is possible by allowing subtrees of extinct species to have overlapping x-coordinates with species that evolved after extinction events. In this chapter, we show that min-width orthogonal phylogenetic tree is NP-hard if the ordering of leaves is unconstrained

[2]Without loss of generality, each node is embedded below its parent; other orientations, such as drawing nodes above their parents, are equivalent to this one via rotation.

and we give a linear-time algorithm for ordered trees. Also, we describe several heuristic algorithms for the unconstrained case and show their effectiveness by experimentation.

3.1.1 Related work

There is considerable prior work on methods for producing combinatorial representations of phylogenetic trees, that is, to determine the branching structures and edge lengths for such trees, e.g., see [81, 94, 66, 58, 100]. For this chapter, we assume that a phylogenetic tree is given as input.

Existing software systems produce orthogonal drawings of phylogenetic trees, e.g., see [32, 67, 83, 26, 84, 103], however we are not familiar with any previous work on characterizing the algorithmic complexity of the minimum-width orthogonal phylogenetic tree drawing problem. Bachmaier *et al.* [14] present linear-time algorithms producing other types of drawings of ordered phylogenetic trees, including radial and circular drawings, neither of which are orthogonal drawings.

Although we are not aware of prior work on the min-width orthogonal phylogenetic tree drawing problem, there is prior related work on orthogonal tree drawing and drawing graphs with fixed edge lengths. Several researchers have studied area optimization problems for planar upward tree drawing without edge-length constraints, e.g., see [52, 92, 72, 34, 33, 51, 89, 3, 38].

For example, Chan *et al.* [34] show that every n-node binary tree has a planar upward orthogonal straight-line grid drawing with $O(n \log n)$ area for a wide range of aspect ratios, and this result was recently improved by Chan [33]. Also, Kim [72] shows that every n-node ternary tree has a planar upward orthogonal straight-line grid drawing with $O(n \log n)$ area. In addition, Frati [51] studies order-preserving straight-line orthogonal drawings of binary trees and ternary trees, showing that subquadratic area drawings are possible, but if such drawings must be upward, then there is a quadratic area lower bound. (See also Rusu and Fabian [89].) Alam *et al.* [3, 4] study the problem of upward planar orthogonal drawings of binary trees with a minimum number of edge segments. Di Battista *et al.* [38] study orthogonal and quasi-upward drawings of graphs with vertices of prescribed sizes.

In terms of hardness, Biedl and Mondal [20] show it is NP-hard to decide whether a tree has a strictly upward straight-line drawing in a given $W \times H$ grid. Bhatt and Cosmadakis [19] show that it is NP-complete to decide whether a degree-4 tree has a straight-line (non-upward) orthogonal grid drawing where every edge has length 1. Gregori [57] extends their result to binary trees and Brunner and Matzeder [30] extend their result to ternary trees.

Eades and Warmald [47] show that it is NP-hard to determine whether one can produce a straight-line drawing of a graph, G, with given edge lengths and without crossings, even if G is 2-connected and all edge lengths are 1 (i.e., for 2-connected matchstick graphs). Cabello *et al.* [31] show that this problem is solvable in linear-time for planar 3-connected triangulations, but it is NP-hard for planar 3-connected infinitesimally rigid graphs. More recently, Abel *et al.* [2] show that this decision problem is $\exists\mathbb{R}$-complete.

In addition, Brandes and Pampel [28] show that several order-constrained orthogonal graph drawing problems are NP-hard, and Bannister and Eppstein [16] show that compacting certain nonplanar orthogonal drawings is difficult to approximate in polynomial time unless $P = NP$. These previous hardness proofs do not apply to the min-width orthogonal phylogenetic tree drawing problem, however.

3.2 Min-Width Phylogenetic Tree Drawing is NP-hard

In this section, we prove the following theorem:

Theorem 3.1. *Computing the minimum width required for an upward planar orthogonal* *drawing of a binary tree with fixed vertical edge lengths is NP-hard.*

We prove this via a reduction from NAE-3-SAT, a variant of NAESAT where each clause consists of exactly 3 literals. We remind the reader that in NAESAT an assignment is considered satisfying when each clause's boolean values are not all the same. An instance, ϕ, of NAE-3-SAT is defined by n variables $X = \{x_1, ..., x_n\}$ and m clauses $C = \{c_1, ..., c_m\}$.

Given a truth assignment, a literal is considered *satisfied* if the literal evaluates to true. A truth assignment $\mathcal{A}$ satisfies ϕ when each clause contains only one or two satisfied literals. Each clause must therefore also have one or two unsatisfied literals.

Given a truth assignment, $\mathcal{A}$, we define $\overline{\mathcal{A}}$ to be the truth assignment where each assigned truth value is the negation of the truth value assigned in $\mathcal{A}$. If $\mathcal{A}$ satisfies ϕ each clause must contain a satisfied literal, l_1, and an unsatisfied literal, l_2, then $\overline{\mathcal{A}}$ must also contain an unsatisfied literal, l_1, and a satisfied literal, l_2. Thus for any $\mathcal{A}$ satisfying ϕ, $\overline{\mathcal{A}}$ must also satisfy ϕ.

We use this property to create a combinatorial phylogenetic tree T from an instance ϕ such that T admits an upward orthogonal drawing of width $4n + 4$ if and only if ϕ is satisfiable. Since the vertical length of each edge is fixed, each node in T has a fixed level or height at which it needs to be drawn. We say that two nodes are *horizontally aligned* when they lie at the same level. Our reduction follows the general structure shown in Fig. 3.2a. The top part of the tree forms a *truth-assignment gadget*, where each variable has two branches, one corresponding to a true value (shown in green) and the other corresponding to a false value (shown in red). The truth-assignment gadget's combinatorial order therefore defines truth assignments for $\mathcal{A}$ and $\overline{\mathcal{A}}$, on the left and right, respectively. Fig. 3.2a illustrates all true and all false truth assignments (respectively) whereas Fig. 3.2b illustrates a satisfying assignment for a two-clause formula. The middle part comprises a sequence of *clause gadgets*, with 3 rows for each clause gadget, plus rows of *alignment nodes*, separating the clause gadgets. The

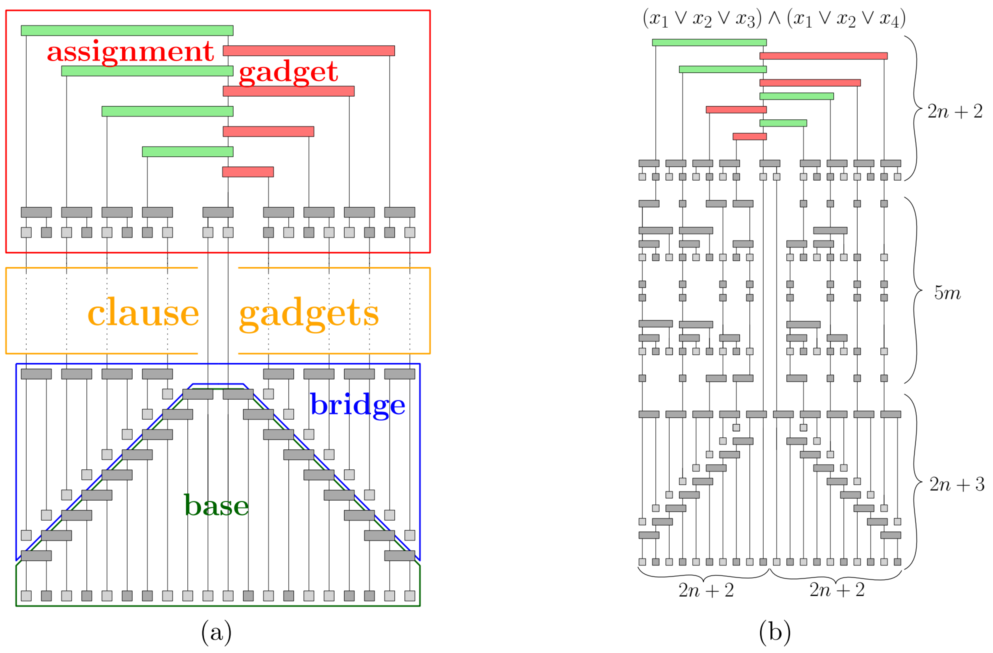


Figure 3.2: Overview of Theorem 3.1 (a) General structure. (b) Drawing of reduction corresponding to a satisfying assignment, $x = \{true, true, false, false\}$

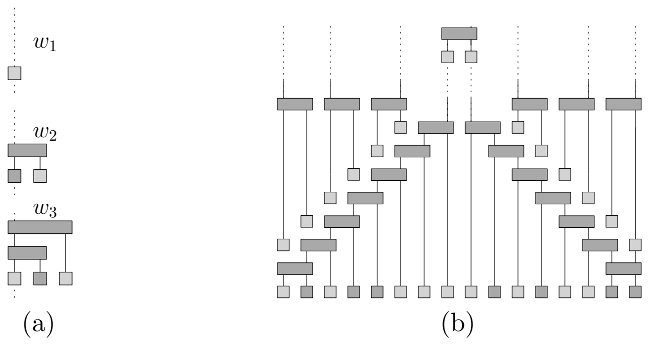


Figure 3.3: Construction pieces: (a) w_k substructures (b) Pyramidal embedding of bridge and base for $n = 3$

bottom parts of the tree comprise *bridge* and *base* gadgets, which ensure that the pair of branches corresponding to the true and false values for each variable must be on opposite sides (see Lemma 3.2).

Let $P(k)$ be a path of k nodes with unit edge-lengths, where p_1 is the root of the path and $p_i \in P, 2 \leq i \leq k$ is the i-th node along the path. A node p_i may have two children, the node p_{i+1} (if it exists) and some other new node (or two if $i = k$). We define the substructures

w_1, w_2, and w_3 (Fig. 3.3a), consisting of the minimal tree with 1, 2, and 3 leaves at the same height.

We proceed from top to bottom to describe the structure of T. The tree, T, is rooted on the first node of a path $P(2n)$, called the *variable path*. We first add a w_2 component to p_{2n} a unit-length away as its first child, which we use to root the base. We then add a w_2 component as the second child of each node $p_i, 1 \leq i \leq 2n$ so the $2n$ new w_2 components lay in horizontal alignment with the one rooting the base. These $2n$ w_2 components connect each node in the variable path to a branch. Of the $2n$ branches, there are two for each of the n variables in ϕ. We label the branch originating from p_i with the truth assignment setting $x_{\lfloor (i+1)/2 \rfloor}$ to 'true' if i is odd and 'false' if i is even. These branches are now called *assignment branches*.

Each assignment branch consists of m clause components connected in a path by unit-length edges, where the j-th component (for $1 \leq j \leq m$) belongs to the clause gadget of clause c_j. Each clause component has the same height so each clause, c_j, occupies a distinct horizontal band of space in the tree. Each clause component consists of a w_k component and two surrounding alignment nodes, one above the w_k component and one below. Each w_k component has height 3 and for clause components corresponding to the same clause the leaves of all w_k are horizontally aligned.

Recall that each variable has two assignment branches associated to it, one corresponding to setting the variable to true and one to false. A clause component is defined by both the clause, c_j, and the branch's labeled truth assignment, $x_i = \{true, false\}$. If x_i does not participate in clause c_j then its clause component has a w_2 substructure. If x_i appears as a literal in c_j and evaluating it with the assigned truth value satisfies the literal then the clause component has a w_3 substructure. However, if the literal is unsatisfied then it contains a w_1 substructure. For example, in Fig. 3.2b clause $c_1 = (x_1 \vee x_2 \vee x_3)$, the branch labeled $x_1 = true$ contains w_3, whereas $x_1 = false$ contains w_1 and both x_4 assignment branches contain w_2.

Top alignment nodes have incoming edges connecting from the bottom alignment nodes of the previous clause. These two consecutive rows of alignment nodes enable the clause gadgets along an assignment branch to be shifted left or right to efficiently align into the available space within that gadget (see Section 3.2.1).

After the last clause component in each branch (labeled with x_i) we attach a node one unit away from the last shifter and give it two children, one $2n - 2i + 1$ units away and the other $2n - 2i + 2$. These nodes together form the bridge gadget.

To build the base, we set a path $P(2n + 2)$ as a child $5m + 2$ units away for each of the two leaves in the remaining copy of w_2 attached to p_{2n}. For each non-leaf node in the path we just connected, we set their remaining child to be a single node horizontally aligned with the leaf.

Lemma 3.1. *At minimal width, the base and bridge can only assume a pyramidal embedding. We define a pyramidal embedding as the embedding of the base in which nodes closer to the root lie closer to the center and nodes further from the root approach the outer sides, as shown in Fig. 3.3b.*

Proof. We define the filled area of a drawing as the sum of the space used by each node (equal to its width), and the space used by each edge (equal to its length). The length of the edges is fixed, but we can change the filled area by changing the layout to minimize the width of the non-leaf nodes. When the base is in the pyramidal embedding, it fills the least possible area, since every non-leaf node must have width equal to its number of children. We now show that this is the only configuration with minimal area.

We begin from the parents of the base gadget, which belong to a copy of w_2 (Fig. 3.3a). The two incoming edges from this structure must be next to one another, with no gap between them. The two nodes at the top level of the base each then have both a leaf and a large subtree attached.

The width of each of these top-level nodes must be two, and the only way to achieve this is that each node's leaf must lie on the inside and its subtree on the outside. Similarly, for each subsequent node along the path, the same argument shows that its leaf must lie on the inside. This proves by induction that the base needs to be in a pyramidal embedding. The bridge then must fit against the base. The bridge nodes with the lowest leaves must be on the outside, with the next lowest leaves next to them, and so on by induction back to the center. This shows that the pyramidal embedding is the unique embedding that minimizes the filled area. Since the levels containing the base and bridge nodes are completely packed with no gaps, this also implies that the pyramidal embedding is the unique embedding that minimizes the width of these levels. □

Lemma 3.2. *The embedding of the truth-assignment gadget defines assignments $\mathcal{A}$ and $\overline{\mathcal{A}}$.*

Proof. As a consequence of Lemma 3.1, the two edges coming into the base must be centered. We refer to these two edges as 'the split'. The base takes up width $2n + 1$ on either side of these two edges, and each branch needs width two, so at most n can fit on either side.

Furthermore, in order for the bridge to assume the pyramidal embedding, each leaf on the left side must have a corresponding leaf at the same level on the right side. Note that the height of the leaves in the bridge gadget depends on which variable branch they are attached to, so that leaves on the same level correspond to the two assignments of the same variable. Therefore, there is one assignment branch for each variable on each side, so the assignments labeling the branches on one side must all be of different variables and thus describe a truth assignment. The branches on the opposite side have the opposite assignment for each variable. Let the truth assignment on the left side be $\mathcal{A}$, and the one on the right side be $\overline{\mathcal{A}}$. □

Lemma 3.3. *Given truth assignments $\mathcal{A}$ and $\overline{\mathcal{A}}$, a clause in ϕ is satisfied if and only if the width used by the clause gadget is at most $2n + 1$ on both sides of the split.*

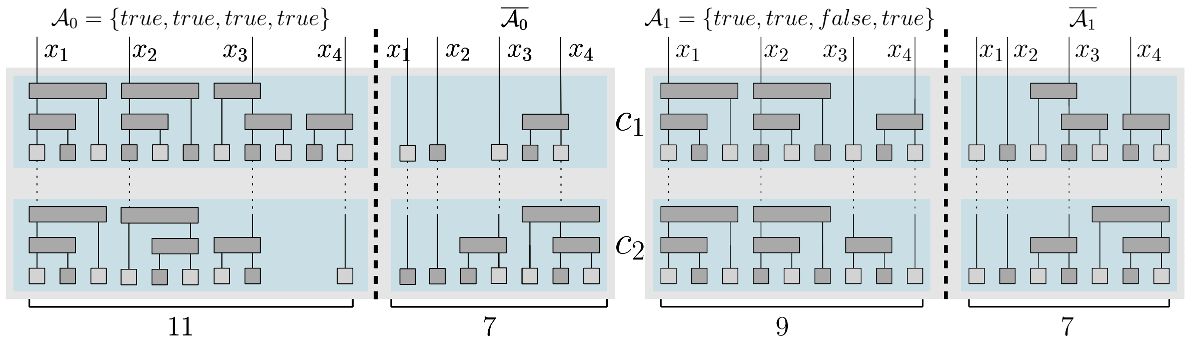
Proof. The maximum number of horizontally aligned nodes on the left side of the clause gadget is equal to the sum of the width of the w_k's, $k \in \{1, 2, 3\}$, in each assignment branch for assignment $\mathcal{A}$. Recall that within a clause gadget each assignment branch has an embedded copy of either w_1, w_2 or w_3. We define function $p_{i,j}$, which describes the width of an embedded copy w_k, and $S_j(\mathcal{A})$, which describes the sum of all the embedded element's widths, as follows:

$$S_j(\mathcal{A}) = \sum_{i=1}^{n} p_{i,j}(\mathcal{A}), \text{ where } p_{i,j}(\mathcal{A}) = \begin{cases} 3 & \text{if } \mathcal{A}\text{does satisfy the literal of } x_i \text{ in } c_j \\ 1 & \text{if } \mathcal{A}\text{doesn't satisfy the literal of } x_i \text{ in } c_j \\ 2 & \text{if } c_j \text{ has no literal of } x_i \end{cases}$$

By definition, a clause can only be satisfied if one or two of the clause's literals are satisfied and the remaining one or two literals must be unsatisfied. Without loss of generality we can assume that each clause consists of two or three literals from distinct variables.[3] For clauses with literals from three distinct variables, any clause c_j satisfied by a truth assignment $\mathcal{A}$ must have only one or two satisfied literals. If $\mathcal{A}$ satisfies only one literal in clause, c_j, then $S_j(\mathcal{A})$ evaluates to $3+1+1+2(n-3) = 2n-1$, since only 3 of the n variables participate in a clause. If it satisfies two literals, it evaluates to $3+3+1+2(n-3) = 2n+1$ instead. If $\mathcal{A}$ satisfies c_j, then $\overline{\mathcal{A}}$ also satisfies c_j implying $S_j(\mathcal{A})$ and $S_j(\overline{\mathcal{A}})$ are both at most $2n+1$. On the other hand, if $\mathcal{A}$ doesn't satisfy c_j, then it either satisfies all three literals, and $S_j(\mathcal{A})$ evaluates to $3+3+3+2(n-3) = 2n+3$, or $\overline{\mathcal{A}}$ satisfies all three literals and $S_j(\overline{\mathcal{A}}) = 2n+3$. Therefore if $\mathcal{A}$ doesn't satisfy c_j, $S_j(\mathcal{A})$ or $S_j(\overline{\mathcal{A}})$ will exceed $2n+1$.

For clauses with literals from two distinct variables, any $\mathcal{A}$ can only satisfy c_j with one satisfied literal and one unsatisfied literal. $S_j(\mathcal{A})$ and $S_j(\overline{\mathcal{A}})$ both evaluate to $3+1+2(n-2) = 2n$, both remaining strictly less than $2n+1$. However if $\mathcal{A}$ doesn't satisfy c_j then either $\mathcal{A}$ satisfies both literals, and $S_j(\mathcal{A}) = 3+3+2(n-2) = 2n+2$, or $\overline{\mathcal{A}}$ satisfies both literals. Therefore if

[3] Degenerate cases to consider include cases when a variable contributes multiple literals to a single clause. We can safely ignore cases when all three identical literals are present (which is not satisfiable) and when positive and negated literals of the same variable are present (since the clause is always satisfied). When a literal is repeated exactly twice, we handle it as a clause of only the two distinct literals.



$\mathcal{A}_0 = \{true, true, true, true\}$
$\overline{\mathcal{A}_0}$
$\mathcal{A}_1 = \{true, true, false, true\}$
$\overline{\mathcal{A}_1}$
x_1
x_2
x_3
x_4
c_1
c_2
11
7
9
7

Figure 3.4: Original and satisfied clause gadgets for $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_4})$

$\mathcal{A}$ doesn't satisfy c_j, $S_j(\mathcal{A})$ or $S_j(\overline{\mathcal{A}})$ will exceed $2n+1$. We have now proved that $S_j(\mathcal{A})$ and $S_j(\overline{\mathcal{A}})$ are both at most $2n+1$ if and only if $\mathcal{A}$ and $\overline{\mathcal{A}}$ both satisfy c_j. If the clause gadget is able to assume a dense embedding, then the width needed by the w_k components on opposite sides should be exactly equal to $S_j(\mathcal{A})$ and $S_j(\overline{\mathcal{A}})$. What remains is to prove that a dense embedding is possible, we prove this using the alignment nodes.

3.2.1 Alignment Nodes

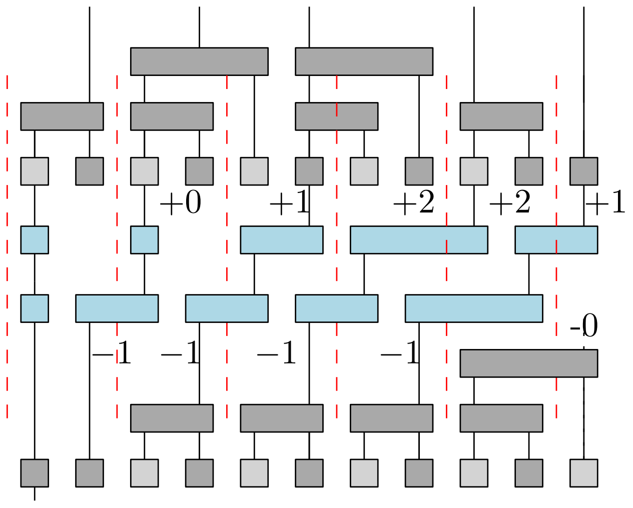


Figure 3.5: Impossible consecutive set of clauses, $c_1 = x_2 \vee x_3 \vee x_6$ and $c_2 = x_1 \vee \overline{x_2} \vee x_6$, with satisfying assignment $x = \{false, true, true, ...\}$

Alignment nodes only need to be able to fully realign satisfied clauses; therefore within the three structures at least one must be of width three and at least one of width one. It follows that if we consider the periodicity of the column at which the edge drops, the maximum possible phase difference in these periods is shown in Fig. 3.5. Considering this order as an extreme case is sufficient because after using both satisfied literal structures (after x_3 in c_1) the only remaining widths could be two (which maintains the phase difference) or one (which reduces the phase difference). The same argument can be made after using both unsatisfied literal structures after x_2.

Without alignment nodes it would be impossible to connect x_2 and x_3 to the next clause, but adding the two row of alignment nodes (shown in blue) between clauses enable them to remain connected. This allows us to maintain the tightness of each clause and not artificially widen the drawing because of interactions between adjacent clauses. $\square$

Wrapping up the main proof, if any clause is not satisfied the clause gadget will exceed the allowable space of $2n+1$ and increase the width to at least $4n+5$. Therefore only a satisfying assignment $\mathcal{A}$ would retain a width of $4n+4$, proving that if a satisfying assignment for ϕ exists then there exists an embedding of T with width $4n+4$.

On the other hand, if a tree T has a drawing of width $4n+4$ then every clause was satisfied (following Lemma 3.3), and thus $\mathcal{A}$ must describe a satisfying assignment for ϕ. This proves that T can be embedded with width $4n+4$ if and only if ϕ is satisfiable.

Furthermore, our reduction features a multi-linear number of nodes: the variable gadget has $8n+3$ nodes, the clause gadgets exactly $5mn$ nodes, the bridge gadget $6n$ and the base gadget $8n+6$ totaling exactly $22n+5mn+12$ nodes. This completes our proof of Theorem 3.1.

3.3 Linearity for Fixed-Order Phylogenetic Trees

Theorem 3.2. *A minimum width upward orthogonal drawing of a fixed-order n-node phylogenetic tree can be computed in $O(n)$ time.*

We provide an algorithm that computes a minimum width drawing. The key idea is to construct a directed acyclic graph (DAG) of the positional constraints between nodes and edges. The DAG can then be processed efficiently to determine a positioning of each node and edge that ensures the minimum width. Let S be a set of non-intersecting orthogonal objects (e.g., rectangles and segments) in the plane. Two objects s and s' are horizontally visible if there exists a horizontal segment that intersects s and s' but no other object of S. Since the height of each object of our drawing is fully determined by the edge lengths of the tree, determining which objects are horizontally visible is essential to construct a minimum width drawing. For a fixed order combinatorial phylogenetic tree $T = (V, E)$, the *Constraint Graph* $D = (U, A)$ of T is a directed graph with a vertex for each left and right side (of the rectangle representing the node in the drawing) of each node of T and one for each edge of T. (See Fig. 3.6.) An arc $e = uv \in A$ if the objects corresponding to u and v are horizontally visible and u precedes v as determined by the fixed order.

Lemma 3.4. *The constraint graph D of the fixed order n-node phylogenetic tree $T = (V, E)$*

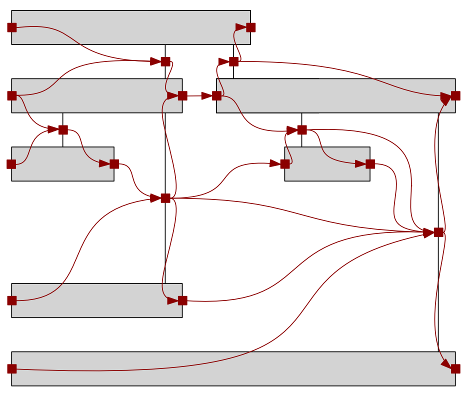


Figure 3.6: The constraint graph of Lemma 3.4

is a DAG with $3n-1$ *vertices and* $O(n)$ *edges, where* $n = |V|$.

Proof. Our objects are the left and right sides of each vertex in T, and the edges in T. This gives us two vertices in D for each vertex in T, and one for each edge. Since T is a tree, it must have $n-1$ edges, so D has $3n-1$ vertices. If two objects are horizontally visible, then there is a segment between them that crosses only those two objects. We will use these segments to build a planar embedding of D, which will imply that D has $O(n)$ edges.

Let the collection of segments connecting our objects be S. We first construct a larger planar graph D', in which the vertices are the endpoints of S. The edges of D' include all of the segments in S. We also add edges connecting each vertex to the vertices immediately above and below it that represent the same object. By the definition of horizontally visible, each segment in S can only intersect two objects, so none of the segments in S can intersect. The additional edges also cannot intersect, since they are ordered by the height of the vertices. Therefore, D' is planar.

We then contract all of the edges that connect two vertices in D' corresponding to the same object. This produces the DAG D. Since D is a contraction of a planar graph, it must also be planar. $\square$

The algorithm has two main steps. First, we construct the constraint graph D, and then we process the constraint graph to find a minimum-width drawing. As we have mentioned, the vertices of D can be constructed directly from the vertices and edges of T. We now show that the arcs in D can be created using a single pre-order (node, then children left to right) traversal of the tree, while growing a frontier indicating the rightmost object seen at each height. We maintain the frontier efficiently as an ordered list of height ranges. Whenever we update the frontier we have found a new rightmost object. If we are not extending the frontier (i.e. adding to the end of the list), then we have covered or partially covered some object. The two objects must be horizontally visible so we add a new directed arc from the left object to the right.

For a linear algorithm, we must avoid searching in the frontier for the position of each object. The key observation is that, while processing a node v of T, the edges and nodes of the subtree rooted at v only affect the frontier below v. In other words, the position of v in the frontier doesn't change while processing its subtree. When a child is completely processed we can find the next sibling's position in the frontier by looking at the position of their parent.

Once the constraint graph is constructed, it must be processed to find the positions at which to draw each object. We process the vertices of D in topological order. Vertices that have no incoming arcs, which are the sources of the constraint graph, must be the left side of vertices of T and can be positioned at x-coordinate 0. At each remaining vertex, we check its incoming arcs and assign it the leftmost position that is to the right of every vertex in its in-neighborhood. Because the arcs represent the necessary order at each height and the sources of the DAG are positioned as far left as possible, a simple inductive argument proves that the resulting drawing has minimum width.

Traversing the tree to construct our DAG requires us to update the frontier once for every arc, source, and sink of the DAG. Each update takes constant time, so by Lemma 3.4 determining the arcs of the constraint DAG takes a total of $O(n)$ time. The time taken for the processing step includes the topological sort and the time to check each incoming arc at each vertex. Both of these are bounded by the number of arcs, so by Lemma 3.4 processing the DAG also takes $O(n)$ time. In conclusion both steps take $O(n)$ time so the algorithm takes $O(n)$ in total. This completes the proof of Theorem 3.2.

A Note On Improving The Drawing. The above algorithm produces excessively wide nodes, since some nodes may be extended further to the left than necessary. The aesthetic quality of the tree can be improved by making each node as small as possible, without affecting the minimum width, by doing an extra tree traversal and moving edges and node sides to the right.

3.4 Heuristics and Experiments

In this section we evaluate different heuristics for generating small width drawings of real-world phylogenetic trees, see for example Fig. 3.9. First, we introduce the heuristics evaluated and show that two natural greedy heuristics give a bad approximation guarantee (Section 3.4.1) Then, we describe the conditions used for the experiments (Section 3.4.2).

3.4.1 Heuristics Evaluated

Let T be a combinatorial phylogenetic tree. Once the order of the children of each vertex is determined, we can use Theorem 3.2 to find a minimum width drawing that respects the edge lengths. Consequently, a heuristic only needs to define the ordering of the children in each vertex. We define the *flip* of a tree (or subtree) rooted at v as the operation of reversing the order of the children of v and every descendant of v. Flipping a tree corresponds to flipping its drawing and does not affect its minimum width.

Greedy Heuristic. The *greedy* heuristic proceeds bottom up from the leaves to the tree's root. For each vertex v with children $c_0, \ldots, c_k$ this heuristic assumes that the order of the subtrees rooted at its children are fixed and finds the way to arrange its children to minimize width. To do so it considers every possible permutation and combination of flipped children. In general, for a degree d vertex, the greedy heuristic checks $O(d!3^{d-1})$ possible orderings, bounded degree trees therefore take $O(1)$ time per vertex. Because the algorithm calculates the minimum width drawing using the $O(n)$ algorithm from Theorem 3.2, and runs it $O(1)$ times per vertex, the total running time of the heuristic is $O(n^2)$ for bounded degree trees.

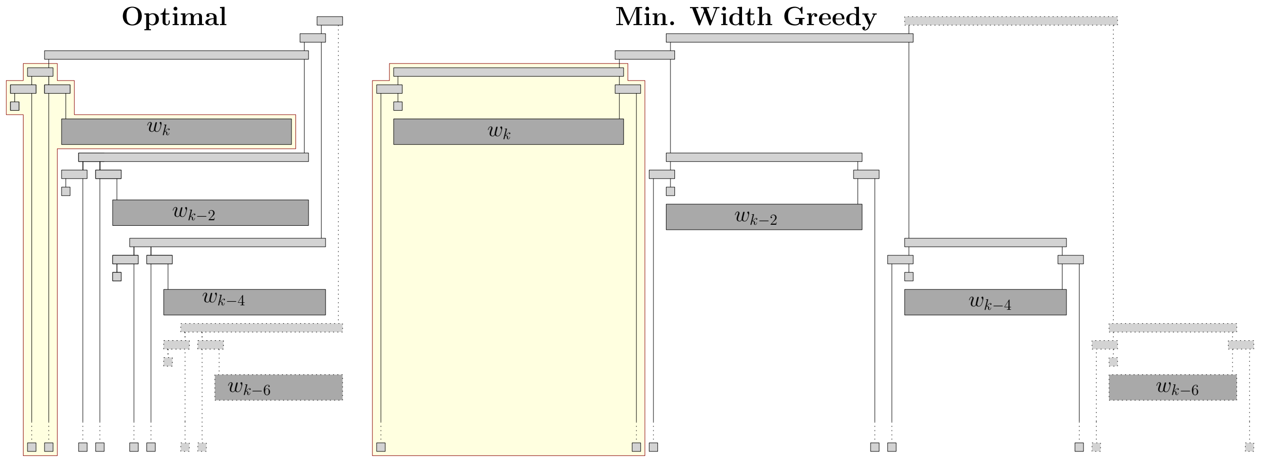


Figure 3.7: Tree structures causing worst case performance for the greedy heuristic.

Theorem 3.3. *The greedy heuristic has an approximation ratio of at least* $\Omega(\sqrt{n})$*, even for binary phylogenetic trees.*

Proof. Recall the structures for w_k as described in Fig. 3.3a and consider equivalent structures for larger values of k where all k leaves lie in horizontal alignment. Using this definition of w_k, Fig. 3.7 shows a tree structure where a minimum width embedding of the subtree in yellow makes it impossible for the entire drawing to admit minimum width. For a minimum width subtree, the greedy heuristic must choose the smallest width possible $(k + 2)$ thus placing the long edges on opposite sides. In an optimal ordering the subtree's embedding would need to be a unit wider $(k + 3)$ and place both of long edges adjacent to each other, making the space below w_k available for other subtrees. Label each subtree with the size of the w_k structure inside it, and consider the tree structure with $k/2 - 1$ subtrees $w_k, w_{k-2} ... w_2$. This structure will have an optimal width of $(k + 3) + (k/2 - 2)$ where the first term accounts for the top-most subtree for w_k (in yellow) and the second from the number of edges connecting to remaining subtrees underneath. The greedy heuristic must instead place each subtree, enclosed by the long edges, side-by-side forcing most leaves into distinct columns. Only one pair of leaves per subtree share their column, therefore the width is equal to the number of leaves $(n + 1)/2$ minus the $k/2 - 1$ overlapping leaves (where n is the total number of nodes). In total, there are $n = \sum_{i=0}^{k/2-1}(7 + 2(k - 2i) + 1) - 1 = k^2/2 + 3k$ nodes, from which we find

k. We find that $k \approx \sqrt{2n}$, and therefore the approximation ratio achieved by the greedy heuristic for this tree is $\frac{(n+1)/2-k/2-1}{3k/2+1} \approx \frac{n-\sqrt{2n})}{3\sqrt{2n}} = \Omega(\sqrt{n})$. □

Minimum Area Heuristic. Similar to the greedy heuristic, the *minimum area heuristic* proceeds bottom up from leaves to root and finds the best way of arranging its children assuming their sub-trees have a fixed order. While the greedy heuristic minimizes the area of the tree's bounding rectangle, the minimum area heuristic minimizes the area of the orthogonal y-monotone bounding polygon at the expense of a potential larger total width. We first describe the running time of the heuristic. For each ordering of the children, the minimum width drawing is calculated using the algorithm from Theorem 3.2 and the bounding polygon is calculated by traversing the tree once to find the extreme-most branches, running in $O(n)$ time. We repeat this $O(1)$ times per vertex for a total running time of $O(n^2)$ for bounded degree trees.

Theorem 3.4. *The minimum area heuristic has an approximation ratio of at least* $\Omega(\sqrt{n})$.

Proof. Recall, the structures for w_k as defined in Theorem 3.3 and further constrain it to be a complete binary tree with all its k leaves in horizontal alignment. Recall the subtrees used in Theorem 3.3 and note the subtrees used in this tree instead increase the size of their w_k by 3 each time (with the exception of the first two which have the same w_k). Furthermore each subtree's nodes end immediately before the first node in the subtree two subtrees away, and the latter subtree also has a node aligned with the former's w_k leaves. The leaves in w_k are horizontally aligned with the top node in the next subtree.

Using these definitions Fig. 3.8 demonstrates a tree structure where a minimum area embedding of the two subtrees in yellow makes it impossible for the entire drawing to admit minimum width and minimum area. The heuristic achieves the right embedding for the subtree but fails to choose the right embedding for the two sibling subtrees. Although the optimal's embedding uses a larger area for the combination of both siblings with w_k, it occupies an almost rectangular space resulting in a really small area (and width) increase when adding the next subtree.

Define each subtree by the size of the w_k structure inside it, consider the tree with $2k/3$ subtrees $w_k, w_{k+3}...w_{3k}$. This structure will have an optimal width of $3k+6$. The minimum area heuristic would instead have two subtrees with their w_k on opposite sides and the w_{k+3} overlapping the bottom-most w_k and every next pair of subtrees overlapping in the same way. The total width achieved by the minimum area heuristic would therefore be $\sum_{i=0}^{2k/6} k+6i+5 = 2k^2/6+10k/6+k/6(2k/6+1) = 7k^2/18+11k/6$. The total number of nodes is $n = \sum_{i=0}^{2k/3}(7+2(k-2i)+1)+6+k = 4k^2/3+19k/3-(2k/3)(k/3-1)+6+k = 2k^2+19k/3$, which we can use to find k in terms of n. We find that $k \approx \sqrt{n/2}$, and therefore the approximation ratio achieved by the greedy heuristic for this tree is $\frac{7k^2/18+11k/6}{3k+6} \approx \frac{7k}{54} = \Omega(\sqrt{n})$. $\square$

Hill Climbing and Simulated Annealing The *hill climbing* algorithm is a standard black-box optimization approach. Beginning from an initial configuration, it repeatedly tests small changes and keeps them if they do not hurt the quality of the solution. The quality of the solution is exactly equal to the width of the resulting drawing of the tree, and each change tested corresponds to reordering one node's children.

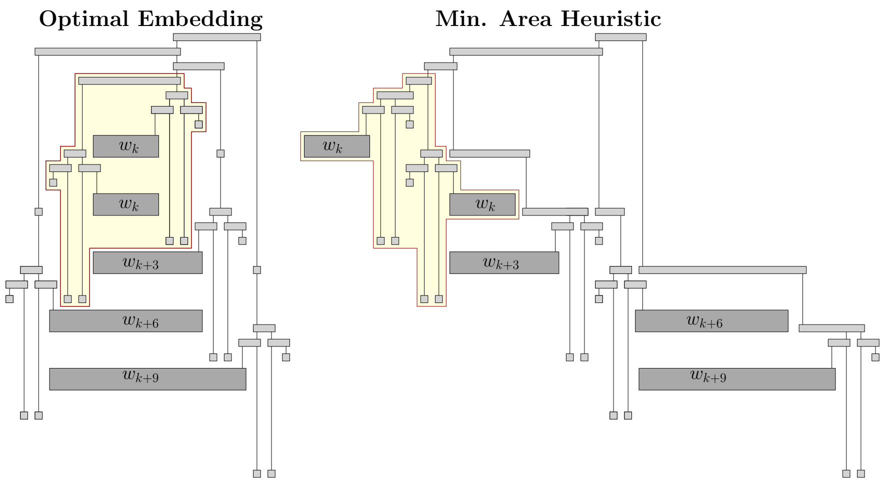


Figure 3.8: Tree structures causing worst case performance for minimum area heuristic.

Our *simulated annealing* algorithm is another black-box optimization approach [80], with the same procedure as hill climbing. The main difference is that changes hurting the solution's quality are kept with probability inversely related to both the difference in quality and the number of steps taken so far. Once the number of steps is large enough, simulated annealing mimics the behavior of the hill climbing algorithm. Compared to hill climbing, simulated annealing has the advantage of not being trapped in a local minimum when a poor starting point was chosen.

Input Ordered The order defined by the description of the tree in its original source file.

3.4.2 Experiments

We evaluate five data sets of real phylogenetic trees obtained from the online phylogenetic tree database TreeBase [85]. The size and compositions of the datasets can be seen in Table 3.1. Each tree in TreeBase originates from a scientific publication, which unfortunately means there are too many for us to list on this dissertation; we instead provide a complete list of the studies associated with phylogenetic trees used in the data sets and complete experiment source code at `github.com/UC-Irvine-Theory/MinWidthPhylogeneticTrees`.

Each dataset is read using Dendropy [97], an open source Python library for phylogenetic computing. Each tree is read with an induced order from the source file, which we will serve as the initial configuration. The datasets are filtered to contain only trees with existing edge-lengths and maximum degree 3. Edge-lengths are normalized into discrete values that preserve nodes' original vertical ordering. For trees with few missing edge-lengths we assume missing edges are of unit length. These normalized datasets are used to evaluate the heuristics and produce easily comparable drawings.

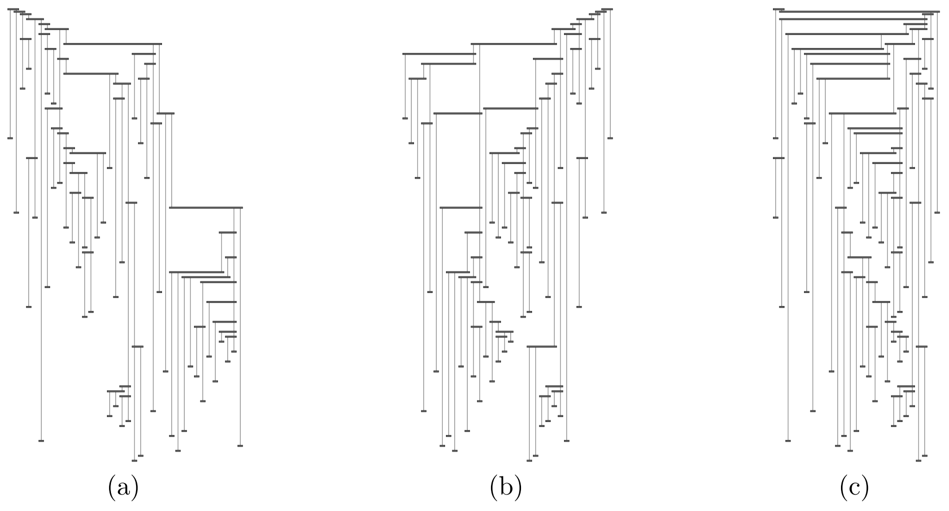


Figure 3.9: Drawings of a tree with 93 nodes (a) Input order, width = 37 (b) Greedy order, width = 33 (c) Simulated annealing order, width = 26

	Data Set Sizes				Avg. Width Diff. from Anneal.				
Name	#Trees	Min	Max	Average	Input	Greedy	MinArea	Hill	Anneal.
Small	363	85	100	92	+26%	+4%	+13%	±0	±0
Medium	1026	188	399	271	+26%	+4%	+12%	-1%	±0
Large	28	2151	3305	2541	+40%	+5%	+9%	-6%	±0
Plant	80	195	3305	754	+57%	+11%	+19%	-3%	±0
Preferred	175	21	2387	192	+21%	+5%	+10%	-1%	±0

Table 3.1: Results. The left side shows the composition of the data sets, while the right side compares the width obtained versus the simulated annealing.

3.4.3 Results

The first thing that stands out from our results is that all of our proposed approaches improve on the original input order. A typical example is shown in Fig. 3.9, where the input width is improved by the greedy heuristic and simulated annealing. Secondly, although the greedy and minimum area heuristics have a bad approximation ratio guarantee, this does not translate to real world trees. As can be seen in Table 3.1, both heuristics performed well for trees regardless of size. For example, in the Preferred data set the greedy heuristic achieved the same width as the Simulated Annealing in 50% of the trees (Fig. 3.10). However, a few cases exist where the greedy heuristic significantly under-performs, such as in Fig. 3.11 and Fig. 3.12.

This is notable for two reasons: the first is that the greedy heuristic produces a drawing 60% wider than hill climbing, and the second is that the greedy heuristic is outperformed by the minimum area heuristic.

Finally, it is clear that blackbox approaches are useful to find small-width drawings as they rarely produce drawings wider than those from the heuristics. However the width decrease achieved by the black box algorithms comes at a cost in running time, since in our implementation they took around 40 times longer to converge on average.

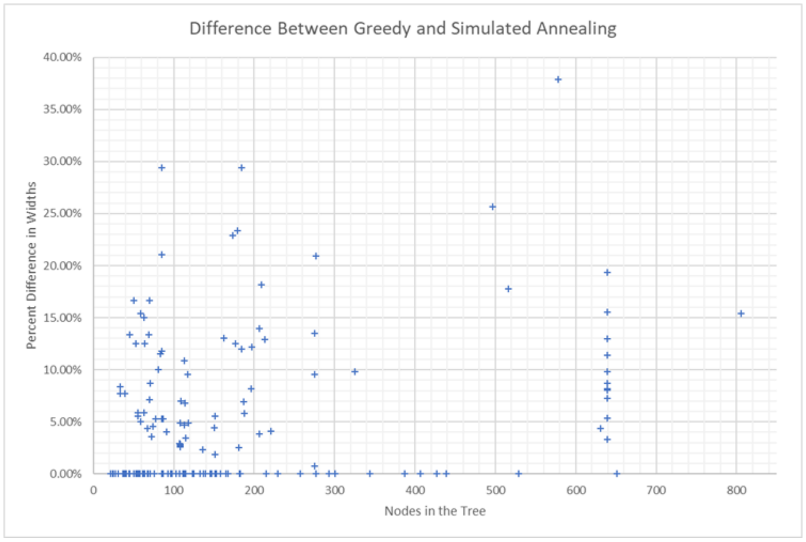


Figure 3.10: Percent width difference between the greedy heuristic and simulated annealing depending on tree size for Preferred dataset.

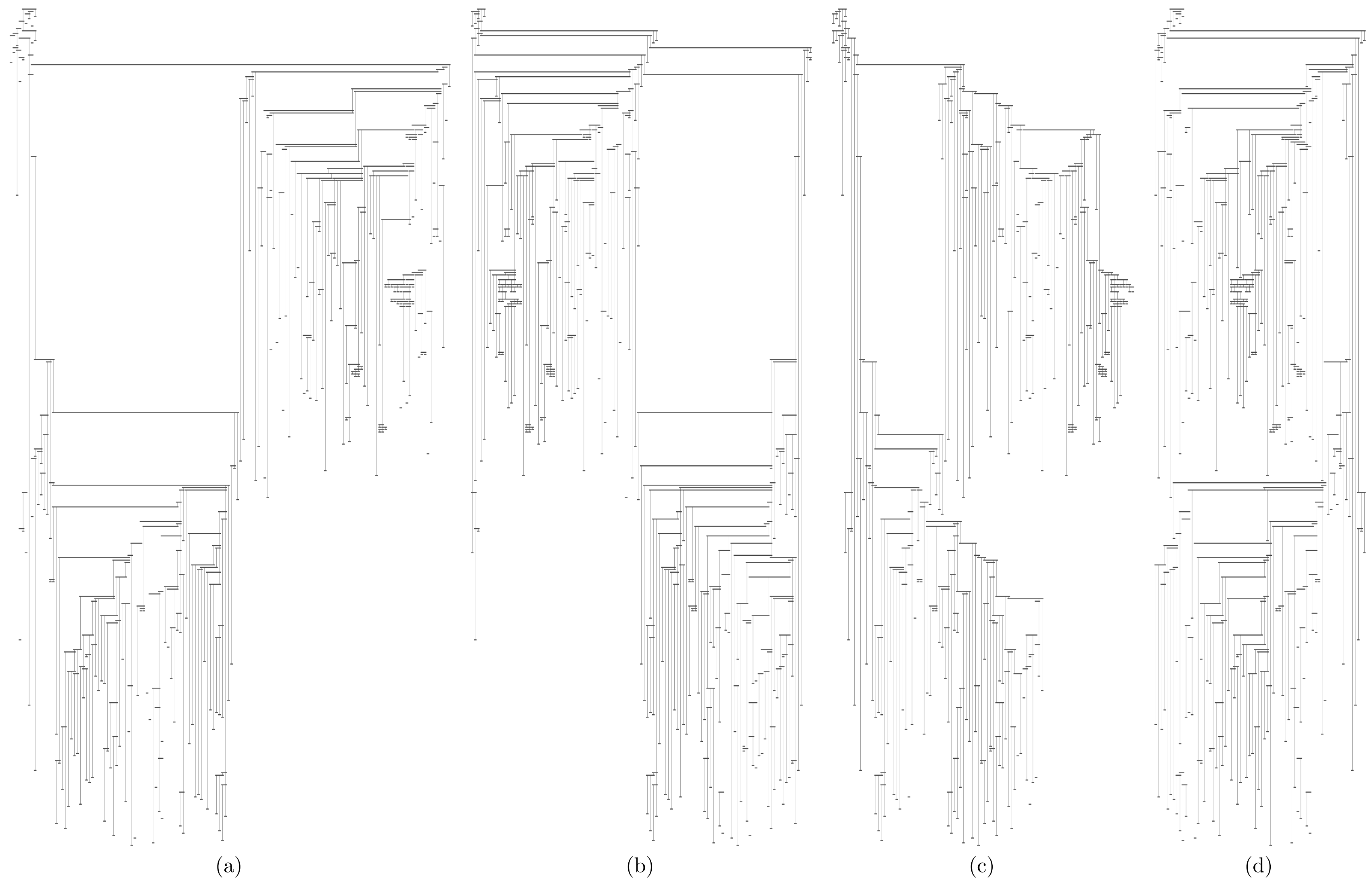


Figure 3.11: Drawings of a Tr91390 which has 390 nodes (a) Input order, width = 145 (b) Greedy order, width = 112 (c) Minimum Area, width = 99 (d) Simulated annealing, width = 69

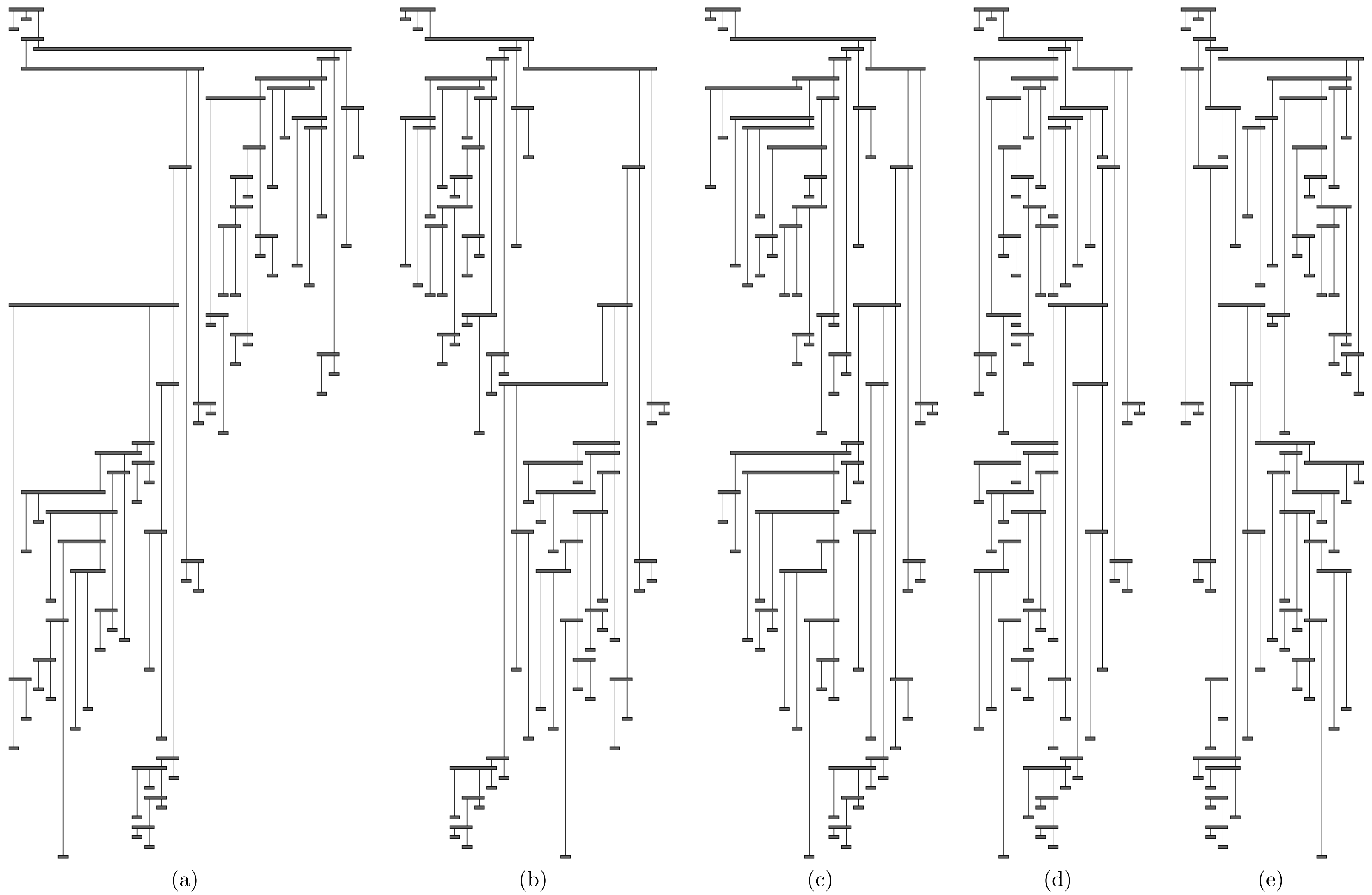


Figure 3.12: Drawings of Tr65395 which has 88 nodes (a) Input order, width = 28 (b) Greedy order, width = 21 (c) Minimum Area, width = 18 (d) Hill climbing, width = 13 (e) Simulated annealing, width = 14

Chapter 4

Layered Drawings: Minimizing Crossings in the Evolving Data Setting

4.1 Introduction

In the traditional Knuthian model [74], an algorithm takes an input, runs for some amount of time, and produces an output. Characterizing an algorithm in this model typically involves providing a function, $f(n)$, such that the running time of the algorithm can be bounded asymptotically as being $O(f(n))$ on average, with high probability (w.h.p.), or in the worst case. Although this has proven to be an extremely useful model for general algorithm design and analysis, there are nevertheless interesting scenarios where it doesn't apply.

4.1.1 The Evolving Data Model

One scenario where the Knuthian model doesn't apply is for applications where the input data is changing while an algorithm is processing it, which has given rise to the *evolving data* model [5]. In this model, input changes are coming so fast that it is hard for an algorithm to keep up, much like Lucy in the classic conveyor belt scene[1] in the TV show, *I Love Lucy*. Thus, rather than producing a single output, as in the Knuthian model, an algorithm in the evolving data model dynamically maintains an output instance over time. The goal of an algorithm in this model is to efficiently maintain its output instance so that it is "close" to the true output at that time. Therefore, analyzing an algorithm in the evolving data model involves defining a distance metric for output instances, parameterizing how input *mutations* occur over time, and characterizing how well an algorithm can maintain its output instance relative to this distance and these parameters for various types of input mutations.

4.1.2 Layered Drawings of Digraphs

A common technique for drawing directed graphs that are hierarchical or nearly hierarchical is layered drawings. These drawings try to capture the transitivity that hierarchies represent by having the edges "flow" in a uniform direction. One of the most popular methods for drawing directed graphs is the *Sugiyama Framework* [96] which separates the drawing into layers. The steps in the Sugiyama framework try to create drawings that meet the following aesthetic considerations [98]:

- Edges should point in a uniform direction
- Short edges are more readable
- Uniformly distributed nodes avoid clutter

[1]E.g., see `https://www.youtube.com/watch?v=8NPzLBSBzPI`.

- Edge crossings obstruct comprehension
- Straight edges are more readable

Achieving all these criteria is in general impossible, since some of them contradict each other. A successive refinement technique is used where each step addresses some of these concerns.

1. **Cycle Removal** Although the graph is assumed to be almost acyclic, there may be some cycles. So that the edges are directed uniformly, any directed cycles are broken by reversing a subset of the edges.
2. **Layer Assignment** Each vertex is assigned to a layer. The *layering* must ensure that every edge is pointed in the same uniform directions. Dummy vertices can be introduced so that "long" edges can be replaced by shorter edges
3. **Crossing Reduction** For each layer an *ordering* of the vertices is computed. The goal is to minimize the number of crossings.
4. **Assign X-coordinate** Each vertex is assigned an x-coordinate so that no overlaps occur and long edges are straightened out.

In this chapter, we will study the crossing reduction step in the evolving data setting, that is a well-studied problem in the Knuthian model. A first observation is that we are only concerned about the relative order of the vertices, not their exact position, it follows that we care only about the order of vertices in the layers. Few systems attempt to optimize the crossings globally, so instead most use a heuristic layer-by-layer sweep method. Such an approach tries to solve the *one sided crossing minimization problem* at each layer.

First we need a few definitions. A *bipartite-graph* is an undirected graph $G = (V, E)$ in which V can be partitioned into two sets V_1 and V_2 and each every edge connects a vertex in V_1 with a vertex in V_2. An *ordering* of V_i is a permutation π_i of the vertices is V_i.

Problem: One Sided Crossing Minimization Problem

Input: Given a bipartite graph $G = (V_1, V_2, E)$ and a permutation π_1 of V_1.

Question: Find a permutation π_2 of V_2 that minimizes the edge crossings in the drawing of G where the vertices of V_1 are placed on a layer L_1 in the order defined by π_1 and the vertices of V_2 are placed on a second layer paralled to L_1.

The minimization problem of the one-sided general crossing minimization problem is NP-hard [48], although fixed-parameter tractable algorithms exist to find a k-crossing [45]. There are many heuristics approaches to this problem [46] [96] a recent survey can be found in [98]. In this chapter, we study a restricted version of the problem. We assume that every vertex

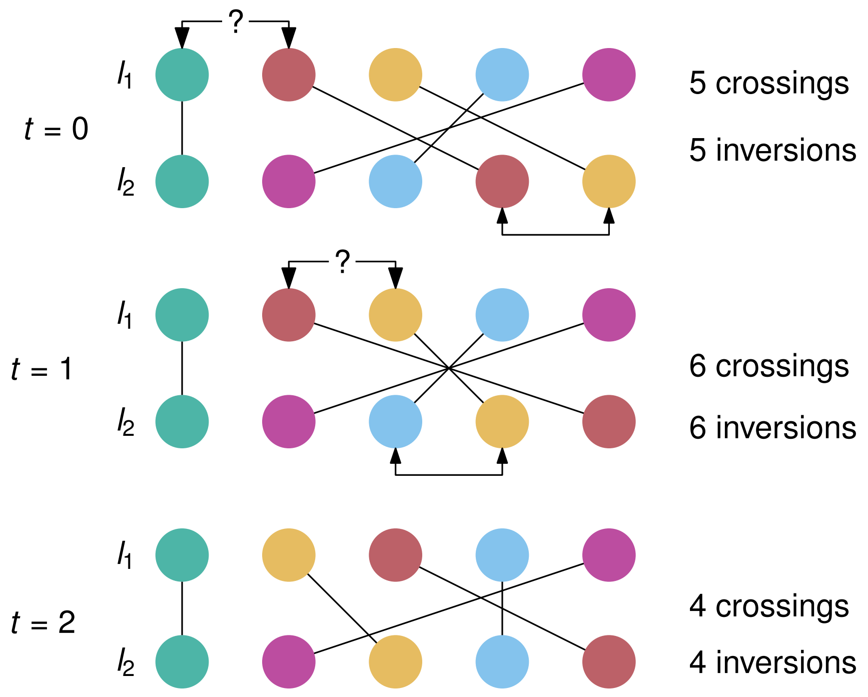


Figure 4.1: Examples of the two layers, l_1, l_2, with the crossings and inversions. In the first step the green and red elements are compared in l_1 and the red and yellow elements are swapped in l_2. In the second step the red and yellow elements are compared and swapped in l_1 and the blue and yellow elements are swapped in l_2.

has degree one, in other words that E is a one-to-one mapping of the vertices in V_1 to the vertices in V_2. The data that is evolving is the order of π_2, and the algorithm is attempting to keep up by reordering π_1, see Fig. 4.1. This restricted setting allows us to consider the problem from another angle, mainly sorting. A crossing in the drawing corresponds exactly to an inversion between the two orderings. Because sorting is the original problem studied in the evolving data setting, we will now mainly discuss the problem in that setting.

4.1.3 Sorting in the Evolving Data Setting

The goal of a sorting algorithm in the evolving data model, then, is to maintain an output order close to the true total order even as it is mutating. For example, the list could be an ordering of political candidates, tennis players, movies, songs, or websites, which is changing as the ranking of these items is evolving, e.g., see [55]. In such contexts, performing a comparison of two elements is considered somewhat slow and expensive, in that it might involve a debate between two candidates, a match between two players, or an online survey or A/B test [44] comparing two movies, songs, or websites. In this model, a comparison-based sorting algorithm would therefore be executing at a rate commensurate with the speed in which the total order is changing, i.e., its mutation rate. Formally, to model this phenomenon, each time an algorithm performs a comparison, we allow for an adversary to perform some changes to the true ordering of the elements.

There are several different adversaries one might consider with respect to mutations that would be anticipated after a sorting algorithm performs a comparison. For example, an adversary (who we call the *uniform* adversary) might choose $r > 0$ consecutive pairs of elements in the true total order uniformly at random and swap their relative order. Indeed, previous work [5] provided theoretical analyses for the uniform adversary for the case when r is a constant. Another type of adversary (who we call the *hot spot* adversary) might choose an element, x, in the total order and repeatedly swap x with a predecessor or successor each time a random "coin flip" comes up "tails," not stopping until the coin comes up "heads."

A natural distance metric to use in this context is the *Kendall tau* distance [71], which counts the number of inversions between a total ordering of n distinct elements and a given list of those elements. That is, a natural goal of a sorting algorithm in the evolving data model is to minimize the Kendall tau distance for its output list over time. In our restricted version of the one sided crossing minimization problem this corresponds exactly to the number of edge crossings in the drawing.

Here, we consider the empirical performance of sorting algorithms in the evolving data model. Whereas previous work looked only at quicksort with respect to theoretical analyses against the uniform adversary, we are interested in this chapter in the "real-world" performance of a variety of sorting algorithms with respect to multiple types of adversaries in the evolving data model. Of particular interest are any experimental results that might be counter-intuitive or would highlight gaps in the theory.

4.1.4 Previous Work on Evolving Data

The evolving data model was introduced by Anagnostopoulos *et al.* [5], who study sorting and selection problems with respect to an evolving total order. They prove that quicksort maintains a Kendall tau distance of $O(n \log n)$ w.h.p. with respect to the true total order, against a uniform adversary that performs a small constant number of random swaps for every comparison made by the algorithm. Furthermore, they show that a batched collection of quicksort algorithms operating on overlapping blocks can maintain a Kendall tau distance of $O(n \log \log n)$ against this same adversary. Additionally they proved that there is a $\Omega(n)$ lower bound to this problem. Later, inspired by the experimental work shown in this chapter, Besa *et al.* [18] give an optimal algorithm (against the same adversary) based on repeated passes of insertion sort. They show that their approach maintains an (optimal) $O(n)$ Kendall tau distance with high probability between a list of n items and an underlying order where a swap in the order occurs after every comparison made by the algorithm.

In addition to this work on sorting, several papers have considered other problems in the evolving data model. Kanade *et al.* [69] study stable matching with evolving preferences. Huang *et al.* [65] present how to select the top-k elements from an evolving list. Zhang and Li [104] consider how to find shortest paths in an evolving graph. Anagnostopoulos *et al.* [6] study (s,t)-connectivity and minimum spanning trees in evolving graphs. Bahmani *et al.* [15] give several PageRank algorithms for evolving graphs and they analyze these algorithms both theoretically and experimentally. We are not aware of previous experimental work on sorting in the evolving data model. We are also not aware of previous work, in general, in the evolving data model for other sorting algorithms or for other types of adversaries.

4.1.5 Our Results

In this chapter, we provide an experimental investigation of sorting in the evolving data model. The starting point for our work is the previous theoretical work [5] on sorting in the evolving data model, which only studies quicksort. Thus, our first result is to report on experiments that address whether these previous theoretical results actually reflect real-world performance.

In addition, we experimentally investigate a number of other classic sorting algorithms to empirically study whether these algorithms lead to good sorting algorithms for evolving data and to study how sensitive they are to parameters in the underlying evolving data model. Interestingly, our experiments provide empirical evidence that quadratic-time sorting algorithms, including bubble sort, cocktail sort, and insertion sort, can outperform more sophisticated algorithms, such as quicksort and even the batched parallel blocked quicksort algorithm of Anagnostopoulos *et al.* [5], in practice. Later this was confirmed by Besa *et al.* [18] for insertion sort, but no theoretical results are known for other quadratic-time sorting algorithms. Furthermore, our results also show that even though these quadratic-time algorithms perform compare-and-swap operations only for *adjacent* elements at each time step, they are nevertheless robust to increasing the rate, r, at which random swaps occur in the underlying list for every comparison done by the algorithm. That is, our results show that these quadratic-time algorithms are robust even in the face of an adversary who can perform many swaps for each of an algorithm's comparisons. Moreover, this robustness holds in spite of the fact that, in such highly volatile situations, each element is, on average, moved more often by random swaps than by the sorting algorithm's operations.

We also introduce the hot spot adversary and study sorting algorithms in the evolving data model with respect to this adversary. Our experiments provide evidence that these sorting algorithms have similar robustness against the hot spot adversary as they do against the uniform adversary. Finally, we show that the constant factors in the Kendall tau distances maintained by classic quadratic-time sorting algorithms appear to be quite reasonable in practice. Therefore, we feel our experimental results are arguably surprising, in that they show the strengths of quadratic-time algorithms in the evolving data model, in spite of the fact that these algorithms are much maligned because of their poor performance in the traditional Knuthian model.

With respect to the organization of the chapter, Section 4.2 formally introduces the evolving sorting model as well as the algorithms we consider. Section 4.3 describes the experiments we performed and presents our results.

4.2 Preliminaries

Let us begin by formally defining the evolving data model for sorting, based on the pioneering work of Anagnostopoulos *et al.* [5]. We assume that there are n distinct elements that belong to a total order relation, "$<$". During each time step, a sorting algorithm is allowed to perform one comparison of a pair of elements and then an adversary is allowed to perform some random swaps between adjacent elements in the true total order. We consider two types of adversaries:

1. The *uniform* adversary. This adversary performs a number, $r > 0$, of swaps between pairs of adjacent elements in the true total order, where these pairs are chosen uniformly at random and independently for each of the r swaps.

2. The *hot spot* adversary. This adversary chooses an element, x, in the total order and then randomly chooses a direction, up or down. The adversary then randomly chooses a bit, b. If this bit is 0, it swaps x with its predecessor (resp., successor), depending on the chosen direction, and repeats this process with a new random bit, b. If $b = 1$, it stops swapping x.

We denote the state of the list the algorithm maintains at time t with ℓ_t and the state of the unknown true ordering with ℓ'_t. If the time step is clear from the context or is the most current step, then we may drop the subscript, t. The main type of adversary that we consider is the same as that considered in Anagnostopoulos *et al.* [5]; namely, what we are calling the uniform adversary. Note that with this adversary, after each comparison, a uniformly random adjacent pair of elements in ℓ' exchange positions and this process is repeated independently for a total of $r > 0$ swaps. We call each such change to ℓ' a *swap* mutation. With respect to the hot spot adversary, we refer to the change made to ℓ' as a *hot spot* mutation, i.e., where an element is picked uniformly at random and flips an unbiased coin to pick left or right, and then an unbiased coin is flipped until it comes up heads and the element is swapped in the chosen direction that many times.

Note that with either adversary, a sorting algorithm cannot hope to correctly maintain the true total order at every step, for at least the reason that it has no knowledge of how the most recent mutation affected the underlying order. Instead, a sorting algorithm's goal is to maintain a list of the n elements that has a small Kendall tau distance, which counts the number of inversions[2] relative to the underlying total order.

We abuse the names of the classical sorting algorithms to refer to evolving sorting algorithms that repeatedly run that classical algorithm. For instance, the insertion sort evolving sorting algorithm repeatedly runs the classical in-place insertion sort algorithm. We refer to each individual run of a classical sorting algorithm as a *round*.

In this chapter, we consider several classical sorting algorithms, which are summarized in simplified form below (see [36, 56] for details and variations on these algorithms):

- *Bubble sort*. For $i = 1, \ldots, n-1$, repeatedly compare the elements at positions i and $i+1$, swapping them if they are out of order. Repeat $n-1$ times.
- *Cocktail sort*. For $i = 1, \ldots, n-1$, repeatedly compare the elements at positions i and $i+1$, swapping them if they are out of order. Then do the same for $i = n-1, \ldots, 1$. Repeat $(n-2)/2$ times.
- *Insertion sort*. For $i = 2, \ldots, n$, compare the element, x, at position i with its predecessor, swapping them if they are out of order. Then repeat this process again with x (at its new position) until it reaches the beginning of the list or isn't swapped. Repeat $n-2$ times.
- *Quicksort*. Randomly choose a *pivot*, x, in the list and divide the list in place as the

[2]Recall that an *inversion* is a pair of elements u and v such that u comes before v in a list but $u > v$.

elements that are less than or equal to x and the elements that are greater than x. Recursively process each sublist if it has at least two elements.

We also consider the following algorithm due to Anagnostopoulos *et al.* [5]:

- *Block sort*.[3] Divide the list into overlapping blocks of size $O(\log n)$. Alternate between steps of quicksort on the entire list and steps of quicksort on each block.

There are no theoretical results on the quality of bubble or cocktail sort in the evolving data model. Besa *et al.* [18] proved that insertion sort maintains $O(n)$ inversions w.h.p. if $r = 1$. For larger values of r nothing is known. Anagnostopoulos *et al.* [5] showed that quicksort achieves $\Theta(n \log n)$ inversions w.h.p. for any small constant, r, of swap mutations. Anagnostopoulos *et al.* [5] also showed that block sort achieves $\Theta(n \log \log n)$ inversions w.h.p. for any small constant rate, r, of swap mutations.

These algorithms can be classified in two ways. First, they can be separated into the worst-case quadratic-time sorting algorithms (bubble sort, cocktail sort, and insertion sort) and the more efficient algorithms (quicksort and block sort), with respect to their performance in the Knuthian model. Second, they can also be separated into two classes based on the types of comparisons they perform and how they update ℓ. Bubble sort, cocktail sort, and insertion sort consist of compare-and-swap operations of adjacent elements in ℓ and additional bookkeeping, while quicksort and block sort perform comparisons of elements that are currently distant in ℓ and only update ℓ after the completion of some subroutines.

During the execution of these algorithms, the Kendall tau distance does not converge to a single number. For example, the batch update behavior of quicksort causes the Kendall tau distance to oscillate each time a round of quicksort finishes. Nevertheless, for all of the

[3]The quicksorts of the entire list guarantee that no element is more than $O(\log n)$ positions from its proper location in the true sorted order. The quicksorts of the blocks account for elements of the list drifting away from their original positions.

algorithms, the Kendall tau distance empirically reaches a final repetitive behavior. We call this the *steady behavior* for the algorithm and judge algorithms by the average number of inversions once they reach their steady behavior. We call the time it takes an algorithm to reach its steady behavior its *convergence time*. Every algorithm's empirical convergence time is at most n^2 steps.

4.3 Experiments

The main goal of our experimental framework is to address the following questions:

1. Do quadratic-time algorithms actually perform as well as or better than quicksort variants on evolving data using swap mutations, e.g., for reasonable values of n?
2. What is the nature of the convergence of sorting algorithms on evolving data, e.g., how quickly do they converge and how much do they oscillate once they converge?
3. How robust are the algorithms to increasing the value of r for swap mutations?
4. How much does an algorithm's convergence behavior and steady behavior depend on the list's initial configuration (e.g., randomly scrambled or initially sorted)?
5. How robust are the algorithms to a change in the mutation behavior, such as with the hot spot adversary?
6. What is the fraction of random swaps that actually improve Kendall tau distance?

We present results below that address each of these questions.

4.3.1 Experimental Setup

We implemented the various algorithms that we study in C++11 and our software is available on Github.[4] Randomness for experiments was generated using the C++ default_engine and in some cases using the C random engine. In the evolving data model, each time step corresponds to one comparison step in an algorithm, which is followed by a mutation performed by an adversary. Therefore, our experiments measure time by the number of comparisons performed. Of course, all the algorithms we study are comparison-based; hence, their number of comparisons correlates with their wall-clock runtimes. To measure Kendall tau distances, we sample the number of inversions every $n/20$ comparisons, where n is the size of a list. We terminate a run after n^2 time steps and well after the algorithm has reached its steady behavior. The experiments primarily used random swap mutations; hence, we omit mentioning the mutation type unless we are using hot spot mutations.

Anagnostopoulos *et al.* [5] does not give an exact block size for their block sort algorithm except in terms of several unknown constants. In our block sort implementation, the block size chosen for block sort is the first even number larger than $10 \ln n$ that divides n. Because all of the n in our experiments are multiples of 1000, the block size is guaranteed to be between $10 \ln n$ and $100 \ln n$.

4.3.2 General Questions Regarding Convergence Behavior

We begin by empirically addressing the first two questions listed above, which concern the general convergence behavior of the different algorithms. Fig. 4.2 shows Kendall tau distance achieved by the various algorithms we studied as a function of the algorithm's execution time, against the uniform adversary (i.e., with random swap mutations), for the case when $r = 1$ and $n = 1000$ starting from a uniformly shuffled list.

[4]See code at `https://github.com/UC-Irvine-Theory/SortingEvolvingData`.

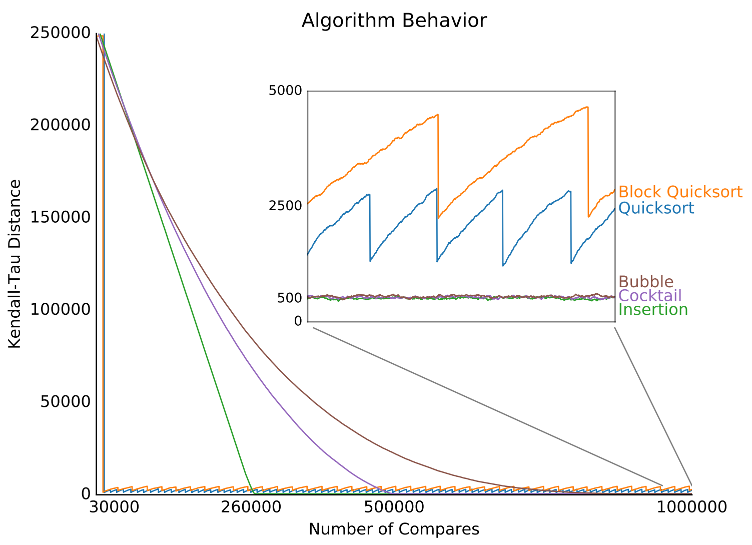


Figure 4.2: Behavior of the algorithms starting from a shuffled list. The plot shows Kendall tau distance as a function of an algorithm's execution, i.e., number of comparisons, with random swap mutations for $r = 1$. We also show an enlarged portion of the tail-end steady behaviors.

The quadratic time algorithms run continuous passes on the list and every time they find two elements in the wrong order they immediately swap them. That is, they are local in scope, at each step fixing inversions by swapping adjacent elements. They differ in their approach, but once each such algorithm establishes a balance between the comparisons performed by the algorithm and the mutations performed by the uniform adversary, their Kendall tau numbers don't differ significantly. These algorithms have very slow convergence because they only compare adjacent elements in the list and so fix at most one inversion with each comparison. The Kendall tau behavior of the quicksort algorithms, on the other hand, follow an oscillating pattern of increasing Kendall tau distance until a block (or recursive call) is finished, at which point ℓ is quickly updated, causing a large decrease in Kendall tau distance.

As can be seen, the convergence behavior of the algorithms can be classified into two groups. The first group consists of the two quicksort variants, which very quickly converge to steady behaviors that oscillate in a small range. The second group consists of the quadratic-time algorithms, which converge more slowly, but to much smaller Kendall tau distances and with no clear oscillating behavior. More interestingly, the quadratic-time algorithms all converge to the same tight range and this range of values is better than the wider range of Kendall tau distances achieved by the quicksort algorithms. Thus, our first experimental result already answers our main question, namely, that these quadratic-time algorithms appear to be optimal for sorting evolving data and this behavior is actually superior in the limit to the quicksort variants.

Given enough time, however, all three quadratic-time algorithms maintain a consistent Kendall tau distance in the limit. Of the three quadratic-time algorithms, the best performer is insertion sort, followed by cocktail sort and then bubble sort. The worst algorithm in our first batch of experiments was block quicksort. This may be because $n = 1000$ is too small for the theoretically proven $O(n \log \log n)$ Kendall tau distance to hold.

4.3.3 Convergence Behavior as a Function of r

Regardless of the categories, after a sufficient number of comparisons, all the algorithms empirically reach a *steady behavior* where the distance between ℓ and ℓ' follows a cyclic pattern over time. This steady behavior depends on the algorithm, the list size, n, and the number of random swaps, r, per comparison, but it is visually consistent across many different runs and starting configurations.

Our next set of experiments, therefore, are concerned with studying convergence behavior as a function of n and r. We show in Fig. 4.3 the convergence values comparing insertion sort and quicksort, as a ratio, K/n, of the steady-state Kendall tau distance, K (averaged across multiple samples once an algorithm has reached its steady behavior), and the list size, n.

As can be seen from the plots, for these values of r, insertion sort consistently beats quicksort in terms of the Kendall tau distance it achieves, and this behavior is surprisingly robust even as n grows. Moreover, all of the quadratic-time algorithms that we studied achieved

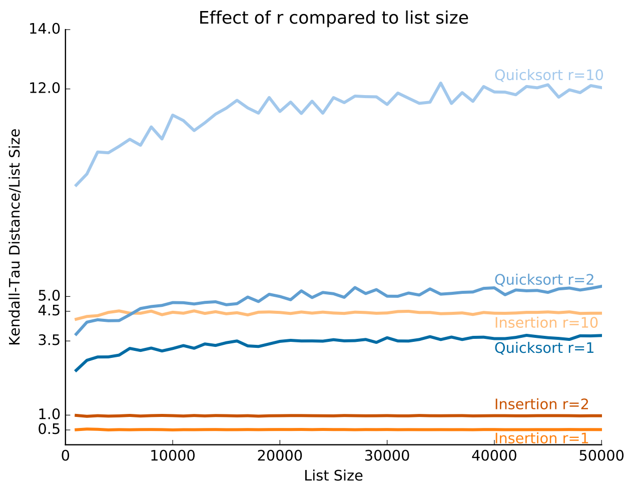


Figure 4.3: Convergence ratios as a function of list size, n, and number of random swaps, r, per comparison. The vertical axis shows the ratio, K/n, where K is the average Kendall tau value (number of inversions) in the steady behavior, and the horizontal axis shows the number of elements, n. The curves show the behaviors of insertion sort and quicksort for $r \in \{1, 2, 10\}$.

similar Kendall-tau-to-size ratios that were consistently competitive with both quicksort and block sort. In Table 4.1, we show the ratios of the number of inversions to list size for various values of r, with respect to the uniform adversary and multiple algorithms. Note that the ratios grow slowly as a function of r and that the quadratic time algorithms are better than the quicksort variants for values of r up to around 50. After that threshold, standard quicksort tends to perform better than the quadratic-time algorithms, but the quadratic-time algorithms nevertheless still converge and perform reasonably well. Interestingly, the quadratic algorithms still beat block sort even for these large values of r.

Our results show that for values of r larger than 50, the quicksort variants tend to perform better than the quadratic-time algorithms, but the quadratic time algorithms nevertheless still converge to reasonable ratios. We show in Table 4.2 specific convergence values for different values of r, from 0 to 10, for insertion sort and quicksort. As can be seen, this table highlights the expected result that quicksort reaches its steady behavior much more quickly than insertion sort. Thus, this table provides empirical evidence supporting a hybrid algorithm where one first performs a quicksort round and then repeatedly performs insertion

r	Insertion	Cocktail	Bubble	Quicksort	Block Quicksort
1	0.51	0.54	0.54	2.17	4.03
2	0.98	0.98	1.13	3.40	5.78
3	1.45	1.42	1.64	4.24	7.19
4	1.84	1.76	2.17	4.51	8.58
5	2.28	2.04	2.69	5.03	9.85
6	2.72	2.46	3.05	5.83	10.11
7	3.16	2.83	3.40	6.62	11.39
8	3.49	3.20	3.89	7.15	12.06
9	4.03	3.63	4.50	7.04	12.74
10	4.37	3.87	4.96	7.45	14.09
11	4.64	4.09	5.58	7.44	14.60
12	5.07	4.61	5.79	8.12	15.91
13	5.32	4.92	6.17	7.96	15.89
14	5.91	5.14	6.73	9.35	16.36
15	6.21	5.76	6.94	9.75	17.55
16	6.52	5.98	7.33	9.91	17.54
17	7.06	6.05	7.74	10.03	18.21
18	7.56	6.43	8.13	10.02	18.59
19	7.79	6.94	8.56	10.38	19.73
20	8.25	7.51	8.68	10.89	20.93
...					
40	14.98	13.53	17.12	15.05	25.18
41	15.32	14.27	17.86	15.19	25.24
42	15.79	14.11	17.77	15.46	25.00
43	16.26	14.36	17.79	15.34	26.85
44	16.42	14.74	18.05	15.79	26.39
45	16.45	15.11	18.73	15.60	27.81
46	17.09	15.40	19.31	16.09	27.47
47	17.37	15.70	19.73	16.36	27.32
48	17.42	16.02	19.97	16.21	28.55
49	17.87	16.22	20.08	16.46	28.08
50	18.55	16.57	20.66	16.72	29.23
...					
100	32.67	30.36	35.18	23.83	43.18
256	65.20	61.30	66.20	38.10	74.53

Table 4.1: Ratio of inversions relative to list size for different values of r

sort rounds after that.

r	Insertion	Quicksort
0	500000	12000
1	510000	16000
2	513000	16000
3	516000	15000
4	516000	16000
5	521000	16000
6	523000	16000
7	521000	17000
8	525000	17000
9	524000	17000
10	527000	16000

Table 4.2: Number of comparisons needed to converge to steady behavior for different values of r.

4.3.4 Starting Configurations

The quadratic time algorithms all approach their steady behavior in a similar manner, namely, at an approximately constant rate attenuated by r. Thus, we also empirically investigated how long each algorithm requires to reach a steady behavior starting from a variety of different start configurations.

Because both quicksort and block sort begin with a quicksort round, their Kendall tau distance drops quickly after $O(n \log n)$ comparisons. The other algorithms require a number of comparisons proportional to the initial distance from the steady-state value. For example, see Table 4.1, which shows that insertion sort's steady behavior when $r = 1$ is roughly $n/2$ inversions. When the initial state of a list has I inversions, the number of comparisons that insertion sort requires to reach the steady behavior is approximately $4|I - n/2|$. Thus, if the list is initially sorted, insertion sort will take approximately $2n$ comparisons to reach its steady behavior. Moreover, increasing r does not seem to affect the convergence rate significantly. Fig. 4.4 shows a plot illustrating the convergence behavior of insertion sort and quicksort for a variety of different starting configurations.

Our experiments show that, as expected, the convergence of insertion sort is sensitive to the

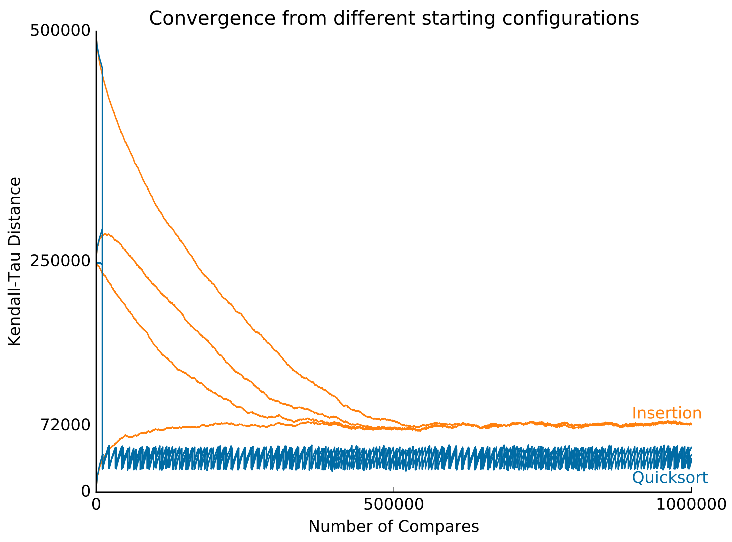


Figure 4.4: Even for a large $r = 256$ over only 1000 elements the algorithms quickly converge to a steady behavior. The plots show the Kendal tau distances achieved for insertion sort and quicksort for four starting configurations of increasing complexity: (i) initially sorted, (ii) random shuffle, (iii) half cyclical shift of a sorted list, and (iv) a list in reverse order.

starting configuration, whereas quicksort is not. Primed with this knowledge, these results justify our starting from a sorted configuration in our subsequent experiments, so as to explore convergence values without having to wait as long to see the steady behavior. Still, it is surprising that the quadratic algorithms converge at all for $r = 256$ in these experiments, since for each inversion fixed by a quadratic algorithm the adversary gets to swap 256 pairs in a list of 1000 elements.

In general, these early experiments show that, after converging, the quadratic time algorithms perform significantly better than the more efficient algorithms for reasonable values of r and that they are competitive with the quicksort values even for larger values of r. But in an initial state with many inversions, the more efficient algorithms require fewer compares to quickly reach a steady behavior. Thus, to optimize convergence time, it is best to run an initial round of quicksort and then switch to repeated rounds consisting of one of the quadratic-time algorithms.

4.3.5 Hot spot mutations

Recall that hot spot mutations simulate an environment in which, instead of a pair of elements swapping with each other, an element changes its rank based on a geometric distribution. Fig. 4.5 shows the convergence behavior of the various algorithms against the hot spot adversary. Comparing Fig. 4.5 to Fig. 4.2, we see that the quadratic algorithms are twice as affected by hot spot mutations as by uniform mutations, although the total number of adversarial swaps is the same in expectation.

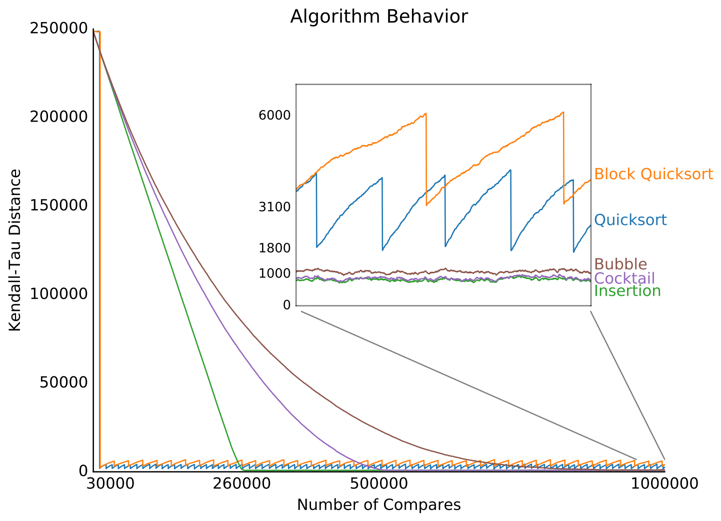


Figure 4.5: Impact of hotspot mutations. We plot Kendall tau distance as a function of an algorithm's number of comparisons. We also show an enlargement of the tail-end steady behaviors.

A possible explanation for this behavior is that a large change in rank of a single element in a hot spot mutation can block a large number of other inversions from being fixed during a round of a quadratic algorithm. For example, in insertion sort, during a round, the element at each position, $0 \le i \le n$, is expected to be moved to its correct position with respect to $0, \ldots, i-1$ in l. To do so, an element x is swapped left until it reaches a smaller element y. But if y has mutated to become smaller, then all the elements left of y in l, which remain larger than x, will stay inverted with respect to x until the end of the round (unless some other mutation fixes some of them). Bubble and cocktail sort have a similar problem. An element

x which mutates can block other local, smaller inversions involving elements in between x's starting and ending position. When these inversions are not be fixed, hot spot mutations make each pass of these algorithms coarser. Batch algorithms on the other hand are not affected as strongly by hot spot mutations, because their behavior depends on non-local factors such as pivot selection and the location at which the list was partitioned. Therefore, the movement of a single element has a smaller effect on their behavior. Thus, we find it even more surprising that the quadratic algorithms still beat the quicksort variants even for the hot spot adversary (albeit by a lesser degree than the amount they beat the quicksort variants for the uniform adversary).

4.3.6 Beneficial Swaps Performed by an Adversary

Note that our quadratic-time algorithms compare only adjacent elements, so they can only fix one inversion at each step. Therefore, they will not reach a steady state until the random swaps fix inversions almost as often as they introduce inversions. Fig. 4.6 shows that the proportion of good swaps (those that fix inversions) to bad swaps (those that introduce inversions) approaches 1 as r increases. This behavior might be useful to exploit, therefore, in future work that would provide theoretical guarantees for the performance of bubble sort and cocktail sort in the evolving data model.

4.4 Conclusion

We studied a restricted case of the one-sided crossing minimization problem in the evolving data model and related it to sorting. Thus, we have given an experimental evaluation of sorting in the evolving data model. Our experiments provide empirical evidence that, in this model, quadratic-time algorithms can be superior to algorithms that are otherwise faster in

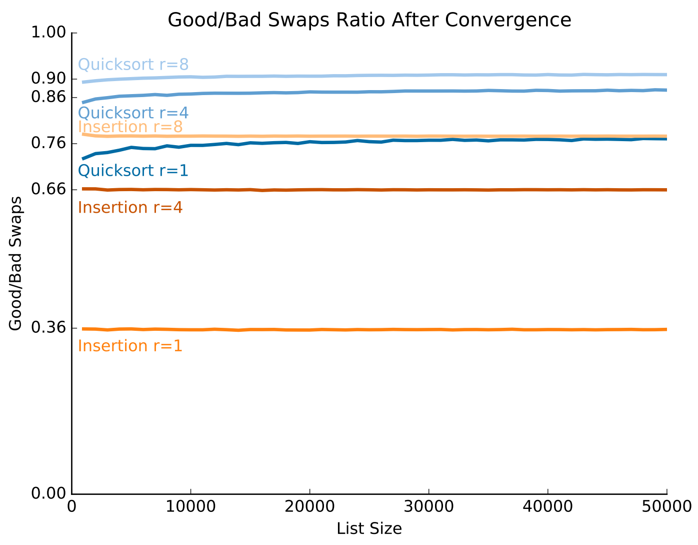


Figure 4.6: As r increases, the number of inversions increases, but so does the number of beneficial mutations.

the traditional Knuthian model. In fact, later work proved that Insertion Sort does behave optimally for a random adversary and swap rate of 1 [18]. We have also studied a number of additional questions regarding sorting in the evolving data model. Given the surprising nature of many of our results, it would be interesting in the future to empirically study algorithms for other problems besides sorting in the evolving data model. Alternatively, it would also be interesting to provide theoretical analyses for more of the experimental phenomena that we observed for sorting in the evolving data model, such as the performance of algorithms against the hot spot adversary.

Bibliography

[1] 111Alleskönner. Political system of the united states, 2004. License: Creative Commons Attribution-Share Alike 3.0 Germany (https://creativecommons.org/licenses/by-sa/3.0/de/deed.en).

[2] Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, and T. B. Schardl. Who needs crossings? Hardness of plane graph rigidity. In *International Symposium on Computational Geometry (SoCG)*, volume 51 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl, 2016.

[3] M. J. Alam, M. Dillencourt, and M. T. Goodrich. Capturing Lombardi flow in orthogonal drawings by minimizing the number of segments. In *Graph Drawing*, volume 9801 of *LNCS*, pages 608–610. Springer, 2016.

[4] M. J. Alam, M. Dillencourt, and M. T. Goodrich. Capturing Lombardi flow in orthogonal drawings by minimizing the number of segments. *arXiv preprint arXiv:1608.03943*, 2016.

[5] A. Anagnostopoulos, R. Kumar, M. Mahdian, and E. Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011. Special issue on selected papers from 36th International Colloquium on Automata, Languages and Programming (ICALP 2009).

[6] A. Anagnostopoulos, R. Kumar, M. Mahdian, E. Upfal, and F. Vandin. Algorithms on evolving graphs. In *3rd ACM Innovations in Theoretical Computer Science Conference (ITCS)*, pages 149–160, 2012.

[7] P. Angelini, G. Da Lozzo, M. D. Bartolomeo, V. D. Donato, M. Patrignani, V. Roselli, and I. G. Tollis. Algorithms and bounds for L-drawings of directed graphs. *International Journal of Foundations of Computer Science*, 29(04):461–480, 2018.

[8] P. Angelini, G. Da Lozzo, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. Beyond level planarity. In *Graph Drawing*, volume 9801 of *LNCS*, pages 482–495. Springer, 2016.

[9] P. Angelini, G. Da Lozzo, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. Intersection-link representations of graphs. *Journal of Graph Algorithms and Applications*, 21(4):731–755, 2017.

[10] P. Angelini, G. Da Lozzo, G. Di Battista, F. Frati, and V. Roselli. The importance of being proper: (in clustered-level planarity and T-level planarity). *Theoretical Computer Science*, 571:1–9, 2015.

[11] P. Angelini, P. Eades, S. Hong, K. Klein, S. G. Kobourov, G. Liotta, A. Navarra, and A. Tappini. Turning cliques into paths to achieve planarity. In *Graph Drawing 2018*, volume 11282 of *LNCS*, pages 67–74. Springer, 2018.

[12] P. Angelini, G. D. Lozzo, G. D. Battista, V. D. Donato, P. Kindermann, G. Rote, and I. Rutter. Windrose planarity: Embedding graphs with direction-constrained edges. *ACM Transactions on Algorithms*, 14(4):54:1–54:24, Sept. 2018.

[13] C. Bachmaier, F. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *Journal on Graph Algorithms and Applcations*, 9(1):53–97, 2005.

[14] C. Bachmaier, U. Brandes, and B. Schlieper. Drawing phylogenetic trees. In *Algorithms and Computation*, pages 1110–1121. Springer, 2005.

[15] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal. Pagerank on an evolving graph. In *18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 24–32, 2012.

[16] M. J. Bannister and D. Eppstein. Hardness of approximate compaction for nonplanar orthogonal graph drawings. In *Graph Drawing*, pages 367–378, 2012.

[17] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(6):476–497, 1994.

[18] J. J. Besa, W. E. Devanny, D. Eppstein, M. T. Goodrich, and T. Johnson. Optimally sorting evolving data. *arXiv preprint arXiv:1805.03350*, 2018.

[19] S. N. Bhatt and S. S. Cosmadakis. The complexity of minimizing wire lengths in VLSI layouts. *Information Processing Letters*, 25(4):263–267, 1987.

[20] T. Biedl and D. Mondal. On upward drawings of trees on a given grid. In *Graph Drawing*, pages 318–325. Springer, 2017.

[21] D. Bienstock and C. L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19(1):79–94, 1989.

[22] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.

[23] C. Binucci, W. Didimo, and F. Giordano. Maximum upward planar subgraphs of embedded planar digraphs. *Computional Geometry*, 41(3):230–246, 2008.

[24] C. Binucci, W. Didimo, and M. Patrignani. Upward and quasi-upward planarity testing of embedded mixed graphs. *Theoretical Computer Science*, 526:75–89, 2014.

[25] R. B.J. Benson, H. Ketchum, D. Naish, and L. E. Turner. A new leptocleidid (sauropterygia, plesiosauria) from the vectis formation (early barremianearly aptian; early cretaceous) of the isle of wight and the evolution of leptocleididae, a controversial clade. *Journal of Systematic Palaeontology*, 11, 02 2013.

[26] A. Boc, A. B. Diallo, and V. Makarenkov. T-REX: a web server for inferring, validating and visualizing phylogenetic trees and networks. *Nucleic Acids Research*, 40(W1):W573–W579, 06 2012.

[27] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer System Science*, 13(3):335–379, 1976.

[28] U. Brandes and B. Pampel. Orthogonal-ordering constraints are tough. *Journal of Graph Algorithms and Applications*, 17(1):1–10, 2013.

[29] G. Brückner and I. Rutter. Partial and constrained level planarity. In *Simposium on Discrete Algorithms SODA*, pages 2000–2011. SIAM, 2017.

[30] W. Brunner and M. Matzeder. Drawing ordered $(k-1)$-ary trees on k-grids. In *Graph Drawing*, volume 6502 of *LNCS*, pages 105–116. Springer, 2011.

[31] S. Cabello, E. D. Demaine, and G. Rote. Planar embeddings of graphs with specified edge lengths. In *Graph Drawing*, volume 2912 of *LNCS*, pages 283–294. Springer, 2004.

[32] S. F. Carrizo. Phylogenetic trees: An information visualisation perspective. In *Proceedings of the Second Conference on Asia-Pacific Bioinformatics - Volume 29*, APBC '04, pages 315–320, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[33] T. M. Chan. Tree drawings revisited. *Discrete & Computational Geometry*, pages 1–22, 2018.

[34] T. M. Chan, M. T. Goodrich, S. R. Kosaraju, and R. Tamassia. Optimizing area and aspect ratio in straight-line orthogonal tree drawings. *Computational Geometry*, 23(2):153 – 162, 2002.

[35] S. Chaplick, M. Chimani, S. Cornelsen, G. Da Lozzo, M. Nöllenburg, M. Patrignani, I. G. Tollis, and A. Wolff. Planar l-drawings of directed graphs. In *Graph Drawing*, volume 10692 of *LNCS*, pages 465–478. Springer, 2017.

[36] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[37] G. Da Lozzo, G. Di Battista, F. Frati, and M. Patrignani. Computing nodetrix representations of clustered graphs. *Journal of Graph Algorithms and Applications*, 22(2):139–176, 2018.

[38] G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed size. In *Graph Drawing*, pages 297–310. Springer, 1999.

[39] G. Di Battista and E. Nardelli. Hierarchies and planarity theory. *IEEE Transactions Systems, Man, and Cybernetics*, 18(6):1035–1046, 1988.

[40] G. Di Battista and R. Tamassia. On-line graph algorithms with SPQR-trees. In *International Colloquium on Automata, Language and Programming ICALP*, volume 443 of *LNCS*, pages 598–611. Springer, 1990.

[41] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996.

[42] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25:956–997, 1996.

[43] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *J. Graph Algorithms Appl.*, 9(1):31–52, 2005.

[44] E. Dixon, E. Enos, and S. Brodmerkle. A/B testing of a webpage, 2011. US Patent 7,975,000.

[45] V. Dujmović, H. Fernau, and M. Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. In *International Symposium on Graph Drawing*, pages 332–344. Springer, 2003.

[46] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin*, 21:89–98, 1986.

[47] P. Eades and N. C. Wormald. Fixed edge-length graph drawing is NP-hard. *Discrete Applied Mathematics*, 28(2):111–134, 1990.

[48] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.

[49] J. Felsenstein. *Inferring phylogenies*, volume 2. Sinauer associates Sunderland, MA, 2004.

[50] Q. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In *European Symposium on Algorithms*, volume 979 of *LNCS*, pages 213–226. Springer, 1995.

[51] F. Frati. Straight-line orthogonal drawings of binary and ternary trees. In *Graph Drawing*, volume 4875 of *LNCS*, pages 76–87. Springer, 2008.

[52] A. Garg, M. T. Goodrich, and R. Tamassia. Planar upward tree drawings with optimal area. *International Journal of Computational Geometry and Applications*, 06(03):333–356, 1996.

[53] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[54] E. D. Giacomo, G. Liotta, M. Patrignani, and A. Tappini. Nodetrix planarity testing with small clusters. In *Graph Drawing*, volume 10692 of *LNCS*, pages 479–491. Springer, 2017.

[55] M. E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 48(3):377–394, 1999.

[56] M. T. Goodrich and R. Tamassia. *Algorithm Design and Applications*. Wiley Publishing, 1st edition, 2014.

[57] A. Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31(4):167–173, 1989.

[58] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991.

[59] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In *Graph Drawing (GD '00)*, volume 1984 of *LNCS*, pages 77–90, 2001.

[60] F. Harary and G. Prins. The block-cutpoint-tree of a graph. *Publicationes Mathematicae Debrecen*, 13:103–107, 1966.

[61] N. Henry, J. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1302–1309, 2007.

[62] D. Holten and J. J. Van Wijk. A user study on visualizing directed edges in graphs. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 2299–2308. ACM, 2009.

[63] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

[64] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[65] Q. Huang, X. Liu, X. Sun, and J. Zhang. Partial sorting problem on evolving data. *Algorithmica*, pages 1–24, 2017.

[66] J. P. Huelsenbeck, F. Ronquist, R. Nielsen, and J. P. Bollback. Bayesian inference of phylogeny and its impact on evolutionary biology. *Science*, 294(5550):2310–2314, 2001.

[67] D. H. Huson and C. Scornavacca. Dendroscope 3: An interactive tool for rooted phylogenetic trees and networks. *Systematic Biology*, 61(6):1061–1067, 09 2012.

[68] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In *Graph Drawing*, volume 1547 of *LNCS*, pages 224–237. Springer, 1998.

[69] V. Kanade, N. Leonardos, and F. Magniez. Stable Matching with Evolving Preferences. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 60 of *LIPIcs*, pages 36:1–36:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[70] M. Kaufmann and D. Wagner. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Nodes in Computer Science*. Springer, 2003.

[71] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[72] S. K. Kim. Simple algorithms for orthogonal upward drawings of binary and ternary trees. In *Canadian Conference on Computational Geometry (CCCG)*, pages 115–120, 1995.

[73] B. Klemz and G. Rote. Ordered level planarity, geodesic planarity and bi-monotonicity. In *Graph Drawing '17*, volume 10692 of *LNCS*, pages 440–453. Springer, 2017.

[74] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Pearson Education, 2nd edition, 1998.

[75] E. M. Kornaropoulos and I. G. Tollis. Overloaded orthogonal drawings. In *International Symposium on Graph Drawing*, pages 242–253. Springer, 2011.

[76] J. Kratochvíl. A special planar satisfiability problem and a consequence of its np-completeness. *Discrete Applied Mathematics*, 52(3):233–252, 1994.

[77] J. Lu, Y. Zhang, J. Xu, G. Xiao, and Q. A. Liang. Data visualization of web service with parallel coordinates and nodetrix. In *IEEE International Conference on Services Computing, SCC*, pages 766–773. IEEE Computer Society, 2014.

[78] S. Mac Lane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3(3):460–472, 09 1937.

[79] D. M. McBride and B. A. Dosher. A comparison of conscious and automatic memory processes for picture and word stimuli: A process dissociation analysis. *Consciousness and cognition*, 11(3):423–460, 2002.

[80] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[81] M. A. Miller, W. Pfeiffer, and T. Schwartz. Creating the CIPRES Science Gateway for inference of large phylogenetic trees. In *Gateway Computing Environments Workshop (GCE)*, pages 1–8, Nov 2010.

[82] B. M. E. Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, June 1988.

[83] J. Müller and K. Müller. TREEGRAPH: automated drawing of complex tree figures using an extensible tree description format. *Molecular Ecology Notes*, 4(4):786–788, 2004.

[84] R. D. Page. Visualizing phylogenetic trees using treeview. *Current Protocols in Bioinformatics*, 00(1):6.2.1–6.2.15, 2003.

[85] W. H. Piel, L. Chan, M. J. Dominus, J. Ruan, R. A. Vos, and V. Tannen. Treebase v. 2: a database of phylogenetic knowledge. *e-BioSphere*, 2009.

[86] S. Porschen, B. Randerath, and E. Speckenmeyer. Linear time algorithms for some not-all-equal satisfiability problems. In *SAT*, volume 2919 of *LNCS*, pages 172–187. Springer, 2003.

[87] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Graph Drawing*, pages 248–261, 1997.

[88] H. C. Purchase. Effective information visualisation: a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2):147–162, 2000.

[89] A. Rusu and A. Fabian. A straight-line order-preserving binary tree drawing algorithm with linear area and arbitrary aspect ratio. *Computational Geometry*, 48(3):268–294, 2015.

[90] T. J. Schaefer. The complexity of satisfiability problems. In *ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 216–226. ACM, 1978.

[91] W. Shih, S. Wu, and Y. Kuo. Unifying maximum cut and minimum cut of a planar graph. *IEEE Transactions on Computers*, 39(5):694–697, 1990.

[92] C.-S. Shin, S. K. Kim, and K.-Y. Chwa. Area-efficient algorithms for straight-line tree drawings. *Computational Geometry*, 15(4):175–202, 2000.

[93] M. Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[94] A. Stamatakis, H. Meier, and T. Ludwig. RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, 12 2004.

[95] S. M. Stanley and S. M. Stanley. Macroevolution: Pattern and Process. 1979.

[96] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[97] J. Sukumaran and M. T. Holder. Dendropy: a Python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571, 2010.

[98] R. Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2013.

[99] W. Tutte. *Connectivity in Graphs*. Mathematical expositions. University of Toronto Press, 1966.

[100] T. Warnow. Tree compatibility and inferring evolutionary history. *Journal of Algorithms*, 16(3):388 – 407, 1994.

[101] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

[102] X. Yang, L. Shi, M. Daianu, H. Tong, Q. Liu, and P. M. Thompson. Blockwise human brain network visual comparison using nodetrix representation. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):181–190, 2017.

[103] W. N. W. Zainon and P. Calder. Visualising phylogenetic trees. In *Proceedings of the 7th Australasian User Interface Conference - Volume 50*, AUIC '06, pages 145–152, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[104] J. Zhang and Q. Li. Shortest paths on evolving graphs. In *5th Int. Conf. on Computational Social Networks (CSoNet)*, volume 9795 of *Lecture Notes in Computer Science*, pages 1–13, Berlin, Heidelberg, 2016. Springer.

[105] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pages 57–64. IEEE, 2005.