UNIVERSITY OF CALIFORNIA,
IRVINE

Topological Algorithms for Geographic and Geometric Graphs

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Siddharth Gupta

Dissertation Committee:
Professor David Eppstein, Chair
Professor Michael T. Goodrich
Professor Sandy Irani

2018

DEDICATION

To my parents Govind Gupta and Vimla Gupta, my sister-in-law Shraddha Gupta and especially to my brother and best friend Yatharth Gupta for always having faith in me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Professor David Eppstein and Professor Michael Goodrich for advising me during my graduate studies. Additionally, I would also like to thank Professor Sandy Irani for serving on my dissertation and advancement committees and Professor Amelia Regan and Professor R. Jayakrishnan for serving on my advancement committee.

I would like to thank all my coauthors, Giordano Da Lozzo, David Eppstein, Michael T. Goodrich and Manuel R. Torres, it was a pleasure working with you all.

I would like to thank my fellow graduate students and postdocs in the Center for Algorithm and Theory of Computation for many enjoyable and enlightening conversations over the past years, especially Giordano Da Lozzo with whom I also had the opportunity to conduct research.

I would like to thank the Mathematics Department at BITS-Pilani, Goa Campus and in particular Professor Tarkeshwar Singh for introducing me to the beautiful world of graph theory.

I would like to thank the Donald Bren School of Information and Computer Science at University of California, Irvine for their funding support that enabled my research. Additionally, much of the research in this dissertation was supported by the National Science Foundation under grants 1228639, 1526631, CCF-1618301, CCF-1616248 and by the U.S. Defense Advanced Research Projects Agency (DARPA) under agreement no. AFRL FA8750-15-2-0092. 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 also like to thank NSF for providing me with travel grant to attend the ACM SIGSPATIAL conference.

I would like to thank Springer and ACM for giving me permission to include their copyrighted material in this dissertation. Reference to original sources are given at the beginning of each chapter containing their copyrighted material.

I would like to thank my family and friends for their love and support. I would especially like to thank Gagan, Rangoli, Shreeaa and Vinayak for all the fun times we had together whenever I visited home.

CURRICULUM VITAE

Siddharth Gupta

EDUCATION

Doctor of Philosophy in Computer Science — **2018**
University of California, Irvine — *Irvine, USA*

Master of Science in Mathematics — **2014**
BITS-Pilani, Goa Campus — *Goa, India*

Bachelor of Engineering in Computer Science — **2014**
BITS-Pilani, Goa Campus — *Goa, India*

RESEARCH EXPERIENCE

Research Visit, INRIA — **Jun - Jul 2017**
Host: Dr Laurent Viennot — *Paris, France*

Research Intern, Northwestern University — **Jun - Dec 2013**
Mentor: Professor Ankit Agrawal — *Chicago, USA*

Research Intern, Indian Institute of Science — **May - Jul 2012**
Mentor: Professor L. Sunil Chandran — *Bangalore, India*

TEACHING EXPERIENCE

Teaching Assistant — **2016 - 2017**
University of California, Irvine — *Irvine, USA*

Teaching Assistant — **2011 - 2013**
BITS-Pilani, Goa Campus — *Goa, India*

PUBLICATIONS

Conference Publications

Subexponential-Time and FPT Algorithms for Embedded Flat Clustered Planarity **2018**
Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, Siddharth Gupta
International Workshop on Graph-Theoretic Concepts in Computer Science

Crossing Patterns in Nonplanar Road Networks **2017**
David Eppstein, Siddharth Gupta
ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems

A Topological Algorithm for Determining How Road Networks Evolve Over Time **2016**
Michael T. Goodrich, Siddharth Gupta, Manuel R. Torres
ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems

Workshop Publications

A New Parallel Algorithm for Two-Pass Connected Component Labeling **2014**
Siddharth Gupta, Diana Palsetia, Md. Mostofa Ali Patwary, Ankit Agrawal, Alok Choudhary
IEEE IPDPS Workshop on Multithreaded Architectures and Applications

Papers in Submission

Exact Distance Oracles Using Hopsets **2018**
Siddharth Gupta, Adrian Kosowski, Laurent Viennot

C-Planarity Testing for Embedded Flat C-Graphs with Bounded Embedded-Width **2018**
Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, Siddharth Gupta

ABSTRACT OF THE DISSERTATION

Topological Algorithms for Geographic and Geometric Graphs

By

Siddharth Gupta

Doctor of Philosophy in Computer Science

University of California, Irvine, 2018

Professor David Eppstein, Chair

We study some geographic and geometric graphs namely *road networks* and *clustered graphs* from topological viewpoint, i.e., we consider them as embedded graphs, graphs in which the ordering of edges (clockwise or anti-clockwise) incident on each vertex is uniquely defined. A road network is a graph with a vertex at each intersection of roads and an edge for each segment of road between the intersections. A clustered graph is a graph whose vertices belong to properly nested clusters. We present algorithms and models for some problems related to road networks and clustered graphs.

The first problem we consider is how a road network has evolved over time, given two snapshots from different dates. These graph can also have geometric/geographic information, such as the GPS coordinates of some vertices or labels identifying road names. As there can be vertices and edges without such geometric/geographic information, we take a strictly topological approach for the problem. We propose an algorithm, which runs in polynomial time for non degenerate-road networks, and outputs portions of the network that remained intact and also points out added or removed portions. We also analyze our algorithm experimentally taking US road network data from the TIGER/Line archive of the U.S. Census Bureau as input data set and show that our algorithm produces good results in practice.

In the second problem, we study the non-planar properties of road networks. We normally

consider road networks as planar graph but they are actually non-planar due to non-intersecting crossings caused by overpass, underpass, or tunnel. We provide a mathematical model of nearly-planar graphs and show that the non-planar graphs that fit this model do have polynomial expansion, i.e., they and all their subgraphs have small separators. We also analyze the Urban Road Network Data set and show that the model is indeed a good fit for non-planar road networks.

Finally, we investigate the C-Planarity problem which asks for drawing of a clustered graph in which each cluster is represented by a simple closed region with no edge-edge crossings, no region-region crossings, and no unnecessary edge-region crossings. We study C-Planarity for *embedded flat clustered graphs*, embedded graphs whose clusters partition the vertex set. We provide a subexponential-time algorithm to test C-Planarity for these graphs when the face size is bounded. Further, we also study a variation of tree decomposition in which, for each face, including the outer face, there is a bag that contains every vertex of the face. We show that C-Planarity is fixed-parameter tractable with the embedded-width of the underlying graph and the number of disconnected clusters as parameters.

Chapter 1

Introduction

In this dissertation, we investigate some problems related to *road networks* and *clustered graphs* considering them as embedded graphs. Formally, a graph consists of a set of vertices and a set of edges connecting the vertices. An embedded graph uniquely defines the cyclic ordering of edges incident to every vertex. (see Fig. 1.1)

Road networks can be modeled as graphs by placing a graph vertex at each intersection or terminus of roads, and connecting every two vertex by an edge if there exists a road segment between them. In addition, these road networks can also have geometric/geographic information, such as the GPS coordinates of some vertices or labels identifying road names. When we consider road networks as embedded graphs, we don't consider the geometric or geographic information associated with them.

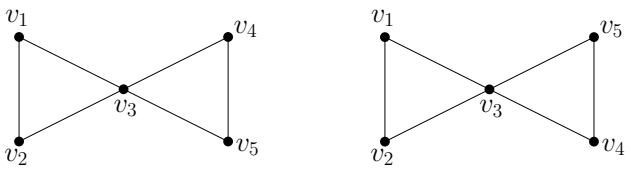


Figure 1.1: Two different embedded graphs on same set of vertices and edges.

Formally, a clustered graph (or c-graph) is a pair $\mathcal{C}(G, \mathcal{T})$ with an underlying graph G along with a hierarchical clustering $\mathcal{T}$ on it's vertices. When we consider c-graphs as embedded graphs, the underlying graph G is an embedded graph.

1.1 Results

In this dissertation, we consider two problems related to road networks and one related to c-graphs.

Road Network Evolution: In chapter 2, we study the road network evolution problem which is related to the map alignment problem (see [72, 73]). In the problem of map alignment, we are given two road networks, including both topological information and geometric/geographic information, and we are interested in computing a most likely matching between the two networks. In our problem, we are given only topological information. As defining the best matching in terms of a maximum common subgraph is unlikely to lead to a polynomial-time algorithm as then the problem is nothing but the subgraph isomorphism problem, we introduce the notion of *conformal matching* as a parameter to define the best matching between two road networks in our case.

We propose an algorithm that runs in polynomial time for finding conformal matchings between non-degenerate embedded graphs, which also includes road networks. Our algorithm uses flood based breadth-first search technique and has mainly two phases. In the first phase, we find a good pair of starting nodes and in the second phase, we find the conformal matching. As the pair of starting nodes may not be unique so we choose the pair which minimizes the probability of wrong matches. As our algorithm is probabilistic in nature, we analyze our algorithm experimentally and also verify the results using the actual geometric information of the vertices and show that our algorithm produces good matching between the vertices in

practice.

Crossing Patterns in Nonplanar Road Networks: Most of the past work done on road networks either consider them as planar graphs or planarize them by introducing artificial intersection points to the roads which cross each other without intersection. Considering them as planar graphs allows us to develop more efficient algorithms due to planar graphs properties like planar graph duality and planar graph separator theorems. But as observed by Eppstein et al. [35], these networks include many crossings and are not actually planar.

In chapter 3, we study crossing patterns in nonplanar road networks. We introduce a new graph called the *crossing graph* of a embedded graph G as a graph which has a vertex for each edge in G and two vertices have an edge between them if the corresponding edges intersect in G. We provide a mathematical model based on crossing graphs and prove that the graphs contained in this model have polynomial expansion which can be used to design efficient separator based algorithms. We also analyze the Urban Road Network Data set and show that this model fits well to non-planar road networks.

C-Planarity Testing: In chapter 4, we study the problem of C-Planarity which deals with the existence of clustered-planar (c-planar) drawing of a c-graph. A c-planar drawing consists of drawing of the underlying graph and drawing of each cluster as a closed curve such that no two edges may cross each other (i.e. the underlying graph should be planar), no two curves may cross each other, an edge may only intersect a cluster boundary only when it connects a vertex inside the cluster to a vertex outside the cluster and it can intersect a cluster boundary only once. (see Fig. 4.1)

The problem was introduced by Feng et al. [42] in 1995. It is still unknown whether it is possible to construct c-planar drawings in polynomial time. Although the complexity of the general problem is unknown, there are many special cases for which polynomial time and fixed-parameter tractable (FPT) algorithms are known (see e.g. [14, 49, 61]). In chapter 4, we

study the problem for the special case of *embedded flat c-graph* where the embedding of the underlying graph is fixed and the clustering is flat i.e. the clusters partition the vertices of the underlying graph.

We provide two algorithms for these kinds of input. The first algorithm is a separator based divide-and-conquer algorithm which runs in subexponential time for embedded flat c-graph with bounded face size. The second algorithm is an FPT algorithm based on Courcelle's theorem, parameterized by embedded-width and number of disconnected clusters of the input graph. Embedded-width is a variant of treewidth (introduced by Borradaile et al. [16]) in which for each inner face of the underlying graph, there exists atleast a bag in the decomposition which contains all the vertices of that face. We use a slightly modified definition in which this property is also true for outer face.

Chapter 2

A Topological Algorithm for Road Networks Evolution[1]

2.1 Introduction

Road network algorithms are an important topic of study in Geographic Information Systems (GIS), in that road networks facilitate transportation and are the products of social, geographic, economic, and political forces. In addition, road networks are interesting data types, in that they combine both geometric information and graph-theoretic information. (E.g., see [35].) Formally, we view a road networks as a graph, where we create a vertex for every road intersection or major jog, and we create an edge for every pair of such vertices that have a road segment that joins them. In addition, some road networks are annotated with geometric/geographic information, such as the GPS coordinates of some vertices or labels identifying road names. Nevertheless, because road networks may contain many vertices and edges without such geometric/geographic information, we are interested in this chapter in

[1]This chapter is included with permission from ACM [48].

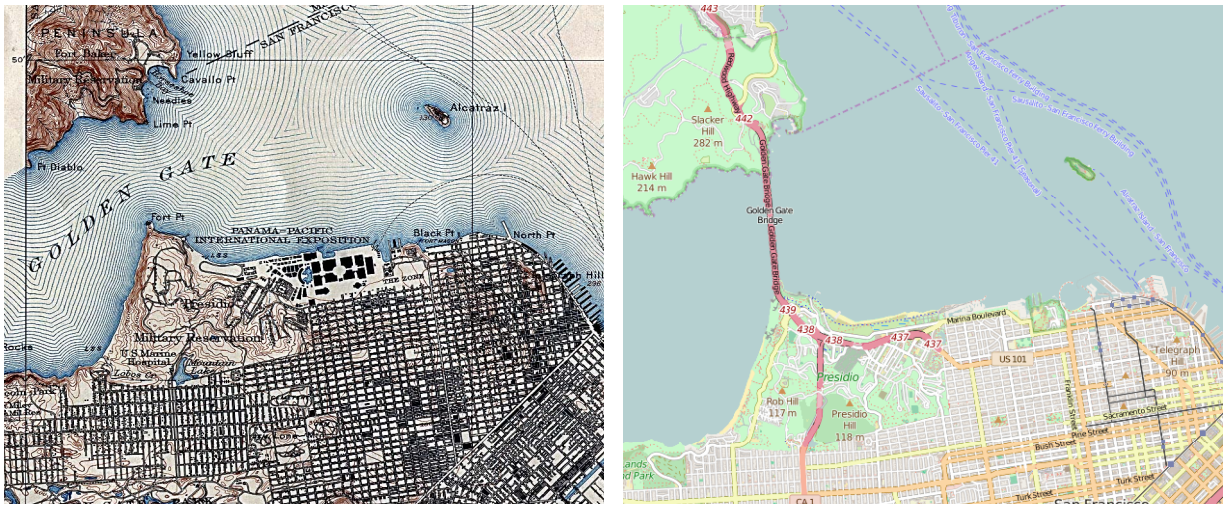


Figure 2.1: A map of San Francisco from 1915 and one from 2016 (taken from OpenStreetMap). The left image is in the public domain; the right image is licensed under the Open Database License, CC BY-SA. Note that most of the roads in both maps are not labeled.

studying road networks from strictly a topological viewpoint, that is, as embedded graphs. Specifically, we are interested in the problem of determining how road networks evolve over time, e.g., highlighting places where new roads and bridges are added and where old roads and bridges are removed. (See Figure 2.1.)

2.1.1 Problem Definition

Viewed topologically in terms of their graph properties, road networks are embedded graphs, that is, the edges incident on each vertex are given in a particular order (i.e., clockwise or counterclockwise), which defines a topological structure for the graph known as a *rotation system* (e.g., see [76]). Road networks are not typically planar graphs (e.g., see [35]), however, since there are edge crossings, for example, at overpasses. Thus, we cannot in general apply algorithms for planar graphs to road networks. Nevertheless, the vertices in road networks have bounded degrees (since the number of roads that meet at a single junction cannot be arbitrarily large); hence, a road network with n vertices has $O(n)$ edges.

Given two undirected graphs, G_1 and G_2, an *isomorphism* of G_1 and G_2 is a bijection, f, from the vertices of G_1 to the vertices of G_2 such that (u, v) is an edge in G_1 if and only if $(f(u), f(w))$ is an edge in G_2 (e.g., see [66]). In the *subgraph isomorphism* problem, we are given two graphs, G_1 and G_2, and asked to determine whether there is a subgraph of G_1 isomorphic to G_2. This problem is NP-complete, even if G_1 is an embedded planar graph, by a reduction from the planar Hamiltonian circuit problem [46]. Thus, defining the best matching between two road networks simply in terms of a maximum common subgraph is unlikely to lead to a polynomial-time algorithm. So let us restrict the types of matchings we consider.

Suppose we are given a subgraph, G_1', of a graph, G_1, and a subgraph, G_2' of a graph, G_2, such that G_1 and G_2 are embedded graphs, i.e., having specified rotation systems. Suppose further that f is an isomorphism from G_1' to G_2'. We say that f is *conformal* if it satisfies the following conditions:

1. For every vertex v in G_1', v has the same degree in G_1 as $f(v)$ has in G_2. That is, we only match vertices having the same degree.

2. For every pair of incident edges, (v, u) and (v, w), in G_1', (v, u) precedes (v, w) in the clockwise order of edges around v in G_1 if and only if $(f(v), f(u))$ precedes $(f(v), f(w))$ in the clockwise order of edges around $f(v)$ in G_2. That is, we match vertices consistently with the edge orderings around each vertex.

Since road evolution tends to involve adding or removing whole roads or neighborhoods, we restrict our notion of road network evolution in this chapter to be defined in terms conformal matchings. There is still one more restriction that we need to add, however, which deals with degeneracies that are unlikely to occur in real-world road networks.

Suppose we are given two road networks, G_1 and G_2, and a maximum-cardinality subgraph,

G'_1, having a conformal matching, f, to a subgraph of G_2 (which is how we determine the parts of G_1 that are the same in G_2). We say that G_1 and G_2 are *degenerate* if, for any vertex v in G'_1 and edge (v, w) in G_1, we can change the assignment, $f(w)$, for w and still have f be a conformal matching (even allowing for $f(w)$ to be undefined). Since our intended applications involve the second road network being a newer copy of the first, a maximum-cardinality subgraph with a conformal matching identifies the portions of the road network that have not changed over time; hence, the portions outside of this maximum-cardinality subgraph identify the portions that have changed. Thus, we argue that such applications involve non-degenerate graphs, since it is unlikely, for example, for us to encounter an 8×16 grid that evolves into a grid-like annulus of 64 nodes with radius 8, which would be degenerate. Given that such configurations are likely to be rare in the real world, we are interested in this chapter only in finding maximum conformal matchings in non-degenerate pairs of road networks, which is a problem we refer to as the *map evolution* problem.

Incidentally, the map evolution problem should not be confused with the *map matching* problem (e.g., see [63, 64]), which is the unrelated problem of matching a trajectory of (possibly noisy) GPS coordinates, as might be produced by a moving vehicle, to the geometry of the road network in which the trajectory is traveling.

2.1.2 Prior Related Work

As noted above, the map evolution problem is related to the graph isomorphism problem, which has a rich history (e.g., see [51, 66]), due to the fact that it is not known to be NP-complete, and the best known worst-case algorithm runs in quasipolynomial time [10], but the problem tends to be feasible in practice (e.g., see [66]). For the generalized approximate graph isomorphism problem, which is NP-hard, Arvind *et al.* [8] give a quasipolynomial approximation algorithm. Such algorithms are necessarily not taking advantage of any

efficiencies, however, that could come from topological considerations like our restrictions to embedded graphs and conformal matchings.

The map evolution problem is also related to the *map alignment* problem, which is also known as *GIS conflation* (e.g., see [72, 73]). In this problem, one is given two road networks, including both topological information (such as vertex-edge-face relationships) and geometric information (such as vertex coordinates and edge directions and lengths), and one is interested in computing a "most likely" matching between the two networks. Rosen and Saalfeld [72, 73] develop an iterative process involving a human operator based on matchings that use topology/geometry classifications of the vertices, edges, and faces of the maps. Xiong [81] extends these topological/geometric approaches using more sophisticated classifications. Savary and Zeitouni [74] and Zhang [82] extends these approaches further by including additional properties, such as geographic data, including road names and shapes. Their use of geometry, however, implies that all of these conflation methods are not strictly topological algorithms and their performance degrades when roads or vertices lack geometric or geographic information.

Detecting changes in road networks and geographic regions has also been studied from the perspective of image processing, e.g., using satellite images (e.g., see [79]). For example, Zhang and Couloigner [83] use image analysis to extract polylines defining roads and match them between two images of the same geographic region taken at different times. Such image-analysis approaches are inherently geometric, however; hence these are also not strictly topological algorithms and do not apply when image data is not available.

Our topological approach is more closely aligned with the work of Eppstein *et al.* [36], which uses a topological approach for approximately matching for quadrilateral meshes used in computer-generated animations. Our approach differs from their methods, however, in that we do not consider faces in our matching algorithm (since road network faces can be large and complex), whereas their method crucially depends on matching faces (which in their

application are always quadrilaterals or triangles).

2.1.3 Our Results

In this chapter, we study the map evolution problem, for matching two road network graph of same area but from different time, by using only topological properties. The primary motivation for this approach is to show that the map evolution problem problem can be solved effectively using only topological information. Thus, this gives GIS practitioners a tool that can be applied for solving the map evolution problem even for problem instances where geometric and geographic information is missing, such as in older hand-drawn maps, pairs of maps where only one of them is derived from an image, pairs of maps annotated in different languages, or maps missing geographic and geometric annotations due to scaling resolution.

We develop an algorithm for the map evolution problem that runs in polynomial time for finding conformal matchings between non-degenerate embedded graphs, such as real-world road networks. Our algorithm uses a breadth-first flooding technique that begins each flooding phase by finding potentially-matching "seed" vertices using a labeling technique similar to that used in the the Weisfeiler-Leman (WL) graph isomorphism algorithm (e.g., see [51]). So as to limit the amount of flooding done in subgraphs that ultimately are determined not to match, our algorithm is probabilistic in nature—when we don't have any pair of unique starting nodes, we take the pair which minimizes an estimate of the probability of a wrong match. We provide verification of our algorithm in experiments and case studies that show empirically that our algorithm produces good matches in practice.

2.2 Our Algorithm

In this section, we describe our topological algorithm for finding a best conformal matching between two non-degenerate road networks, G_1 and G_2:

1. Create quasi-unique labels for each vertex, v, in G_1 and G_2 based on the degrees of the nodes at distance at most k from v, for an input parameter, k. (We show in our experimental section that choosing k between 5 and 8 tends to give the best results.)

2. Choose a good pair of starting nodes, $s_1 \in G_1$ and $s_2 \in G_2$, with the same quasi-unique label, L, and, for each such pair having label L, perform the following:

   (a) Perform a breadth-first search (BFS) matching of the corresponding portions in G_1 and G_2 that are respectively reachable from s_1 and s_2 according to a greedy conformal matching that emanates out from these starting nodes.

   (b) Save this conformal matching that starts from s_1 and s_2 if it is the best (highest cardinality) such matching found so far for this quasi-unique label.

3. Commit the conformal matching that began with s_1 and s_2, removing all matched nodes as candidates for starting nodes.

4. Repeat the above process for another good pair of starting nodes, if such a pair of nodes still remains.

We describe these steps in more detail below.

2.2.1 Labeling Vertices

The first step of our algorithm is to give each vertex, v, in G_1 and G_2 a quasi-unique label, based on the degrees of the nodes at distance at most k from v, for a given parameter,

k. This approach is similar to a labeling method used in the (exact) graph isomorphism algorithm by Weisfeiler and Leman (WL) [51]. Specifically, we begin by determining the degree, $\deg(v)$, of each vertex, v. Then we create a list for each vertex, v, which contains its degree, followed by the degrees of nodes at distance 1 from v, nodes at distance 2 from v, and so on, up to a distance k, where k is an input parameter for this step. So as to make sure that these labels are quasi-unique, we add the degrees of these nodes at distance at most k from v according to a canonical ordering, which in our case is a lexicographically minimum breadth-first search (BFS) ordering. This BFS ordering sorts the immediate neighbors of v according to a lexicographically minimum cyclic ordering of v's neighbors based on their degrees, and then it performs a BFS from this queue, adding nodes to the queue based on the cyclic ordering of edges around each vertex so long as they are at distance at most k from v.

We return a dictionary for G_i (for $i = 1, 2$), which we call masterTable(G_i), such that each entry in this dictionary is a list of vertices having the same quasi-unique label. That is, the keys we use to index the (list) entries in masterTable(G_i) are the label[v] lists produced by our quasi-labeling method.

The pseudocode for this step is given in Algorithm 1.

Algorithm 1: Algorithm for labeling each vertex with a quasi-unique label. The method, lexicographicBFS(v, k), returns an ordered list of nodes as would be visited a breadth-first search (BFS) from v, starting with the neighbors of v enqueued according to a lexicographically minimum cyclic ordering of v's neighbors based on their degrees. This BFS explores all nodes at distance at most k from v.

```
function labelNodes(k, G);
for each v ∈ G do
    label[v] = (deg(v))        # label is a list;
    for u ∈ lexicographicBFS(v, k) do
        Append deg(u) to end of label[v];
    end
    Add v to masterTable(G) [label[v]];
end
```

With respect to the efficiency for performing this step, note that the time needed for this step

is dominated by our doing a BFS from each node, v, to explore those other nodes at distance k from v. Since the vertices of a road network have degree bounded by some parameter, d, this step takes worst-case time $O(d^k n)$, for a road network of n nodes. In practice, k is a constant, d is usually 3 or 4, and the graph is rather sparse; hence, this step runs in $O(n)$ time in practice.

2.2.2 Choosing Pairs of Starting Nodes

After we have labeled each vertex of G_1 and G_2 with quasi-unique labels, we need to choose a pair of starting nodes in G_1 and G_2 with the same label to start the matching process. If we are able to find a unique pair of nodes having the same label, then we can take them as starting nodes and start our matching. But it may happen that we don't have any such unique pair of nodes; that is, it might be the case that there are at least 3 nodes from $G_1 \cup G_2$ for each quasi-unique label of vertices in the master table.

For each distinct label, L, let $n_1(L)$ denote the number of vertices in G_1 with label L and let $n_2(L)$ denote the number of vertices in G_2 with label L. As mentioned above, if we have a label, L, such that $n_1(L) = n_2(L) = 1$, then we choose the unique pair of vertices, $s_1 \in G_1$ and $s_2 \in G_2$, with label L as a good pair of starting vertices.

Otherwise, we would like to choose a pair, $s_1 \in G_1$ and $s_2 \in G_2$, that maximizes the probability that there is a large conformal matching of the connected components of G_1 and G_2 respectively containing s_1 and s_2, such that s_1 and s_2 have the same quasi-unique label, L. For any such label, L, the number of such candidate pairs is $n_1(L) \cdot n_2(L)$; hence, to maximize the probability of finding a good pair of starting nodes, we choose a pair, s_1 and s_2, that minimizes the product, $n_1(L) \cdot n_2(L)$, since the probability such a pair actually correspond to corresponding nodes in G_1 and G_2, conditioned on their having the same label, L, is at least $1/(n_1(L) \cdot n_2(L))$.

We then perform a flooding-based search from each such s_1 and s_2 with label L, committing to the pairing that results in the largest matched components in G_1 and G_2. Then, we remove all the matched vertices in G_1 and G_2 from consideration (since they are now matched), and we repeat our search for another good pair of starting seed vertices.

In order to perform such searches and updates quickly, we use an auxiliary priority queue data structure that stores each quasi-unique label, L, according to its priority, $n_1(L) \cdot n_2(L)$. Such products can be found by taking the product of lengths of both lists for each label used as a key in masterTable. As we are performing our greedy matching processes, we also need to update these lists by removing each matched pair of nodes. Of course, this will also change the product for each label, so we have to update labels in our priority queue to now have new priorities. Since these products are always integers in the range $[1, C]$, for some parameter, $C \leq n^2$, let us use a van Emde Boas tree [77, 78] (vebTree) for storing non-zero products, $n_1(L) \cdot n_2(L)$, for each label, L, as well as a hash table, productTable, that gives us the product for any existing label, L. This allows us to perform searches, updates, and finding of labels with minimal product values in $O(\log \log C)$ time.

Every time the algorithm needs a pair of starting nodes, it finds a label, L, with minimum product, $n_1(L) \cdot n_2(L)$, from vebTree. If there are multiple labels having that product, we randomly choose any one of them. After finding the required label, we take a pair of nodes having the same label from the masterTable. After finding the starting pair of nodes, we update these data structures, and the productTable, so that we don't consider this pair of nodes again. Note this approach works even when we have unique pair of nodes having the same label. In that scenario, the product will be 1 and that will be minimum product in vebTree.

With respect to efficiency, we can do all the setup for this step in $O(n \log \log C)$ time. Moreover, we can determine already at this point what is the maximum product, $n_1(L) \cdot n_2(L)$, over all labels, L, for a given value of k. Since k is a constant for real-world road networks and there

is an inverse relationship between k and the size of these products, we can perform a (binary) search to choose k so that the maximum product size is bounded by some constant, C. The running time of this search would be $O(n)$ for constants k and C.

There is a tradeoff, however, between using a large value for k and getting good matches, since two starting nodes are paired only if their quasi-unique labels are the same, that is, if the respective portions of the road network at distance k from these nodes is the same. Since we are considering road networks that are evolving, we therefore don't want to set too high a value for k. Thus, we would like to choose k as small as possible so that the products, $n_1(L) \cdot n_2(L)$, are bounded by a constant, C. (Say, $C \leq 24$.) As we note in the experimental section of this chapter, choosing k between 5 and 8 seems to work well in practice for this purpose.

2.2.3 Flood-based Conformal Matching

After finding a starting pair of nodes, we start our greedy BFS matching process. We begin by marking the starting nodes as matched and we add them to our current tentative matching. As we perform our BFS matching process, we will tentatively be matching up additional pairs of nodes from G_1 and G_2, updating our supporting data structures as we go, e.g., to tentatively remove each such pair from consideration in vebTree. Moreover, if a starting node has more than one lexicographically minimum ordering of the degrees of its neighbors, then we also consider each such ordering of the edges, performing our BFS matching process for each. Tentative matchings are compared on the basis of number of matched nodes and the matching with maximum number of matched nodes is taken as best matching.

This raises an important implementation detail, which we should probably discuss before going on to other details. Our matching algorithm considers different pairs of starting vertices (and even possibly different starting orientations of their incident edges), looking for the pair

that produces the largest portions of matching subgraphs. Thus, we may have tentative matches that need to be undone so that other tentative matches can be considered.

There are at least two possible ways to deal with this branch-and-bound element in our conformal matching algorithm. One way is to checkpoint our supporting data structures, like vebTree, masterTable, and productTable, saving the version that produced the best tentative match so far. This is the method we use, for example, in the version of our algorithm that we implemented for our experiments, since it is easy to implement. Another way is to perform a two-phase commit, where we perform updates to global copies of these data structures, but keep a history of the updates we have performed during a tentative matching, so that we can then roll back these updates if we do not commit to that tentative matching (because there is another one that gave a larger number of matched vertices). This is the version of our algorithm that we analyze for our theoretical analysis.

Given that there is some method that allows us to roll back to an earlier state of our supporting data structures, vebTree, masterTable, and productTable, let us discuss in more detail how our conformal BFS proceeds.

Once we map the neighbors around a pair of starting nodes, as discussed above, we flood-search both graphs using a conformal-matching BFS. When we reach any other node except a starting node in the flooding, we know the edge we are coming from and as we are following clocking ordering around any node, there will be exactly one ordering around that node in which we can traverse and map the neighbors with another graph, so as to be forming a conformal matching. Figure 2.2 shows an example.

For matching any two nodes, $v_1 \in G_1$ and $v_2 \in G_2$, that are not starting nodes, they should satisfy following properties:

- Both v_1 and v_2 should be unmatched.

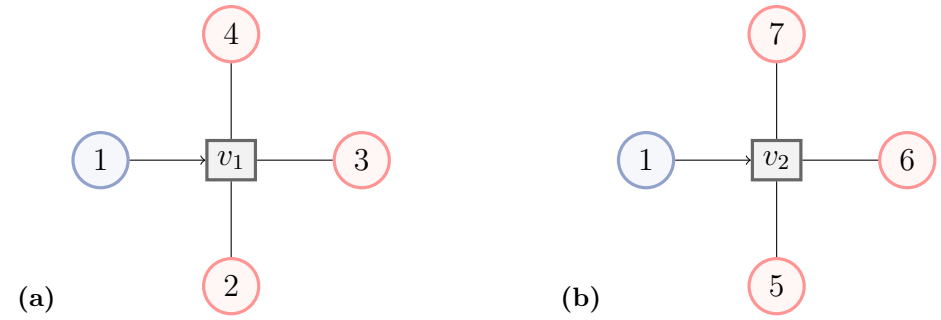


Figure 2.2: Neighbor ordering around a degree-4 node, v_1, and its matching node, v_2. In this example, node 1 in G_1 matches with node 1 in G_2, and we know that the we reached the matched nodes v_1 and v_2 through the respective nodes, 1, so now their clockwise ordering is fixed and the mapping of neighbors will be $(4, 7), (3, 6), (2, 5)$.

- The degree of v_1 should be same as v_2.

If any of these two conditions fail, we don't match v_1 and v_2 and we terminate that branch of the BFS. If both the conditions are satisfied, then we mark v_1 and v_2 as matched, add them to current matching and the queue for the BFS. Then we remove them from masterTable, vebTree and productTable, so that they are not considered again in the matching process. The pseudocode for this step in our algorithm is given as Algorithm 2.

Algorithm 2: Algorithm to process nodes in BFS.

```
function processNodes(u1, u2);
    add (u1, u2) to matching;
    mark u1 and u2 as matched;
    add corresponding neighbors of u1 and u2 to bfsQueue;
    update masterTable, vebTree and productTable to remove u1 and u2;
```

When there is no further branch that can be matched, our BFS search terminates. If this is the best tentative matching for the given quasi-unique label, L, then we tentatively save the matching corresponding to this BFS to the total matching. Then we check if there is still any remaining pair of seed nodes having this same label. If so, then we perform another conformal BFS for this next pair of seed vertices. Once we have completed performing a tentative matching for each pair of seed nodes having the same quasi-unique label, L, we commit to the matching for this label that produced the largest match.

Then we check if vebTree is empty or not. If vebTree is empty, we terminate the algorithm and return the total matching. If not, we repeat our search for a quasi-unique label, L, having the smallest product, $n_1(L) \cdot n_2(L)$, and repeat the above conformal BFS for that label.

The pseudocode for this step in our algorithm is given as Algorithm 3.

Algorithm 3: Our flood-based conformal matching algorithm.

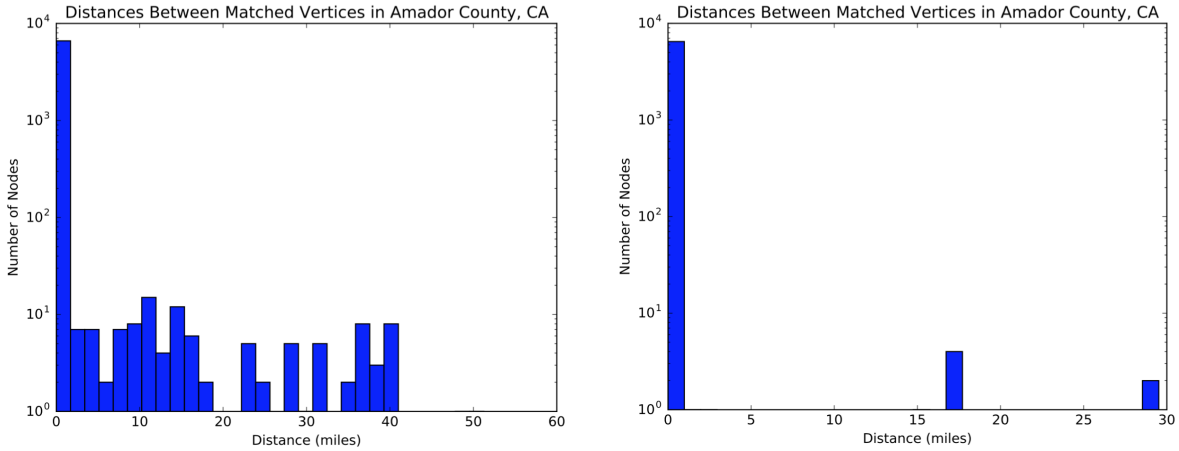
```
function matching(masterTable,G_1, G_2);
create productTable and vebTree;
totalMatching = [ ];
while vebTree is not empty do
    minProd = vebTree.min();
    startingLabel = productTable[minProd];
    startingPairs = (masterTable(G_1) [startingLabel], masterTable(G_2) [startingLabel]);
    find all mappings of neighbors around each pair in startingPairs;
    for each of the mappings in a startingPair do
        matching = [ ];
        (s_1, s_2) = this instance of startingPair[0],startingPair[1];
        bfsQueue = ();
        processNodes(s_1,s_2);
        while bfsQueue is not empty do
            v_1, v_2 = pop(bfsQueue);
            if v_1 and v_2 are both unmatched then
                if deg(v_1) = deg(v_2) then
                    processNodes(v_1,v_2);
                end
            end
        end
        checkpoint this matching if it's best for this startingPair;
    end
    add the best matching found to totalMatching;
end
```

Each time we explore subgraphs of G_1 and G_2 for a particular starting pair, s_1 and s_2, that are in the starting label set of pairs for some quasi-unique label, L, and one of the $\deg(s_1)$ possible orientations of edges, we traverse subgraphs of some size at most, $n(L) \leq n$, where $n(L)$ is the size of the largest match for the label L. Thus, the running time of this part of our algorithm is at most $O(n(L) \log \log C)$, where C is the maximum value of a product,

$n_1(L') \cdot n_2(L')$, for some label, L'. If d is the maximum degree in a road network (e.g., $d \leq 8$), then the total worst-case running time of our BFS matching algorithm is therefore

$$O\left(dC \sum_L n(L) \log\log C\right) = O(dCn \log\log C),$$

since $\sum_L n(L) \leq n$, because the maximum amount of nodes we can ultimately match in a pair of non-degenerate road networks is n. Combining this with the theoretical analysis of the other steps in our matching algorithm implies that the total running time of our entire algorithm is $O(d^k n + dCn \log\log C)$, where d is the maximum degree of a road network, k is the distance we choose for producing quasi-unique labels, and C is the maximum value of a product, $n_1(L) \cdot n_2(L)$, for any label, L. Thus, in the practical case when d, k, and C are constants, our matching algorithm runs in $O(n)$ time.

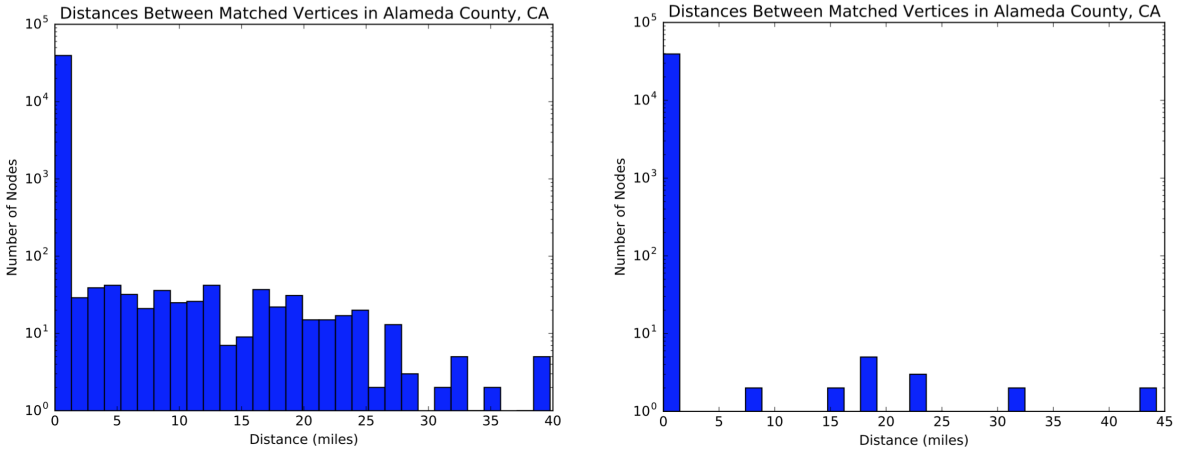


(a) $k = 1$ **(b)** $k = 5$

Figure 2.3: Histogram plots for Amador County, CA from 2000 to 2006.

2.3 Experiments

In this section, we provide an empirical evaluation of our topological flood-based matching. All of our experiments were ran on data from the U.S. TIGER/Line road network database [18].



(a) $k = 1$ **(b)** $k = 7$

Figure 2.4: Histogram plots for Alameda County, CA from 2000 to 2006.

2.3.1 Preprocessing the Data

The TIGER/Line database provides the road networks in two different file formats: shapefile and TIGER/Line ASCII format. The data the shapefile format provides allows a graph to be created that not only has a node for every intersection of two roads, but also nodes to indicate the curvature of a road. That is, the format allows for curved roads to be represented as a sequence of many two-degree vertices. Therefore, for the preprocessing of files in the shapefile format, we simply take the first and the last vertex for each road to avoid introducing unnecessary two-degree vertices. With this approach to processing files in the shapefile format and fact that the TIGER/Line ASCII format lends itself to easy conversion to the definition of a road network given in the introduction, our algorithm performs well on both file formats.

2.3.2 Tuning the Seed-labeling Parameter

Let us consider the choice of the value for k, the parameter that is input to Algorithm 1 that defines the distance to which to perform a lexicographic BFS so as to improve the uniqueness of vertex labels. To characterize this uniqueness factor, let us define the *approximation ratio*

of a labeling as a/b, where a is the number of pairs of nodes with the same label and b is the minimum of the number of nodes in the two graphs. Note that, this is different from the *approximation ration of an algorithm* which is the ratio between cost of the output obtained by the algorithm and the optimal solution cost. Intuitively, if k is small, there will likely be many pairs of nodes (u, v) with u in G_1 and v in G_2 that both have label L where $n_1(L) \cdot n_2(L)$ is large. For example, labels like "44444", which indicates a four-way intersection that leads to four other four-way intersections, are likely to be common, and many other examples like this are likely from real-world. As many of these products are expected to be large, we would expect the approximation ratio to be larger for smaller value for k, because we could be possibly finding many pairs of vertices with the same label that should not actually be matched. For instance, we might find two vertices labeled "44444" even though they are not similar beyond their immediate neighbors. We expect to run into this situation only when the product is large since our algorithm matches the pair of vertices for a given label that maximizes the number of nodes matched.

As we increase the value of k, we would expect that the approximation ratio to decrease. That is, if k is large, we expect there to be more labels L' such that $n_1(L') \cdot n_2(L')$ is small or even 1, as the labels should become more distinct as k increases. Because the labels are expected to be more distinct in this case, it should be less likely to find pairs of vertices with those labels, causing the approximation ratio to decrease.

The histograms in Figures 2.3 and 2.4 exemplify the preceding interpretation of the parameter k. The x-axis indicates the physical distance between every node and its pair partner(s) with the same quasi-unique label, L, using the longitude and latitude values given from the database. The distance is determined using the haversine formula, which yields the shortest distance between two points on a sphere [75]. (Although our algorithm doesn't use geometric information to determine matching pairs, we used geometric information in this experiment to empirically validate our approach.) Ideally, all pairs should be at distance 0 from each

other.

As we expected, larger k values minimize the physical distances between pairs of nodes with the same label, which gives us a more accurate matching; hence, it reduces the number of false pairs that our algorithm needs to consider. A histogram that is highly skewed is desirable, as that implies that the number of incorrect nodes being falsely matched is small. Note that the Amador County data from 2000 and 2006 in Figure 2.3 included 6,970 and 6,784 nodes, respectively, and the Alameda County data from 2000 and 2006 in Figure 2.4 included 52,566 and 51,054 nodes, respectively.

Figure 2.5 shows the change of the approximation ratio with respect to the change in k for Amador County. The decrease in the approximation ratio with the increase in k again matched our intuition. The plot with the same x-axis and y-axis values for Alameda County started at a similar approximation ratio and decreased at a similar rate, so it was omitted.

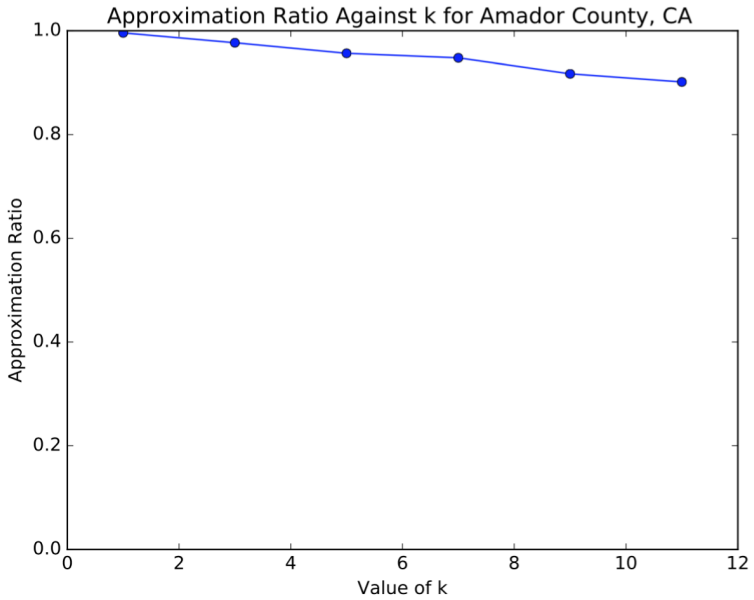


Figure 2.5: Change in approximation ratio for Amador County, CA from 2000 to 2006

We also plot the change in the maximum product with respect to k in Figure 2.6. As described in Section 2, the maximum product is the value $\max\{n_1(L) \cdot n_2(L) : L \text{ is a label generated by Algorithm 1}\}$. As expected, the maximum product decreases as

k increases. Note that only for San Francisco County does the maximum product reach 1. This is due to the fact that for the other counties, there are labels that do not change as k increases as the nodes the labels correspond to are in small connected components e.g. "121" is the cause of this in San Mateo County.

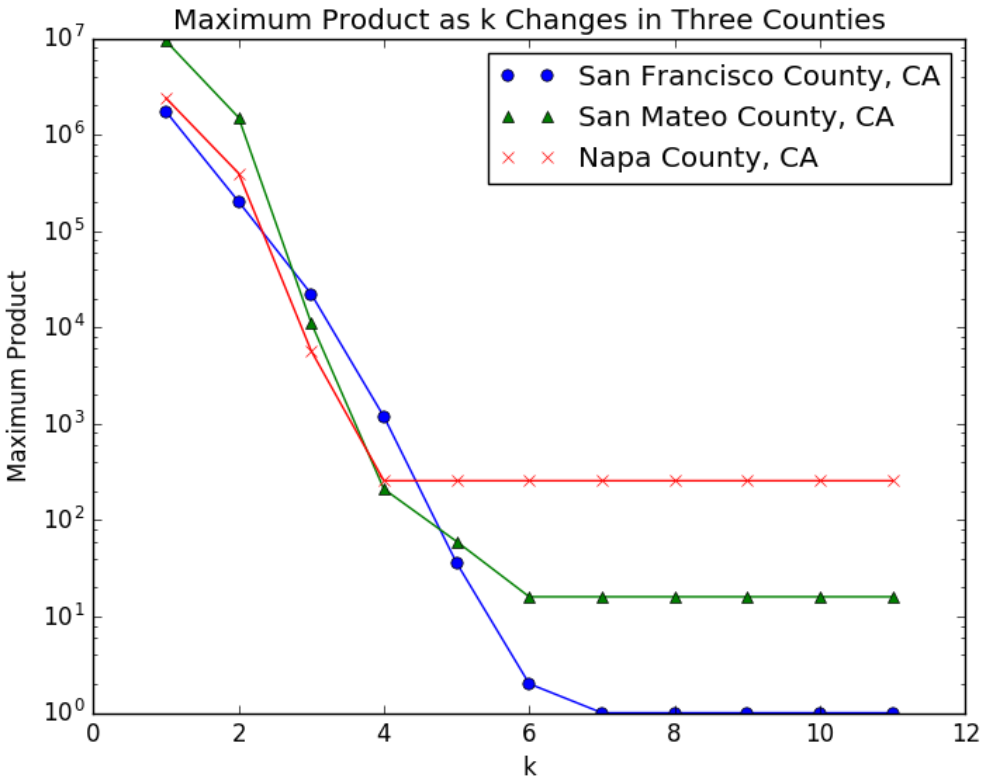


Figure 2.6: Change in maximum product for Napa, San Francisco, and San Mateo Counties with road networks from 2000 and 2006

2.3.3 Example Output of Our Algorithm

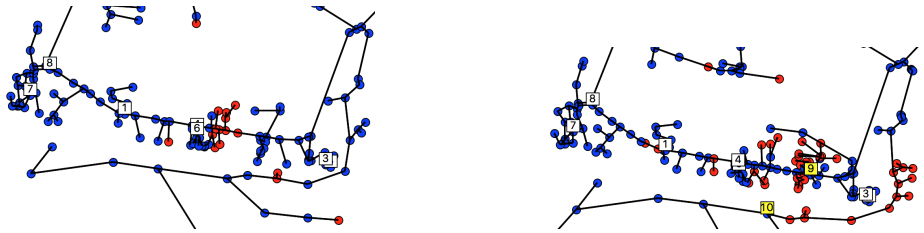
In this subsection, we provide a visualization of the matching our algorithm created for Del Norte County, CA. We performed the matching with $k = 3$ and then took four snapshots of the matching to enlarge the details. For Figures 2.7, 2.8, 2.9, and 2.10, a node is colored blue if it was matched and red otherwise. Furthermore, a node with a white box above it from

the first image containing number i matches the node with a white box above it containing the number i from the second image.

First, consider Figure 2.7. Solely based off of geographic location, it is clear that the nodes are being matched to the correct area. After further inspection, it can be seen that the graph has remained nearly the same around the white boxes containing "1", "3", "7", and "8". Near each of these white boxes, our matching algorithm has matched the correct nodes, indicated by all of the blue nodes surrounding said boxes. Figure 2.7 also demonstrates the issue of using a small value for k. The yellow boxes in Figure 2.7b indicate nodes that have been matched to other nodes in the graph from Figure 2.7a that are not included in the image. This incorrect matching is due to the fact that when k is small, as mentioned earlier, it is likely that many nodes will end up with the same label, yielding a higher likelihood of incorrectly matching two nodes that should not be matched.

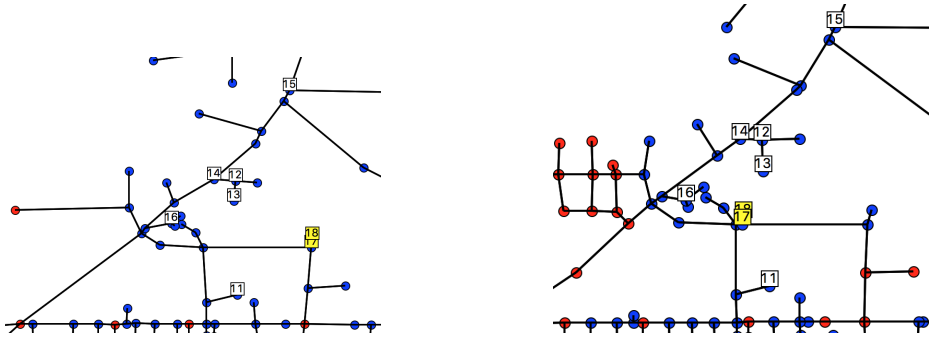
Second, consider Figure 2.8. The white boxes in these figures are here to indicate that the matching algorithm is performing properly in many parts of the graph. As we are just using topological features, we also get some unexpected matching as shown in Figure 2.8. The two yellow boxes in Figure 2.8a are matched to the two yellows boxes in Figure 2.8b. A new vertex was added in the 2006 graph that caused the matching of the vertices under the yellow boxes to occur in the wrong place. Because we are only using topological features, our matching algorithm cannot distinguish between the new vertex and the old one that it should be matching to.

Last, consider Figures 2.9 and 2.10. Many more white boxes were included to show the success of our matching algorithm in these portions of the graph.



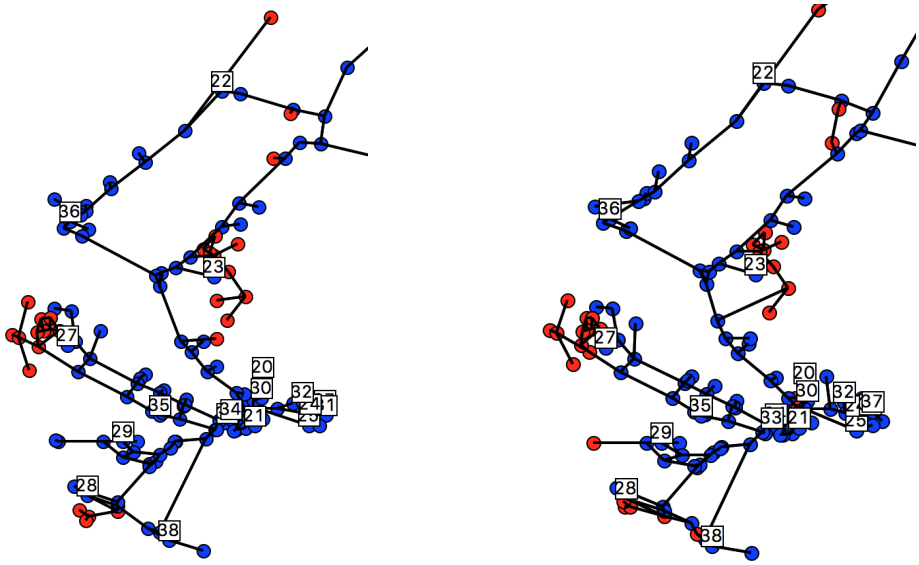
(a) 2000 **(b)** 2006

Figure 2.7: First example of a portion of a matching for Del Norte County, CA where $k = 3$.



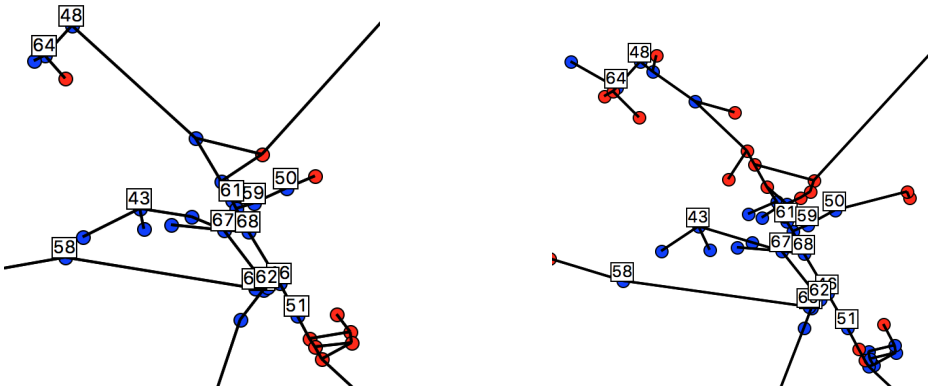
(a) 2000 **(b)** 2006

Figure 2.8: Second example of a portion of a matching for Del Norte County, CA where $k = 3$



(a) 2000 **(b)** 2006

Figure 2.9: Third example of a portion of a matching for Del Norte County, CA where $k = 3$.



(a) 2000 **(b)** 2006

Figure 2.10: Fourth example of a portion of a matching for Del Norte County, CA where $k = 3$.

2.3.4 Detailed Analysis

We ran our algorithm on 40 different counties in California ranging from small counties to big counties. The results for our experiments are shown in Table 2.1. Each row gives analysis about one particular county where G_1 is obtained from TIGER/Line ASCII format from the year 2000 and G_2 is obtained from TIGER/Line ASCII format from the year 2006. The column titled "seed time" indicates the time taken to find the seed vertices for the given value of k and the column titled "match time" indicates the amount of time taken for the topological flood-based matching algorithm. We ran the experiments on a machine with 3.1 GHz Intel Core i7 CPU and 16 GB of RAM and report the timings in seconds. The last column titled "thresh. ratio" is the ratio of number of pairs of matched vertices within 5 miles of each other to the total number of pairs of matched vertices which gives up the quality of matching. We can see from the table that thresh. ratio is always greater than 0.9 which tells us that our algorithm performs well on all kinds of inputs.

Figure 2.11 plots the experiments in Table 2.1 with the total running time (seed time plus match time) as the y-axis and the size of the smaller graph as the x-axis. The red line represents the function $0.003n \log \log n$. Therefore, it seems that the variable C defined in Section 2 is much less than n, which is good for the running time of our algorithm.

county	k	nodes of G_1	nodes of G_2	edges of G_1	edges of G_2	seed time (seconds)	match time (seconds)	approx. ratio	thres. ratio
Alameda	9	40752	40242	67226	66644	778.544	7.686	0.9777	0.9997
Alpine	5	1448	1427	1838	1811	0.536	0.311	0.9439	0.9985
Amador	5	6970	6784	9198	8991	3.017	0.764	0.9553	0.9998
Butte	8	19856	21304	27896	29955	66.540	34.509	0.6878	0.9963
Calaveras	8	13770	13043	18141	17690	23.050	1.185	0.0400	0.9196
Colusa	5	5039	5700	7285	8589	4.231	8.106	0.8106	0.9867
Contra Costa	9	37148	36564	55555	54750	323.512	70.794	0.9482	0.9995
Del Norte	5	5383	7034	7386	9785	3.861	39.562	0.4811	0.9258
El Dorado	9	24248	24103	33331	33271	108.336	10.559	0.9766	0.9991
Fresno	9	51006	50614	83081	82640	744.009	346.011	0.9796	0.9992
Imperial	8	18104	18105	28716	28639	92.026	1.481	0.9592	1.0
Kings	8	11842	15328	18775	25521	87.083	7.071	0.3541	0.9978
Lake	8	12437	18500	17486	26176	47.485	391.934	0.1331	0.9553
Lassen	8	16216	19519	24024	28044	57.645	241.791	0.3804	0.9837
Madera	8	16936	16633	25164	24842	77.837	4.864	0.9633	0.9998
Marin	8	13733	13455	19722	19372	44.946	1.231	0.9446	1.0
Mariposa	5	9241	10538	12033	13830	4.558	119.975	0.5746	0.9546
Mendocino	9	22326	26153	30231	36142	107.554	403.188	0.3761	0.9901
Merced	8	16576	19619	25266	29568	78.726	9.331	0.6058	0.9940
Modoc	8	13304	17408	19674	24889	44.782	3.998	0.3229	0.9837
Mono	5	9159	11345	13203	16178	6.440	30.914	0.6179	0.9595
Monterey	9	31887	33831	48204	51313	324.838	350.278	0.7654	0.9978
Napa	5	6932	6827	10054	9867	4.813	19.717	0.9491	0.9933
Nevada	8	15903	15268	21729	20935	30.650	6.726	0.9135	0.9994
Placer	9	25365	25437	35603	35913	116.914	41.591	0.9287	0.9996
San Benito	5	7555	10311	10421	14649	5.801	30.310	0.5597	0.9503
San Francisco	5	9803	11570	20313	24218	20.241	0.977	0.7138	1.0
San Mateo	9	21571	21101	35132	34532	288.204	5.469	0.9666	0.9997
Santa Cruz	8	14374	14063	20545	20142	48.941	19.229	0.9694	0.9997
Shasta	9	25436	33824	35129	47588	163.593	431.638	0.1867	0.9732
Sierra	5	4809	6522	6603	8912	3.273	5.000	0.3559	0.8603
Siskiyou	9	28210	38150	39682	52616	140.771	705.416	0.1442	0.9528
Solano	8	15930	31249	24859	46954	106.300	63.848	0.1398	0.9654
Stanislaus	8	18254	19240	29327	31094	110.927	1.703	0.5786	0.9962
Sutter	5	6311	6164	9670	9494	4.293	0.319	0.9704	1.0
Tehama	8	15399	19177	21756	27353	53.793	13.337	0.3802	0.9844
Trinity	8	12042	11944	15501	15434	15.961	77.516	0.9780	0.9945
Tulare	9	27555	27302	42308	42257	269.419	10.146	0.9632	0.9993
Tuolumne	9	14830	17135	19985	23283	47.406	131.849	0.2621	0.9809
Yuba	5	9354	9267	13529	13407	5.791	4.158	0.9840	0.9995

Table 2.1: Results for various counties throughout California

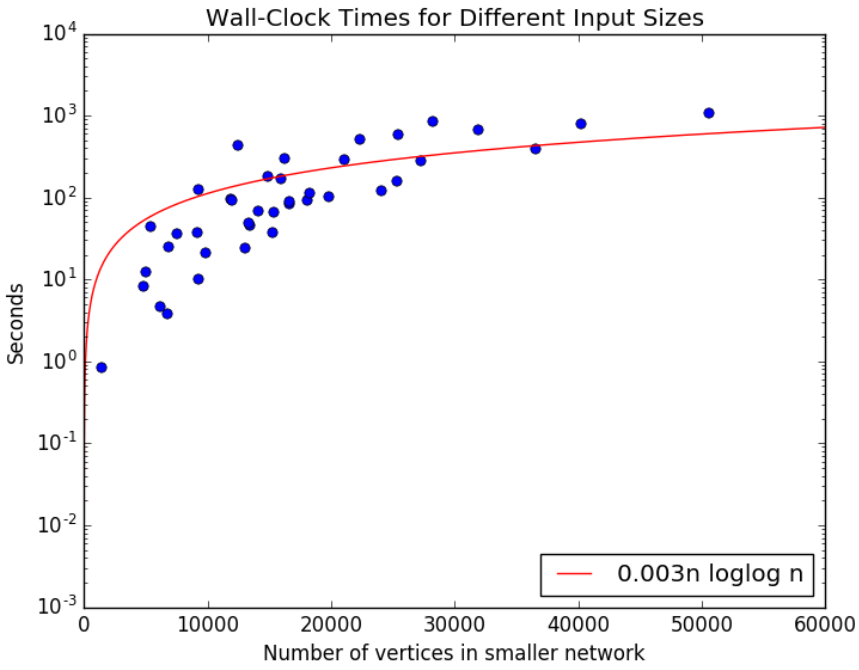


Figure 2.11: Running times for our algorithm on the graphs given in Table 2.1

2.4 Conclusion

We have given a purely topological algorithm for determining the changes that occur between two road networks, and we have provided both theoretical and experimental analysis to show that our algorithm is effective and efficient. We therefore feel that this algorithm provides a good tool for solving the map evolution problem when geometric or geographic features are missing from one or both of the road networks being considered.

Chapter 3

Crossing Patterns in Nonplanar Road Networks[1]

3.1 Introduction

Road networks are often modeled graph-theoretically, by placing a graph vertex at each intersection or terminus of roads, and connecting vertices by edges that represent each segment of road between two vertices. Thus, each vertex is naturally associated with a coordinate on the earth's surface.

Much past work on algorithms for road networks has either assumed that these networks are planar (that is, that no two roads cross without forming an intersection at their crossing point) or has added artificial intersection points to roads that cross without intersection, to force these networks to be planar. Planar graphs have many convenient properties, including planar graph duality and planar graph separator theorems [22, 62] that allow natural and important problems on these networks to be solved more quickly. For instance, for planar graphs,

[1]This chapter is included with permission from ACM [39].

it is known how to compute shortest paths in linear time, based on the planar separator theorem [59], in contrast to the situation for general graphs where shortest paths are slower by a logarithmic factor. Unfortunately, as Eppstein et al. [35] observed, the available data for real-world road networks shows that these networks are not actually planar: they include many crossings. This discovery naturally raises the question of how to model nonplanar road networks, in a way that allows efficient algorithms to be based on their properties.

In this context, one would like a model of road networks that is in some sense near-planar (after all, road networks have few points where roads cross without intersecting, although their number is not zero), that is realistic (accurately modeling real-world road networks), and that is useful (leading to efficient algorithms).

One clear property of road networks is that they are *sparse*: the number of road segments exceeds the number of road intersections by only a small factor, half of the average number of segments that meet at an intersection. Since the vast majority of intersections are the meeting point of three or four road segments, this means that the number of road segments should be between 1.5 and 2 times the number of intersections. Researchers in graph algorithms and graph theory have developed a sophisticated hierarchy of classifications of sparse graph families centered around the intuitive notion of sparseness. Many of these types of sparseness imply general algorithmic meta-theorems about the properties that can be computed efficiently for graphs in the given family. In particular, many of the known algorithms for planar graphs can be extended to the class of graphs of *polynomial expansion*, a property that was originally defined using graph minor theory but that has a more natural equivalent definition (for classes of graphs closed under taking subgraphs) in terms of the existence of sublinear-size separators [34, 68]. Graphs of polynomial expansion support efficient separator-based divide-and-conquer algorithms, as well as more sophisticated pattern matching algorithms based on their graph minor properties. We would like to show that road networks, too, have small separators, and therefore that they can support all of these algorithms.

In this chapter, we provide a mathematical model of non-planar road networks in terms of the sparseness of their *crossing graphs*, graphs representing pairs of road segments that cross in the network. We analyze the Urban Road Network Data set and show that, indeed, it is a good fit for the model. Additionally, we prove that networks within this model have polynomial expansion, from which it follows that the linear-time planar shortest path algorithm of Klein et al [59] can be adapted to work on these networks, despite their non-planarities.

3.2 Past work

3.2.1 Nonplanar road networks

The past work by Eppstein et al. [35, 37, 38] has attempted to model nonplanarities in planar road networks in two different ways. In [35] the authors posited that road networks are subgraphs of the intersection graphs of systems of disks (the disks centered at each intersection of roads with radius equal to half the length of the longest segment of roads meeting at that intersection) and that, with a small number of exceptional high-radius disks, these disks have low *ply* (at most a constant number of non-exceptional disks cover any point of the Earth's surface). They performed an empirical analysis of road network data showing that this model fits actual road networks reasonably well, and they used the assumption to develop efficient road network algorithms. Unfortunately, the class of networks defined by this model does not fit well into the theory of sparse networks, because of its handling of the exceptional disks. Because these disks could be arbitrary, they could in principle heavily overlap each other, producing dense subgraphs that prevent the graphs defined in this way from having polynomial expansion or other good sparseness properties. This misbehavior seems unlikely to happen in actual road networks, but this mismatch between theory (where these graphs can have dense subgraphs) and practice (where dense subgraphs are unlikely) indicates that

it should be possible to replace their model of road networks by another model that more accurately matches the sparsity properties of real-world road networks.

Later work of the same authors [38] attempted to justify the low number of crossings in road networks by showing that randomly chosen lines (modeling, for instance, a highway cutting across an older city grid) typically have a sublinear number of crossings with other roads. Another paper [37] used this observation of few crossings per line as the basis for a very weak assumption about road networks, that the total number of crossings is smaller than the number of intersections by a sufficiently large non-constant factor. This assumption allows some algorithms to be performed efficiently; notably, the crossings themselves can all be found in linear time. However, it is not strong enough to imply the existence of small graph separators for arbitrary subgraphs of road networks, a property that is required by fast separator-based graph algorithms. Additionally, this paper's assumptions about the sparsity of crossings are on dubious ground from an empirical point of view: what reason is there to believe that the ratio of intersections to crossings in large street networks is non-constant rather than a large constant?

3.2.2 Nearly-planar graphs

The graph theory literature includes many natural generalizations of planar graphs that we may choose from. Among these, the k-apex graphs have been defined as the graphs that can be made planar by the removal of k vertices [58]; however, in road networks, the number of vertices that would need to be removed would typically be proportional to the number of crossings, a large enough number that it is not reasonable to treat it as a constant. Similarly, the k-genus graphs are the graphs that can be embedded without crossings into a surface of genus at most k [30]. A road network can be embedded without crossings on a surface with a handle for each overpass or tunnel, and this surface would have genus proportional to the

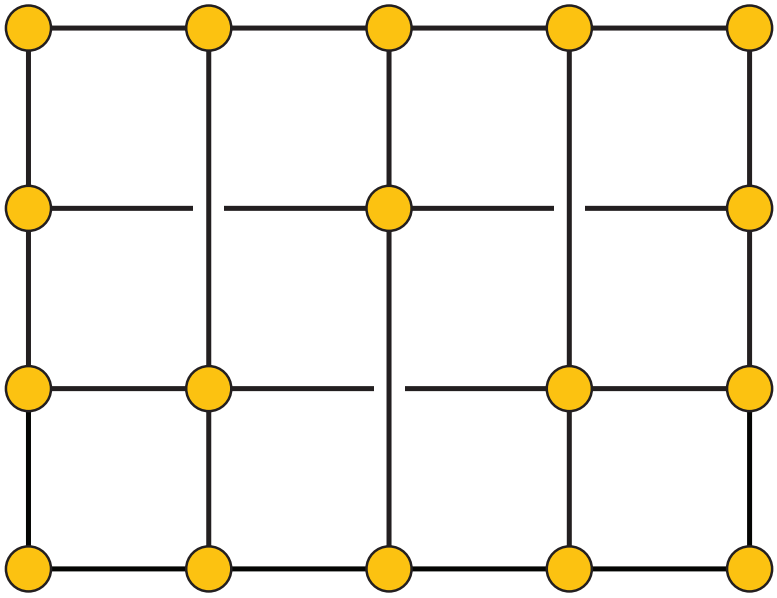


Figure 3.1: A 1-planar graph: each edge is crossed at most once. Although more general than planar graphs, this class of graphs does not adequately model real-world road networks.

number of overpasses and tunnels, but again this number would be too large to treat as a constant. These two graph families form minor-free graph families: families of graphs that, like the planar graphs, have some forbidden graphs that cannot be formed from contractions or deletions of graphs in their family. However, there is no reason to expect any particular forbidden minors in road networks.

Among the many generalizations of planar graphs that have been studied in the graph theory literature, another one seems more promising as a model for road networks: the 1-planar graphs [71] or more generally k-planar graphs [33, 50, 70]. A 1-planar graph is a graph in which every road segment has at most one crossing (Figure 3.1). More generally a k-planar graph is a graph in which every road segment has at most k crossings. Many of the sparsity properties of these graphs follow directly from planarization: if one replaces each crossing with a vertex, one obtains a planar graph in which the number of vertices has been blown up only by a factor of $O(k)$. Based on this principle, it is known that these form sparse families of graphs, and in particular they obey a separator theorem like that for planar

graphs but with a dependence on k as well as on the number of vertices in the size of the separator [33]. Although k-planar graphs are NP-hard to recognize from their graph structure alone [11, 50], that is not problematic for their application to road networks, because in this case an embedding with few crossings would already be known: the actual embedding of the roads on the surface of the earth.

Therefore, it is tempting to model road networks as 1-planar or k-planar graphs. However, the restriction on the number of crossings per edge may be too restrictive to model real-world road network graphs. As we show, the assumption that road networks are 1-planar does not fit the actual data, because real-world road network data includes road segments that have many crossings. In particular, a long segment of highway may have many crossings between interchanges. However, despite the poor fit of this model to the data, we may take inspiration from 1-planar graphs in finding a more general class of graphs with few crossings per edge in some more general sense, that still maintains the other desirable properties of this graph class and that allows the algorithms from the theory of 1-planar graphs to be applied to road networks.

3.3 Overview of new results

3.3.1 The crossing graph

The main new idea of this chapter is to study crossings in road networks by introducing a new auxiliary graph, the *crossing graph* of the road network.

Definition 3.1. We define the *crossing graph* of an embedded graph G to be an undirected graph, different from G itself. Each edge of G becomes a vertex in the crossing graph. When two edges of G cross each other. we connect the two corresponding vertices of the crossing graph (the ones representing the two segments) by an edge representing the crossing.

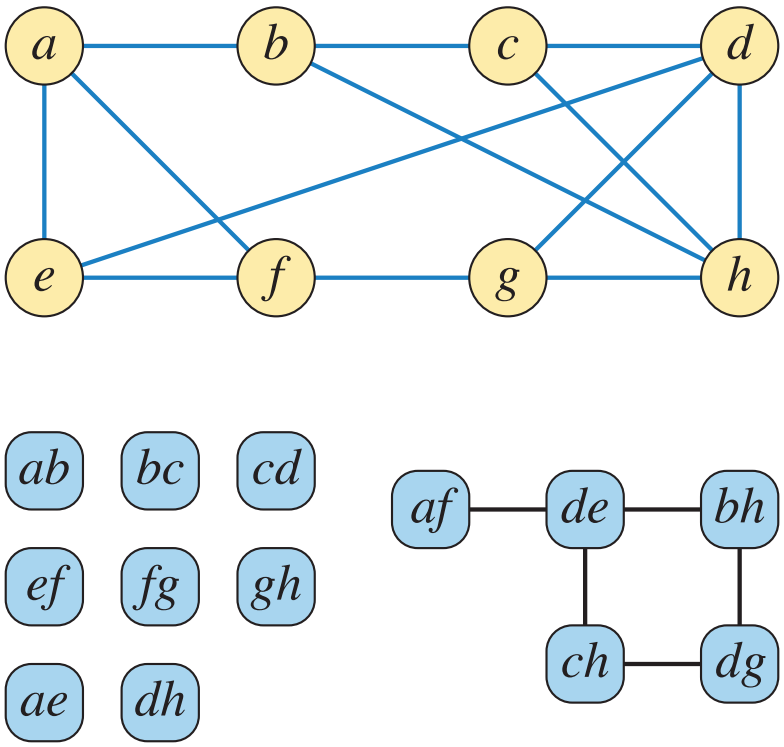


Figure 3.2: A drawing of a graph with crossings (top) and its crossing graph (bottom). The eight isolated vertices of the crossing graph correspond to the eight uncrossed edges in the upper graph.

This concept is illustrated in Figure 3.2. In particular, for a road network, each vertex of the crossing graph represents a segment of road, and each edge of the crossing graph represents two road segments that cross without meeting at an intersection.

This structure allows many natural properties of the underlying road network can be read off directly from the crossing graph. For instance, a segment of road is uncrossed if it corresponds to an isolated vertex (one without any incident edges) in the crossing graph. A road network is crossing-free (planar) if and only if all its segments are uncrossed, if and only if the vertices of the crossing graph are all isolated, if and only if the crossing graph itself is an independent set.

For a more complex example, a road network can be modeled as a 1-planar graph (each edge has at most one crossing) if and only if the crossing graph has maximum degree one; that is, if its crossing graph is a matching. Similarly, the road network is k-planar if its crossing

graph has maximum degree k.

Our hypothesis is that the crossing graph is a sparse graph (although possibly not one with constant maximum degree) and that its structure can be used to investigate the graph-theoretic structure of the road network itself. We study this question both empirically (by computing and examining the structure of crossing graphs for actual large-scale road networks) and theoretically (by proving that certain types of sparsity in the crossing graph imply the existence of small separators and efficient algorithms for the underlying road networks).

3.3.2 Empirical experiments

We investigate empirically the graph structure of the crossing graph, by constructing this graph for the 80 of the most populated urban areas in the world given by the Urban Road Network Dataset [57].

Our investigations show that the degree of the crossing graph is small, but not necessarily small enough to consider to be a constant: we found vertices of degree up to 166 (road segments with up to 166 other segments crossing them, none of the crossings forming an intersection). However, we found that the crossing graphs tend to have much smaller *degeneracy*, a number d such that every graph in a given family of graphs (closed under taking subgraphs) has at least one vertex of degree at most d. For instance, trees are exactly the connected graphs of degeneracy one, and our work found that many of the connected components of the crossing graph are trees. More generally, the maximum degeneracy that we found in the crossing graph of any road network was 6.

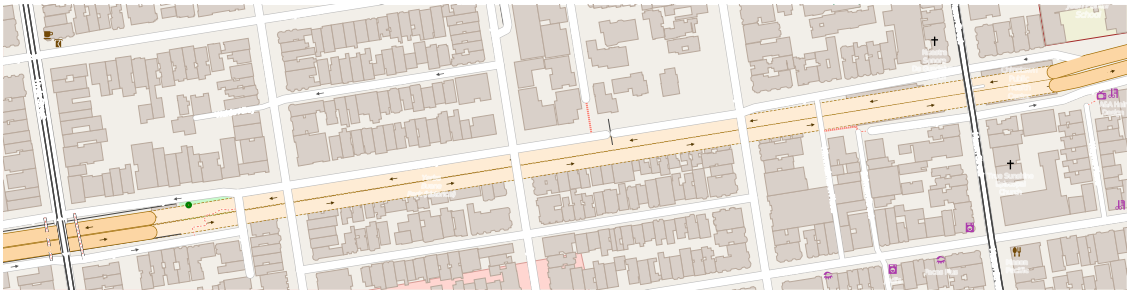


Figure 3.3: The Robert C. Levy tunnel in San Francisco, in which Broadway (orange) passes under seven other streets (L–R: Hyde, Cyrus Pl., Leavenworth, Jones, Taylor, Himmelmann Pl., and Mason) without intersecting them. CC-BY-SA image from OpenStreetMap.

3.3.3 Theory of networks with sparse crossing graphs

Based on our empirical investigations, we undertook a theoretical study of the networks whose crossing graphs have bounded degeneracy. We prove theoretical results showing that these networks are closed under subgraphs and that (like the k-planar graphs) they always have small separators. Therefore, they form a new family of graphs of polynomial expansion.

3.4 Preliminaries

Before detailing our experimental and theoretical results, we provide some necessary definitions.

3.4.1 Sparse graph properties

All graphs in this chapter are finite. Although road networks are typically directed (by the direction of traffic on one-way streets and divided highways), the direction of the edges does not matter for the crossing pattern and separator properties considered here. Therefore, we treat these graphs as undirected.

Definition 3.2. The *degree* of a vertex in a graph is the number of edges touching that vertex. The *minimum degree* $\delta(G)$ and *maximum degree* $\Delta(G)$ of a graph G are the minimum and maximum, respectively, of the degrees of the vertices in G. A family $\mathscr{F}$ of graphs has *bounded degree* if all graphs in $\mathscr{F}$ have maximum degree $O(1)$; that is, if there is an upper bound on the maximum degree that may depend on $\mathscr{F}$ itself but that does not depend on the choice of a graph within $\mathscr{F}$.

Definition 3.3. The *degeneracy* of a graph G is the maximum, over subgraphs of G, of the minimum degree of the subgraph.

A concept equivalent to degeneracy (but differing from it by one) was originally called the coloring number by Erdős and Hajnal [40]. For instance, the graphs of degeneracy one are exactly the forests. A graph has degeneracy at most d if and only if its vertices can be ordered in such a way that every vertex has at most d later neighbors. For, given such an ordering, every subgraph of G has a vertex of degree at most d, namely the first vertex of the subgraph to appear in the ordering. Given a graph of degeneracy d, an ordering with this property can be found by greedily removing the minimum degree vertex from each remaining subgraph. This greedy removal process can be performed in linear time and allows the degeneracy to be computed in linear time [65].

Definition 3.4. *As with degree, we define a family $\mathscr{F}$ of graphs to have* bounded degeneracy *if all graphs in $\mathscr{F}$ have degeneracy $O(1)$.*

That is, family $\mathscr{F}$ has bounded degeneracy if there is an upper bound on the degeneracy of the graphs in $\mathscr{F}$ that may depend on $\mathscr{F}$ itself but that does not depend on the choice of a graph within $\mathscr{F}$.

Definition 3.5. The *hop count* from a vertex u to vertex v in a graph is the minimum number of edges between them. The *radius* of a graph or subgraph is the smallest number r such that there exists a vertex within hop count of r of all other vertices. An *r-shallow minor*

of a graph G is a graph obtained from G by possibly deleting some edges and/or vertices of G, and then contracting some radius-r subgraphs of the remaining graph into supervertices. A family $\mathscr{F}$ of graphs has *bounded expansion* if, for all choices of the parameter r, the r-shallow minors of the graphs in $\mathscr{F}$ have bounded edge/vertex ratio. More strongly, family $\mathscr{F}$ has *polynomial expansion* if this edge/vertex ratio is bounded by a polynomial in r.

In particular, for $r = 0$ the shallow minors are just the subgraphs, so a family of graphs with bounded expansion or polynomial expansion must have subgraphs with bounded edge/vertex ratio. This implies that they necessarily also have bounded degeneracy. The graphs of polynomial expansion include the k-apex graphs, k-genus graphs, and k-planar graphs [68], described in our earlier discussion of near-planar families of graphs (Section 3.2.2).

Graph families of polynomial expansion can also be characterized in terms of separators, small sets of vertices that partition the graph and form the basis of many divide-and-conquer graph algorithms.

Definition 3.6. For an n-vertex graph G and a constant $c < 1$ we define a c-separator to be a subset S of vertices of G such that every connected component of $G \setminus S$ (the subgraph formed by deleting S from G) has at most cn vertices. We say that a family $\mathscr{F}$ of graphs has *sublinear separators* if there exist constants c, d, and e, with $c < 1$ and $e < 1$, such that every n-vertex graph in $\mathscr{F}$ has a c-separator of size (number of vertices) at most dn^e.

For instance, the famous planar separator theorem states that planar graphs have 2/3-separators of size $O(\sqrt{n})$, so we can take $c = 2/3$, $d = O(1)$, and $e = 1/2$ [62]. A *separator hierarchy* is formed by taking separators recursively, until all remaining components have size $O(1)$; for planar graphs it can be constructed in linear time [47], and enables linear-time computation of shortest paths [59], among other problems.

Then, for a family $\mathscr{F}$ of graphs that is closed under taking subgraphs (every subgraph of a graph in $\mathscr{F}$ is also in $\mathscr{F}$), $\mathscr{F}$ has polynomial expansion if and only if $\mathscr{F}$ has sublinear

separators [34]. The graphs of polynomial expansion have other important algorithmic properties, not directly deriving from their separators; for instance, for every fixed pattern graph H it is possible to test whether H is a subgraph of a graph in a family of bounded expansion (the subgraph isomorphism problem) in linear time. More generally, one can test any property that can be formulated in the first-order logic of graphs, for members of a family of graphs of bounded expansion, in linear time [68].

3.4.2 Classification of nonplanarities

We distinguish between two kinds of nonplanarity in a road network.

Definition 3.7. An *embedding* of a graph is a mapping from its vertices to points and its edges to curves, such that the vertices at the ends of each edge are mapped to the points at the ends of the corresponding curve. A *crossing* is a point where two edge curves intersect that is not a common endpoint of both curves. A *removable crossing* is a crossing between two edges in an embedding of a graph that can be removed (without introducing other crossings) by making only local reroutings in the embedding.

Such a crossing may occur, for instance, when an actual segment of road follows a curved path, passing between segments of other roads without crossing them, but the network segment representing it follows a different straight path that crosses nearby road segments. In such a case, re-routing the segment to follow the actual path of the road will cause these crossings to go away.

Definition 3.8. An *essential crossing* is a crossing between two edges in an embedding of a graph that represent disjoint (non-intersecting) road segments and that cannot be removed by local changes.

Figure 3.4: High Five Interchange in Dallas, Texas. CC-BY image File:High Five.jpg by fatguyinalittlecoat from Wikimedia commons.

Such a crossing may be caused, for instance, when one road follows a tunnel that crosses under several other roads (Figure 3.3), or by the multiple crossing segments of a highway interchange (Figure 3.4).

It is possible to remove the essential crossings of a road network by replacing each crossing by an artificial intersection point, one that exists in the graph but not in the actual road network that it represents. We call a planar network constructed in this way the *planarization* of the road network. Indeed, the TIGER data set, commonly used for experiments on road networks, has been planarized in this way. If the edges in this data set are drawn as straight line segments, instead of following the curves of the actual roads that the edges represent, some crossings may arise from the straightening, but the underlying graph of this data set is planar. For this reason, our experiments use a different data set, the Urban Road Network Dataset, that has not already been planarized [57].

However, planarized networks cannot be used to obtain correct results for many road network computations. For instance, using the planarization of a road network in a shortest path

routing algorithm would potentially create routes that turn from one road to another at the artificial intersections added in planarization, which do not form usable routes in the real-world network. For this reason, it is problematic to use algorithms designed specifically to work in planar networks, such as the known linear-time planar graph shortest path algorithms [59], on road network data. Instead, we must show that despite their non-planarities, road networks have the underlying properties necessary to support generalized versions of these efficient algorithms.

3.5 Experiments

In this section, we examine the sparsity of crossing graphs experimentally, on real-world road networks. We compute and analyze the crossing graphs for the 80 of the most populated urban areas in the world. Our analysis uses the Urban Road Network Dataset [57].

The Urban Road Network Dataset includes graphs with self-loops and parallel edges. We removed self-loops and parallel edges before processing the data. The data contains both essential and removable crossings.

To find the crossings in the Urban Road Network Dataset, we used a plane sweep algorithm for line segment intersection detection [15], as implemented in the CGAL computational geometry library [41]. Although theoretically-faster algorithms are known for the type of data considered here, in which the segments form a connected geometric graph with few crossings [37], CGAL's plane sweep is practical and usable, and the slight superlinearity of its time bound is not problematic for the problem sizes we tested.

Because a perfect identification of essential crossings would require the solution of the NP-hard problem of minimizing the number of crossings in graph embeddings [45], our experiments determined whether a crossing is essential or removable heuristically, by using the fact that

City	Roads	Crossings	Uncrossed roads	Degener--acy	Degree	Trees	Non-trees
Ankara	111054	2713	107943	3	16	643	126
Atlanta	381649	7505	372135	4	33	2501	236
Bangdung	26154	537	25555	2	23	140	16
Bangkok	188069	9940	178225	5	38	1637	354
Barcelona	296969	13218	281762	4	38	3309	483
Beijing	110115	13491	98339	5	71	1915	525
BeloHorizonte	94207	2797	91221	4	23	531	101
Bengaluru	200828	3102	197273	4	36	846	100
Bogota	199846	5929	194621	5	166	960	110
Boston	470360	9324	458955	5	25	2830	295
BuenosAires	435039	5344	429340	6	60	1083	201
Calcutta	90283	991	88995	3	18	374	21
Chengdu	25847	3278	23076	4	100	412	118
Chongqing	27251	3899	23207	4	18	658	215
Dalian	18017	1169	16638	3	21	292	60
Dallas	548721	15287	532025	4	52	3136	746
Delhi	76240	1853	74240	3	24	364	94
Detroit	390597	8163	381056	4	34	2070	304
Dhaka	19922	477	19445	2	20	120	8
Dongguan	11076	1773	9557	4	28	183	96
Fuzhou	16093	1948	14496	4	23	247	83
Guangzhou	69410	9765	60238	4	26	1234	570
Hangzhou	28270	3433	25238	4	26	459	164
Harbin	14805	1383	13465	3	38	195	76
HoChiMinh	110953	1627	109087	3	13	469	50
Houston	632289	13337	616864	4	28	3255	633
Hyderabad	172822	2461	169948	3	166	701	36
Istambul	380466	12002	368090	4	56	2059	421
Jacarta	105318	4811	100998	4	159	620	126
Johannesburg	267193	6715	259424	4	38	1710	247
Karachi	55362	1091	54236	3	38	206	29
Lahore	41115	1506	39689	3	61	188	35
London	376437	9015	365645	4	23	2757	231
LosAngeles	552690	17905	532472	5	50	4140	591
Madrid	511910	23482	487090	5	49	4349	790
Manila	328623	5518	321984	4	26	1664	122
Medellin	33310	1118	32174	4	20	225	46
Miami	312082	6396	304945	5	41	1387	281
Milan	262466	8512	252789	4	25	2227	309
Moscow	940251	20469	916398	4	40	6185	462
Mumbai	61299	2085	58976	3	20	514	66

Table 3.1: Crossing Graphs (both essential and removable crossings)

City	Roads	Crossings	Uncrossed roads	Degener--acy	Degree	Trees	Non-trees
Ankara	111054	1041	110053	3	9	144	89
Atlanta	381649	2081	379231	4	14	533	131
Bangdung	26154	175	25976	2	21	37	8
Bangkok	188069	6902	181482	5	35	956	296
Barcelona	296969	5048	291548	4	36	1118	262
Beijing	110115	10772	101255	5	71	1314	450
BeloHorizonte	94207	1047	93155	3	11	187	54
Bengaluru	200828	1158	199700	3	35	190	59
Bogota	199846	3270	197517	5	18	357	65
Boston	470360	3668	466144	5	23	916	182
BuenosAires	435039	3217	431874	6	60	559	156
Calcutta	90283	363	89894	2	15	70	15
Chengdu	25847	2385	23951	4	99	257	89
Chongqing	27251	2163	25020	4	11	368	154
Dalian	18017	757	17126	3	21	186	44
Dallas	548721	7694	541247	4	40	966	587
Delhi	76240	836	75437	3	10	100	71
Detroit	390597	3123	387062	4	10	709	182
Dhaka	19922	247	19711	2	7	56	5
Dongguan	11076	1162	10133	4	26	101	70
Fuzhou	16093	1228	15134	3	14	130	64
Guangzhou	69410	6817	63105	4	25	752	489
Hangzhou	28270	2462	26166	3	23	305	133
Harbin	14805	896	13973	3	38	113	51
HoChiMinh	110953	616	110325	3	12	116	33
Houston	632289	6769	625518	4	19	894	537
Hyderabad	172822	998	171967	2	166	89	14
Istambul	380466	4391	376409	4	31	573	272
Jacarta	105318	2338	103588	4	158	171	76
Johannesburg	267193	1871	265257	3	38	379	130
Karachi	55362	460	54931	3	28	76	21
Lahore	41115	523	40632	3	57	49	22
London	376437	4946	370890	4	23	1350	147
LosAngeles	552690	8941	543170	5	16	1683	412
Madrid	511910	8734	503015	5	47	1598	383
Manila	328623	2276	326260	4	19	505	72
Medellin	33310	635	32765	3	12	82	32
Miami	312082	2707	309459	5	13	294	235
Milan	262466	4297	258068	4	25	887	216
Moscow	940251	9686	930967	4	37	1947	302
Mumbai	61299	1218	60081	3	20	224	50

Table 3.2: Crossing Graphs (essential crossings only)

the data associated with each road segment in the Urban Road Network Dataset indicates whether it is a bridge or tunnel. Our heuristic is that when a crossing occurs between two road segments neither of which is a bridge or a tunnel, then it is removable. However, the Urban Road Network Dataset only includes this bridge and tunnel labeling for a subset of the cities that it covers. For this reason, we restricted our experiments to this subset.

We used the NetworkX Python package [53] to study the structure of the crossing graphs we constructed.

3.5.1 Hypothesis

Based on our intuitions concerning bridges and tunnels in real-world road networks, we expected the crossing graphs to include some vertices of moderate degree, but otherwise to be very sparse. For instance, we considered it to be possible that all of the connected components of the crossing graph would be trees.

3.5.2 Results

The results of our experiments can be seen in Table 3.1 and 3.2. We have given some of the key properties of the crossing graphs of road networks for 41 out of 80 cities in the table. The table columns labeled "Roads" and "Crossings" give the total number of nodes and edges, respectively, in each crossing graph; that is, the numbers of road segments and crossings in the original road network. The column giving the number of uncrossed roads lists the number of nodes in the crossing graph which have no incident edges; that is, the number of uncrossed road segments. These isolated nodes constitute the vast majority of crossing graph nodes. Table 3.1 captures both essential and removable crossings whereas Table 3.2 captures only essential crossings.

The columns labeled "Degeneracy" and "Degree" give the degeneracy and maximum degree, respectively, of the crossing graph, as defined in Section 3.4.1. The remaining columns detail the number of components that are trees and the number of components that are not trees, respectively, in the crossing graph.

3.5.3 Analysis

The table show that indeed these graphs are sparse. More specifically, our hypothesis that the degeneracy would be significantly smaller than the maximum degree held up in the experiments. Although it is not true that (as we hypothesized) all components of the crossing graphs are trees, most of them are. The remaining non-tree components all have low degeneracy (at most 6).

3.6 Theoretical Analysis of Graphs with Sparse Crossings

Both our experimental results, and our consideration of nonplanar tunnels and interchanges forming essential crossings in (non-planarized) real road networks, motivate the following question. Suppose that, as the experiments appear to show, road networks have sparse crossing graphs. More precisely, suppose that their crossing graphs have bounded degeneracy, but not necessarily bounded degree. What does this imply about the graph-theoretic properties of road networks? Do they have bounded degeneracy? Polynomial expansion? Here we give positive answers to both of these questions.

First, let us formalize the notion of a graph with a sparse crossing graph.

Definition 3.9. We define a *nice embedding* to be a mapping of the vertices of the graph to points in the plane, and the edges to curves, such that the following conditions are all met:

- Each edge is mapped to a Jordan arc (a non-self-intersecting curve) whose endpoints are the images of the endpoints of the edge.
- If an edge and a vertex are disjoint in the graph, their images in the plane are disjoint.
- If two edges are mapped to curves that intersect, then that intersection consists of a single point, and is either a shared endpoint of both edges or a point where their two curves cross.
- No three edges have curves that all cross at the same point.

Let $\mathscr{C}_d$ denote the family of embedded graphs with nice embeddings, such that the crossing graphs of these embeddings have degeneracy at most d.

Definition 3.10. *We say that a graph G is* d-crossing-degenerate *if it belongs to $\mathscr{C}_d$.*

In order to apply the equivalence between separator theorems and polynomial expansion of Dvořák and Norin [34], we need to verify that the class of graphs we care about is closed under subgraphs.

Lemma 3.1. *Every subgraph of a graph in $\mathscr{C}_d$ also belongs to $\mathscr{C}_d$.*

Proof. Let G be in $\mathscr{C}_d$ and let H be a subgraph of G. Embed H by deleting the edges and vertices of $G \setminus H$ from the embedding of G. Then the crossing graph of the resulting embedding of H is an induced subgraph of the crossing graph of the embedding of G. Since taking induced subgraphs cannot increase the degeneracy, the degeneracy of the crossing graph of H is at most d. Therefore, H belongs to $\mathscr{C}_d$. □

Next, we examine the number of crossings that a graph in this family can have. As we show below, a linear bound on the crossing number follows directly from the assumption of low crossing graph degeneracy.

Lemma 3.2. *For the graphs in $\mathscr{C}_d$, a graph with m edges has at most dm crossings.*

Proof. We can reduce any graph in $\mathscr{C}_d$ to one with no edges (and no crossings) by repeatedly removing an edge that is crossed by at most d other edges, using the assumption that the crossing graph is d-degenerate and therefore that there exists an edge that is crossed at most d times. This process eliminates at most d crossings per step and takes m steps, so there are at most dm crossings in the given graph. □

In contrast, graphs with many edges are known to have many crossings per edge. We use the following well-known *crossing number inequality* of Ajtai, Chvátal, Newborn, Szemerédi, and Leighton [1, 60, 69]:

Lemma 3.3. *Let G be an embedded graph with n vertices and m edges, with $m \geq 4n$. Then G has $\Omega(m^3/n^2)$ crossings.*

This allows us to show that the d-crossing-degenerate graphs are sparse.

Lemma 3.4. *Every n-vertex embedded graph G in $\mathscr{C}_d$ has $O(n\sqrt{d})$ edges.*

Proof. Let the number of edges in G be $\gamma n\sqrt{d}$, for some parameter γ. Then by Lemma 3.3 the number of crossings is $\Omega(n(\gamma\sqrt{d})^3)$ and the number of crossings per edge is $\Omega((\gamma\sqrt{d})^2) = \Omega(\gamma^2 d)$. However, by Lemma 3.2 the number of crossings per edge is also at most d. For these two things to both be true, it must be the case that $\gamma = O(1)$, so the number of edges in G is $O(n\sqrt{d})$. □

Definition 3.11. Given a nicely embedded graph G, we define the *planarization* $P(G)$ to be the planar graph that has a vertex for each vertex or crossing point of G, and an edge for each maximal segment of an edge curve of G that does not contain a crossing point.

It follows from our previous lemmas that d-crossing-degenerate graphs have small planarizations.

Lemma 3.5. *For the graphs in $\mathscr{C}_d$, every n-vertex graph G has a planarization with $O(nd^{3/2})$ vertices and edges.*

Proof. This follows immediately from the already-proven facts that G itself has $O(n\sqrt{d})$ edges (Lemma 3.4) and an average of at most d crossings per edge (Lemma 3.2). □

Using these lemmas, we can prove our main result, that these graphs have sublinear separators.

Theorem 3.1. *The graphs in $\mathscr{C}_d$ have sublinear separators and polynomial expansion. A sublinear separator hierarchy for these graphs can be constructed from their planarizations in linear time.*

Proof. To prove the existence of sublinear separators, we apply the planar separator theorem [62] to the planarization $P(G)$ of a graph G in $\mathscr{C}_d$. By Lemma 3.5 $P(G)$ is larger than G by at most a factor depending only on d, so (treating d as a constant for the purposes of O-notation) $P(G)$ has separators of size $O(\sqrt{n})$. One application of the separator theorem may produce components of $P(G)$ that are larger than the number n of vertices in G, but recursively applying the separator theorem a bounded number of times will produce components that are smaller than n by a constant factor. The resulting separators have vertices in $P(G)$, corresponding to crossings in G; they can be transformed into separators in G itself by replacing each crossing vertex in the separator by the set of four endpoints of the corresponding crossing edges in G. This replacement only increases the separator size by a constant factor.

Since this method uses only planarization and planar separators, we may apply the linear time method for constructing planar separator hierarchies to $P(G)$ [47], to get a separator hierarchy to G as well.

The fact that these graphs have polynomial expansion follows from the existence of sublinear separators, and from the fact that $\mathscr{C}_d$ is closed under subgraphs (Lemma 3.1). □

By applying the method of [59], we obtain:

Corollary 1. *If we are given the planarization of a graph G in $\mathscr{C}_d$, we can compute shortest paths in G itself in linear time.*

In conjunction with known fast algorithms for finding planarizations [37], this leads to a linear-time algorithm for shortest paths whenever the number of crossings is sufficiently smaller than the overall number of road segments (as it was in our experiments).

3.7 Conclusions

We have performed a computational study of the removable crossings in large-scale planarized road network data. Our study shows that these crossings form a crossing graph that has high-degree vertices (up to degree 166), but that most connected components of the crossing graph are trees and that the few remaining components have maximum degeneracy six.

Based on our study, we developed a model of nearly-planar graphs, the d-crossing-degenerate graphs, consisting of the graphs that can be embedded with d-degenerate crossing graphs. We showed that this family of graphs is closed under the operation of taking subgraphs. In addition, for constant values of d, these graphs have a linear number of crossings, a linear number of edges, and separators of size proportional to the square root of the number of vertices. In addition, a separator hierarchy for these graphs can be constructed in linear time,

and applied in separator-based divide and conquer algorithms for shortest paths and other computational problems on road networks.

Chapter 4

Subexponential-Time and FPT Algorithms for C-Planarity Testing[1]

4.1 Introduction

A *clustered graph* (or *c-graph*) is a pair $\mathcal{C}(G, \mathcal{T})$ with *underlying graph* G and *inclusion tree* $\mathcal{T}$, i.e., a rooted tree whose leaves are the vertices of G. Each internal node μ of $\mathcal{T}$ represents a cluster of vertices of G (its leaf descendants) which induces a subgraph $G(\mu)$. A *c-planar drawing* of $\mathcal{C}(G, \mathcal{T})$ (Fig. 4.1) consists of a drawing of G and of a representation of each cluster μ as a *simple closed region* $R(\mu)$, i.e., a region homeomorphic to a closed disc, such that: (1) Each region $R(\mu)$ contains the drawing of $G(\mu)$. (2) For every two clusters $\mu, \nu \in \mathcal{T}$, $R(\nu) \subseteq R(\mu)$ if and only if ν is a descendant of μ in $\mathcal{T}$. (3) No two edges cross. (4) No edge crosses any region boundary more than once. (5) No two region boundaries intersect.

An interesting and challenging line of research in graph drawing concerns the computational complexity of the C-Planarity problem, which asks to test the existence of a c-planar

[1]This chapter is included with permission from Springer [28].

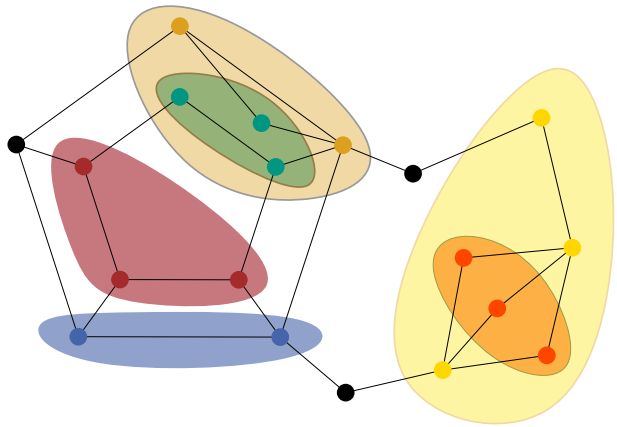


Figure 4.1: A c-planar drawing

drawing of a c-graph. This problem is notoriously difficult, particularly when (as in Fig. 4.1) clusters may be disconnected, faces may have unbounded size, and the cluster hierarchy may have multiple nested levels. No known subexponential-time algorithm solves the (general) C-Planarity problem, and it is unknown whether it is NP-complete, although the related problem of splitting as few clusters as possible to make a c-graph c-planar was proved NP-hard [6]. Thus, there is considerable interest in subexponential-time, slice-wise polynomial, and fixed-parameter tractable algorithms, besides polynomial-time algorithms for special cases of C-Planarity.

C-Planarity was introduced by Feng, Cohen, and Eades [42], who solved it in quadratic time for the *c-connected* case when every cluster induces a connected subgraph. Similar results were given by Lengauer [61] using different terminology. Dahlhaus [29] claimed a linear-time algorithm for c-connected C-Planarity (with some details later provided by Cortese *et al.* [25]). Goodrich *et al.* [49] gave a cubic-time algorithm for disconnected clusters that satisfy an "extroverted" property, and Gutwenger *et al.* [52] provided a polynomial-time algorithm for "almost" c-connected inputs. Cornelsen and Wagner showed polynomiality for completely connected c-graphs, i.e., c-graphs for which not only every cluster but also the complement of each cluster is connected [24]. FPT algorithms have also been investigated [14, 21]. For additional special cases, see, e.g., [3–5, 9, 20, 32].

A c-graph is *flat* when no non-trivial cluster is a subset of another, so $\mathcal{T}$ has only three levels: the root, the clusters, and the leaves. Flat C-Planarity can be solved in polynomial time for embedded c-graphs with at most 5 vertices per face [31, 44] or at most two vertices of each cluster per face [19], for embedded c-graphs in which each cluster induces a subgraph with at most two connected components [55], and for c-graphs with two clusters [13, 44, 54] or three clusters [2]. At the other end of the size spectrum, Jelínková *et al.* [56] provide efficient algorithms for 3-connected flat c-graphs when each cluster has at most 3 vertices. Fulek [43] speculates that C-Planarity could be solvable in subexponential time for more general embedded flat c-graphs.

New Results. In this chapter, we provide subexponential-time and fixed-parameter tractable algorithms for broad classes of c-graphs. We show the following results:

- ◇ C-Planarity can be solved in subexponential time for embedded flat c-graphs with bounded face size (Section 4.3).
- ◇ C-Planarity is fixed-parameter tractable for embedded flat c-graphs with embedded-width and number of disconnected clusters as parameters (Section 4.4).

Our first result uses divide-and-conquer with a large but subexponential branching factor. It exploits cycle separators in planar graphs and a concise representation of the connectivity of each cluster in a c-planar drawing. This method also leads to an XP algorithm for *generalized h-simply nested graphs*, which extend simply-nested graphs with bounded face size (Section 4.3.1). Recall that, XP (short for *slice-wise polynomial*) is the class of parameterized problems with input size n and parameter k than can be solved in $O(n^{f(k)})$ time, where f is a computable function.

We obtain our second result by expressing c-planarity in extended monadic second-order logic for embedded flat c-graphs and applying Courcelle's Theorem. The graphs to which this

result applies, with bounded treewidth and bounded face size, include the *nested triangles graphs*, a standard family of examples that are hard for many graph drawing tasks, the *dual graphs of bounded-treewidth bounded-degree plane graphs* [17], and the *buckytubes*, graphs formed from a planar hexagonal lattice wrapped to form a cylinder of bounded diameter.

4.2 Definitions and Preliminaries

The graphs considered in this chapter are finite, simple, and connected. A graph is *planar* if it admits a drawing in the plane without edge crossings. A *combinatorial embedding* is an equivalence class of planar drawings, where two drawings of a graph are *equivalent* if they determine the same *rotation* at each vertex, i.e, the same circular orderings for the edges around each vertex. 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*. An *embedded graph* (*plane graph*) is a planar graph with a fixed combinatorial embedding (fixed planar embedding). The *length* of a face f is the number of occurrences of edges encountered in a traversal of f. The *maximum face size* of an embedded graph is the length of its largest face.

A graph is *connected* if it contains a path between any two vertices. A *cut-vertex* is a vertex whose removal disconnects the graph. A *separation pair* is a pair of vertices whose removal disconnects the graph. A connected graph is 2-*connected* if it contains at least 3 vertices and does not have a cut-vertex, and a 2-connected graph is 3-*connected* if it contains at least 4 vertices and does not have a separation pair. The *blocks* of a graph are its maximal 2-connected subgraphs. Any (subdivision of a) 3-connected planar graph admits a unique combinatorial embedding (up to a flip) [80].

Tree-width and Embedded-width. A *tree decomposition* of a graph G is a tree T whose nodes, called *bags*, are labeled by subsets of vertices of G. For each vertex v the bags containing v must form a nonempty contiguous subtree of T, and for each edge uv at least one bag must contain both u and v. The *width* of the decomposition is one less than the maximum cardinality of any bag, and the *treewidth* $\mathrm{tw}(G)$ of G is the minimum width of any of its tree decompositions.

Recently, Borradaile *et al.* [16] developed a variant of treewidth, specialized for plane graphs, called embedded-width. According to their definitions, a tree decomposition *respects* an embedding of a plane graph G if, for every inner face f of G, at least one bag contains all the vertices of f. They define the *embedded-width* $\mathrm{emw}(G)$ of G to be the minimum width of a tree decomposition that respects the embedding of G. We will use the following result [16].

Theorem 4.1 ([16], Theorem 2)**.** *If G is a plane graph where every inner face has length at most ℓ, then* $\mathrm{emw}(G) \leq (\mathrm{tw}(G) + 2) \cdot \ell - 1$.

Borradaile *et al.* do not require the vertices of the outer face to be contained in a same bag. In our applications, we modify this concept so that the tree decomposition also includes a bag containing the outer face, and we denote the minimum width of such a tree decomposition as $\overline{\mathrm{emw}}(G)$. We have the following.

Lemma 4.1. *If G is a plane graph whose maximum face size (including the size of the outer face) is ℓ, then* $\overline{\mathrm{emw}}(G) = O(\ell \cdot \mathrm{tw}(G))$.

Proof. To prove the statement, we can proceed as follows.

We augment G to a graph G', by embedding G in the interior of a triangle Δ and by identifying one of the vertices of the outer face of G with a vertex of Δ. Clearly, $tw(G') = \max(\mathrm{tw}(G), 2)$ and G' has maximum face size $\ell' = O(\ell)$. Also, we have that $\overline{\mathrm{emw}}(G) \leq \mathrm{emw}(G')$, since $G \subseteq G'$ and since all the vertices incident to the same face in G are also incident to

the same face in G'. Thus, the statement follows from the fact that, by Theorem 4.1, $\text{emw}(G') \leq (\text{tw}(G') + 2) \cdot \ell' - 1$. □

Clustered Planarity. Recall that, in a c-graph $\mathcal{C}(G, \mathcal{T})$, each internal node μ of $\mathcal{T}$ corresponds to the set $V(\mu)$ of vertices of G at leaves of the subtree of $\mathcal{T}$ rooted at μ. Set $V(\mu)$ induces the subgraph $G(\mu)$ of G. We call the internal nodes other than the root *clusters*. A cluster μ is *connected* if $G(\mu)$ is connected and *disconnected* otherwise. A c-graph $\mathcal{C}(G, \mathcal{T})$ is *c-connected* if every cluster is connected.

A c-graph is *c-planar* if it admits a c-planar drawing. Two c-graphs $\mathcal{C}(G, \mathcal{T})$ and $\mathcal{C}'(G', \mathcal{T}')$ are *equivalent* if both are c-planar or neither is. If the root of $\mathcal{T}$ has leaf children, enclosing each leaf v in a new singleton cluster produces an equivalent c-graph. Therefore, we can safely assume that each vertex belongs to a cluster. A c-graph is *flat* if each leaf-to-root path in $\mathcal{T}$ has exactly three nodes. The clusters of a flat c-graph form a partition of the vertex set.

An *embedded c-graph* $\mathcal{C}(G, \mathcal{T})$ is a c-graph whose underlying graph has a fixed combinatorial embedding. It is *c-planar* if it admits a c-planar drawing that preserves the embedding of G. In what follows, we only deal with embedded flat c-graphs. Therefore, we will refer to such graphs simply as c-graphs.

We define the *candidate saturating edges* of a c-graph $\mathcal{C}(G, \mathcal{T})$ as follows. For each face f of G, let $G(f)$ be the closed walk composed of the vertices and edges of f. For each cluster $\mu \in \mathcal{T}$, consider the set $\mathcal{X}_\mu(f)$ of connected components of $G(f)$ induced by the vertices of μ and, for each component $\xi \in \mathcal{X}_\mu(f)$, assign a vertex of f belonging to ξ as a reference vertex of ξ. We add an edge inside f between the reference vertices of any two components in $\mathcal{X}_\mu(f)$ if and only if such vertices belong to different connected components of $G(\mu)$; see Figs. 4.2a and 4.2b. A c-graph obtained from $\mathcal{C}(G, \mathcal{T})$ by adding to $\mathcal{C}$ a subset E^+ of its candidate saturating edges is a *super c-graph* of $\mathcal{C}$.

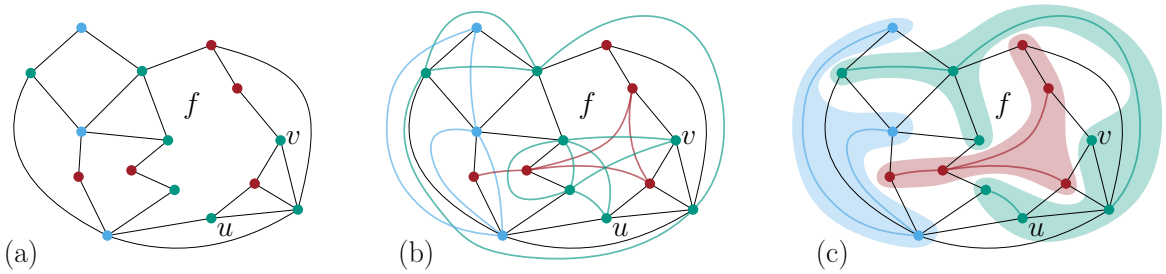


Figure 4.2: (a) An embedded flat c-graph $\mathcal{C}(G,\mathcal{T})$. (b) A super c-graph of $\mathcal{C}$ containing all the candidate saturating edges of $\mathcal{C}$ (thick and colored curves); since vertices u and v belong to different components of $X_\mu(f)$ but to the same connected component of $G(\mu)$, edge (u,v) is not a candidate saturating edge. (c) A super c-graph of $\mathcal{C}$ satisfying Condition (iii) of Theorem 4.2; regions enclosing vertices of each cluster are shaded.

Di Battista and Frati [31] gave the following characterization.

Theorem 4.2 ([31], Theorem 1)**.** *A c-graph $\mathcal{C}(G,\mathcal{T})$ is c-planar if and only if:*

(i) G is planar;

(ii) there exists a face f in G such that when f is chosen as the outer face for G no cycle composed of vertices of the same cluster encloses a vertex of a different cluster in its interior; and

(iii) there exists a super c-graph $\mathcal{C}'(G',\mathcal{T})$ of $\mathcal{C}$ such that G' is planar and $\mathcal{C}'$ is c-connected (see Fig. 4.2c).

Conditions (i) and (ii) of Theorem 4.2 can be easily verified in linear time. Therefore, we can assume that any c-graph satisfies these conditions. Following [31] we thus view the problem of testing c-planarity as one of testing Condition (iii).

A *cluster-separator* in a c-graph $\mathcal{C}(G,\mathcal{T})$ is a cycle ρ in G for which some cluster $\mu \in \mathcal{T}$ has vertices both in the interior and in the exterior of ρ but with $V(\mu) \cap V(\rho) = \emptyset$. Condition (iii) immediately yields the following observation.

Observation 1. *A c-graph that has a cluster-separator is not c-planar.*

Figure 4.3: Transformations for the proof of Lemma 4.2.

In the next sections, it will be useful to only consider c-graphs which are at least 2-connected (Section 4.3) and 3-connected (Section 4.4). The next lemma, conveniently stated in a stronger form[2], shows that this is not a loss of generality.

Lemma 4.2. *Let $\mathcal{C}(G,\mathcal{T})$ be an n-vertex c-graph with maximum face size ℓ. There exists an $O(n)$-time algorithm that constructs an equivalent c-graph $\mathcal{C}^*(G^*,\mathcal{T}^*)$ with $|V(G^*)| = O(n)$ such that: 1. G^* is 3-connected, 2. the maximum face size κ of G^* is $O(\ell)$, and 3. the c-graph $\mathcal{C}^\diamond(G^\diamond,\mathcal{T}^\diamond)$ obtained by augmenting $\mathcal{C}^*(G^*,\mathcal{T}^*)$ with all its candidate saturating edges is such that* $\mathrm{tw}(G^\diamond) = O(\overline{\mathrm{emw}}(G))$.

Proof. To prove the statement, we can proceed as follows.

First, we transform c-graph $\mathcal{C}(G,\mathcal{T})$ into an equivalent c-graph $\mathcal{C}'(G',\mathcal{T}')$, by applying the transformation in Fig. 4.3a to every edge, such that $|V(G')| = O(|V(G)|)$, every vertex of G' has degree at least 3, the maximum face size ℓ' of G' is $O(\ell)$, $tw(G') = \mathrm{tw}(G)$, and each vertex u of G' is incident to at least three edges in each block u belongs to.

Second, we transform c-graph $\mathcal{C}'(G',\mathcal{T}')$ into an equivalent c-graph $\mathcal{C}^*(G^*,\mathcal{T}^*)$, by applying the transformation in Fig. 4.3b to every vertex, such that $|V(G^*)| = O(|V(G')|)$, G^* is 3-connected, the maximum face size ℓ^* of G^* is $O(\ell')$, and the c-graph $\mathcal{C}^\diamond(G^\diamond,\mathcal{T}^\diamond)$ obtained by augmenting $\mathcal{C}^*(G^*,\mathcal{T}^*)$ with all its candidate saturating edges is such that $\mathrm{tw}(G^\diamond) = O(\ell \cdot \mathrm{tw}(G))$, which implies that $\mathrm{tw}(G^\diamond) = \overline{\mathrm{emw}}(G)$, since $\overline{\mathrm{emw}}(G) = O(\ell \cdot \mathrm{tw}(G))$ by Lemma 4.1.

[2]In Section 4.4, we exploit all the properties of the lemma. In Section 4.3, we only exploit the existence of an equivalent 2-connected c-graph with maximum face size $\kappa = O(\ell)$.

We now describe each of the transformations in detail.

First, initialize $\mathcal{C}' = \mathcal{C}$. For every vertex c of G, let (c, x) be any edge incident to c. Add to G' vertices w_1 and w_2 and embed paths (c, w_1, x) and (c, w_2, x) in the interior of each of the two faces of G' edge (c, x) is incident to; also, subdivide edge (c, x) with a vertex w, add edges (w_1, w) and (w_2, w), and assign vertices w_1, w_2, and w to the same cluster of $\mathcal{T}'$ ($\mathcal{T}$) vertex c belongs to. Refer to Fig. 4.3a. By construction, all the newly added vertices have degree at least 3. In particular, observe that each cut-vertex of G' is incident to at least three edges in each of the blocks such a cut-vertex belongs to. It is easy to see that $\mathcal{C}'$ and $\mathcal{C}$ are equivalent. Also, the maximum face size ℓ' of G' is $O(\ell)$. Further, $\text{tw}(G') = \max(\text{tw}(G), 3)$, as the transformation replaces edges with subgraphs of treewidth 3.

Second, initialize $\mathcal{C}^* = \mathcal{C}'$. For every vertex c of G', we subdivide each edge (c, x_i) incident to c with a dummy vertex v_i. Denote such a graph by G^+. Also, add an edge between any two vertices v_i and v_j such that edges (c, x_i) and (c, x_j) are consecutive around c in the unique face shared by v_i and v_j in G^+. Finally, assign each vertex v_i to the same cluster of $\mathcal{T}^*$ ($\mathcal{T}'$) vertex c belongs to. Refer to Fig. 4.3b. The equivalence between $\mathcal{C}^*(G^*, \mathcal{T}^*)$ and $\mathcal{C}'(G', \mathcal{T}')$ is again straightforward. Clearly, $|V(G^*)| = O(|V(G')|)$ and $\ell^* = O(\ell')$. Also, by the observation that the cut-vertices of G' are incident to at least three edges in each of the blocks such cut-vertices belong to, the applied transformation fixes the rotation at all the vertices of G^*. Since each vertex of G^* has minimum degree 3 and G^* has a fixed combinatorial embedding (up to a flip), by the result of Whitney [80], we have that G^* is 3-connected. Furthermore, $\overline{\text{emw}}(G^*) = O(\overline{\text{emw}}(G'))$, since G^* is obtained by subdividing each edge of G' twice, thus obtaining a graph G^+ with the same tree-width as G' and maximum face-size in $O(\ell')$, and by adding edges between some of the vertices incident to the faces of G^+. Since $G^\diamond$ is the graph obtained by adding all the candidate saturating-edges of G^* (recall that such edges only connect vertices in the same face of G^*), we have that $\text{tw}(G^\diamond) = O(\overline{\text{emw}}(G^*)) = O(\overline{\text{emw}}(G'))$. Since, by Lemma 4.1, $\overline{\text{emw}}(G') = O(\ell' \cdot \text{tw}(G'))$ and since $\ell' = O(\ell)$ and $\text{tw}(G') = O(\text{tw}(G))$,

we have that $\mathrm{tw}(G^\diamond) = O(\ell \cdot \mathrm{tw}(G))$. This concludes the proof of the lemma. □

4.3 A Subexponential-Time Algorithm for C-Planarity

In this section, we describe a divide-and-conquer algorithm for testing the c-planarity of 2-connected c-graphs exploiting cycle separators in planar graphs.

The "conquer" part of our divide-and-conquer uses the following operation on pairs of c-graphs. Let G_1 and G_2 be plane graphs on overlapping vertex sets such that the outer face of G_1 and an inner face of G_2 are bounded by the same cycle ρ. *Merging* G_1 and G_2 constructs a new plane graph G from $G_1 \cup G_2$ as follows. We remove multi-edges (belonging to cycle ρ) and assign each vertex v a rotation whose restriction to the edges of G_2 (of G_1) is the same as the rotation at v in G_2 (in G_1). This is possible as cycle ρ bounds the outer face of G_1 and an inner face of G_2. We say that G is a *merge* of G_1 and G_2. Now consider two c-graphs $\mathcal{C}_1(G_1, \mathcal{T}_1)$ and $\mathcal{C}_2(G_2, \mathcal{T}_2)$ such that (i) $G_1 \cap G_2 = \rho$ is a cycle, (ii) for each vertex $v \in V(\rho)$, vertex v belongs to the same cluster μ both in $\mathcal{T}_1$ and in $\mathcal{T}_2$, and (iii) cycle ρ bounds the outer face of G_1 and an inner face of G_2 (when a choice for their outer faces that satisfies Condition (ii) of Theorem 4.2 has been made). *Merging* $\mathcal{C}_1$ and $\mathcal{C}_2$ is the operation that constructs a c-graph $\mathcal{C}(G, \mathcal{T})$ as follows. Graph G is obtained by merging G_1 and G_2. Tree $\mathcal{T}$ is obtained as follows. Initialize $\mathcal{T}$ to $\mathcal{T}_1$. First, for each cluster $\mu \in \mathcal{T}_2 \cap \mathcal{T}_1$, we add the leaves of μ in $\mathcal{T}_2$ as children of cluster μ in $\mathcal{T}$, removing duplicate leaves. Second, for each cluster $\mu \in \mathcal{T}_2 \setminus \mathcal{T}_1$, we add the subtree of $\mathcal{T}_2$ rooted at μ as a child of the root of $\mathcal{T}$. We say that $\mathcal{C}(G, \mathcal{T})$ is a *merge* of $\mathcal{C}_1(G_1, \mathcal{T}_1)$ and $\mathcal{C}_2(G_2, \mathcal{T}_2)$.

In the "divide" part of the divide-and-conquer, we replace subgraphs of the input by smaller planar components called *cycle-stars* that preserve their c-planarity properties. Let G be a connected plane graph that contains a face whose boundary is a cycle ρ. We say that G is a

cycle-star if removing all the edges of ρ makes G a forest of stars; refer to Fig. 4.4c. Also, we say that cycle ρ is *universal* for G and we say that a vertex of G is a *star vertex* of G if it does not belong to ρ. Clearly, the size of G is $O(|\rho|)$.

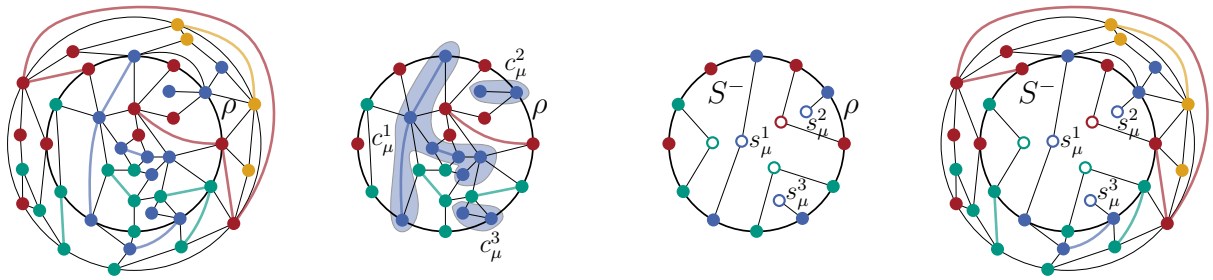
For a c-planar c-graph $\mathcal{C}(G, \mathcal{T})$ and a cycle separator ρ, we denote by $\mathcal{C}^+_\rho(G^+, \mathcal{T}^+)$ (by $\mathcal{C}^-_\rho(G^-, \mathcal{T}^-)$) the c-graph obtained from $\mathcal{C}$ by removing all the vertices and the edges of G that lie in the interior of ρ (in the exterior of ρ). Consider a super c-graph $\mathcal{C}'(G', \mathcal{T})$ of $\mathcal{C}$ satisfying Condition (iii) of Theorem 4.2, which exists since $\mathcal{C}$ is c-planar. We now give a procedure, which will be useful throughout the chapter, to construct two special c-planar c-graphs $\mathcal{C}^-(S^-, \mathcal{K}^-)$ and $\mathcal{C}^+(S^+, \mathcal{K}^+)$ associated with $\mathcal{C}'$ whose properties are described in the following lemma.

Lemma 4.3. *C-graphs $\mathcal{C}^-(S^-, \mathcal{K}^-)$ and $\mathcal{C}^+(S^+, \mathcal{K}^+)$ are such that:*

1. *graph S^- (S^+) is a cycle-star whose universal cycle is ρ,*
2. *cycle ρ bounds the outer face of S^- (an inner face of S^+),*
3. *each star vertex of S^- (S^+) and all its neighbours belong to the same cluster in $\mathcal{K}^-$ ($\mathcal{K}^+$), and*
4. *the c-graph $\mathcal{C}_{out}$ ($\mathcal{C}_{in}$) obtained by merging $\mathcal{C}^-(S^-, \mathcal{K}^-)$ and $\mathcal{C}^+_\rho(G^+, \mathcal{T}^+)$ (by merging $\mathcal{C}^+(S^+, \mathcal{K}^+)$ and $\mathcal{C}^-_\rho(G^-, \mathcal{T}^-)$) is c-planar.*

We describe how to construct $\mathcal{C}^-(S^-, \mathcal{K}^-)$ from $\mathcal{C}'$; refer to Fig. 4.4. The construction of $\mathcal{C}^+(S^+, \mathcal{K}^+)$ is symmetric.

First, for each cluster μ such that $V(\mu) \cap V(\rho) = \emptyset$, we remove all the vertices in $V(\mu)$ lying in the interior of ρ together with their incident edges. By Observation 1, the resulting c-graph is still c-planar and c-connected. Also, we remove edges in the interior of ρ whose endpoints belong to different clusters. Clearly, this simplification preserve c-connectedness. We still denote the resulting c-graph as $\mathcal{C}'$.



(a) $\mathcal{C}'$ **(b)** H **(c)** S^- **(d)** $\mathcal{C}^*$

Figure 4.4: (a) Super c-graph $\mathcal{C}'$ of $\mathcal{C}$. (b) Each component of the blue cluster μ in H lies inside a simple closed region. (c) Cycle-star S^- corresponding to H. (d) The c-connected c-planar c-graph $\mathcal{C}^*$ obtained by *replacing* H with S^- in $\mathcal{C}'$.

Second, consider the c-graph H consisting of the vertices and of the edges of $\mathcal{C}'$ lying in the interior and along the boundary of ρ. For each cluster μ and for each connected component c_μ^i of μ in H, we replace all the vertices and edges of c_μ^i lying in the interior of ρ in $\mathcal{C}'$ with a single vertex s_μ^i, assigning it to the same cluster μ and making it adjacent to all the vertices in $V(c_\mu^i) \cap V(\rho)$. Let $\mathcal{C}^*$ be the resulting c-graph. It is easy to see that such a transformation preserves c-connectedness and planarity. Therefore $\mathcal{C}^*$ is a c-connected c-planar c-graph. By construction, each vertex $v \in V(\rho)$ is adjacent to a single vertex s_μ^i, where μ is the cluster vertex v belongs to; thus, the vertices and the edges in the interior and along the boundary of ρ in $\mathcal{C}^*$ form c-graph $\mathcal{C}^-(S^-, \mathcal{K}^-)$ whose underlying graph S^- is a cycle-star satisfying Properties (1), (2) and (3) of Lemma 4.3. Further, since the subgraph of $\mathcal{C}^*$ consisting of the vertices and of the edges lying in the exterior and along the boundary of ρ coincides with $\mathcal{C}_\rho^+(G^+, \mathcal{T}^+)$, we have that $\mathcal{C}^*$ is a c-planar c-connected super c-graph of $\mathcal{C}_{out}$. Thus, by Condition (iii) of Theorem 4.2, Property (4) of Lemma 4.3 is also satisfied.

Let $\mathcal{C}_\Delta^-(R^-, \mathcal{J}^-)$ ($\mathcal{C}_\Delta^+(R^+, \mathcal{J}^+)$) be a c-graph obtained by augmenting the c-graph $\mathcal{C}^-(S^-, \mathcal{K}^-)$ ($\mathcal{C}^+(S^+, \mathcal{K}^+)$) of Lemma 4.3 by introducing new vertices, each belonging to a distinct cluster, and by adding edges only between the vertices in $V(S^-)$ ($V(S^+)$) and the newly introduced vertices in such a way that cycle ρ bounds a face of R^- (R^+) and all the other faces of R^- (R^+) are triangles. From the construction of Lemma 4.3, we also have the following useful

technical remark.

Remark 1. *The c-graph obtained by merging* $\mathcal{C}^-_\Delta(R^-, \mathcal{J}^-)$ *and* $\mathcal{C}^+_\rho(G^+, \mathcal{T}^+)$ *(by merging* $\mathcal{C}^+_\Delta(R^+, \mathcal{J}^+)$ *and* $\mathcal{C}^-_\rho(G^-, \mathcal{T}^-)$*) is c-planar.*

We now describe a divide-and-conquer algorithm based on Lemma 4.3, called TestCP, that tests the c-planarity of a 2-connected c-graph $\mathcal{C}(G, \mathcal{T})$ and returns a super c-graph $\mathcal{C}^*(G^*, \mathcal{T})$ of $\mathcal{C}$ satisfying Condition (iii) of Theorem 4.2, if $\mathcal{C}$ is c-planar. See Fig. 4.5 for illustrations of the c-graphs constructed during the execution of the algorithm.

We first give an intuition on the role of cycle-stars in Algorithm TestCP.

Let $\mathcal{C}(G, \mathcal{T})$ be a c-planar c-graph and let ρ be a cycle separator of G. By Lemma 4.3, for each c-connected c-planar super c-graph $\mathcal{C}'$ of $\mathcal{C}$, we can injectively map the super c-graph I^- of $\mathcal{C}^-_\rho(G^-, \mathcal{T}^-)$, composed of the vertices of G^- and of the edges in the interior and along the boundary of ρ in $\mathcal{C}'$, with a cycle-star S^- whose universal cycle is ρ. This is due to the fact that there exists a *one-to-one correspondence* between the connected components of I^- induced by the vertices of each cluster in $\mathcal{T}^-$ and the star vertices of S^-. Similar considerations hold for the super c-graph I^+ of $\mathcal{C}^+_\rho(G^+, \mathcal{T}^+)$. Although the c-planarity of $\mathcal{C}^+_\rho$ and $\mathcal{C}^-_\rho$ is necessary for the c-planarity of $\mathcal{C}$, it is not a sufficient condition, as the connectivity of clusters inside ρ in I^- (*internal cluster-connectivity*) and the connectivity of clusters outside ρ in I^+ (*external cluster-connectivity*) must also together determine the c-connectedness of $\mathcal{C}'$. The role of cycle-stars S^- and S^+ in the algorithm presented in this section is exactly that of concisely *representing* the internal cluster-connectivity of I^- and the external cluster-connectivity of I^+, respectively, to devise a divide-and-conquer approach to test the c-planarity of $\mathcal{C}$.

Outline of the algorithm. We overview the main steps of our algorithm below.

- If $n = O(\ell)$, we test c-planarity directly, as a base case for the divide-and-conquer

recursion. Otherwise, we construct a cycle-separator ρ of G and test whether ρ is a cluster-separator. If so, $\mathcal{C}$ cannot be c-planar (Observation 1), and we halt the search.

- We generate all cycle-stars S_i^- with universal cycle ρ. A cycle-star S_i^- represents a potential connection pattern of clusters inside ρ. For each cycle-star S_i^- we apply Procedure OuterCheck to test whether this pattern could be augmented by additional connections outside ρ to complete the desired cluster-connectivity. That is, we test whether $\mathcal{C}_\rho^+$ admits a c-connected c-planar super c-graph whose internal cluster-connectivity is represented by S_i^-. To test this, we replace the subgraph G^- of G in $\mathcal{C}$ with an internally-triangulated supergraph R_i^- of S_i^- to obtain a c-graph $\mathcal{C}^+$ and recursively test $\mathcal{C}^+$ for c-planarity. It is important to observe that, the triangulation step prevents $\mathcal{C}^+$ from having saturating edges inside ρ, thus enforcing exactly the same internal-cluster connectivity represented by S_i^- (Remark 1). If $\mathcal{C}^+$ is c-planar, the procedure returns a c-connected c-planar super c-graph $\mathcal{C}_{con}^+$ of $\mathcal{C}^+$. If no cycle-star passes the test, $\mathcal{C}$ is not c-planar by Lemma 4.3. We call all the cycle-stars that passed this test *admissible*.

- We then apply Procedure InnerCheck to verify whether the internal-cluster connectivity represented by some admissible cycle-star S_i^- can *actually* be realized by a c-connected c-planar super c-graph of $\mathcal{C}$. For each admissible cycle-star S_i^-, the procedure applies the construction of Lemma 4.3 to obtain a cycle-star S_i^+ representing the external cluster-connectivity of $\mathcal{C}_{con}^+$. Then, it tests whether $\mathcal{C}_\rho^-$ admits a c-connected c-planar super c-graph $\mathcal{C}_{con}^-$ whose external cluster-connectivity is represented by S_i^+. This is done similarly to Procedure OuterCheck, by triangulating the exterior of ρ and recursively testing c-planarity of a smaller graph. If Procedure InnerCheck succeeds for any admissible cycle-star S_i^-, then we can merge the subgraphs of $\mathcal{C}_{con}^-$ and of $\mathcal{C}_{con}^+$ induced by the vertices inside and outside ρ, respectively, to obtain a c-connected c-planar super c-graph of $\mathcal{C}$, and we halt the search with a successful outcome. It might

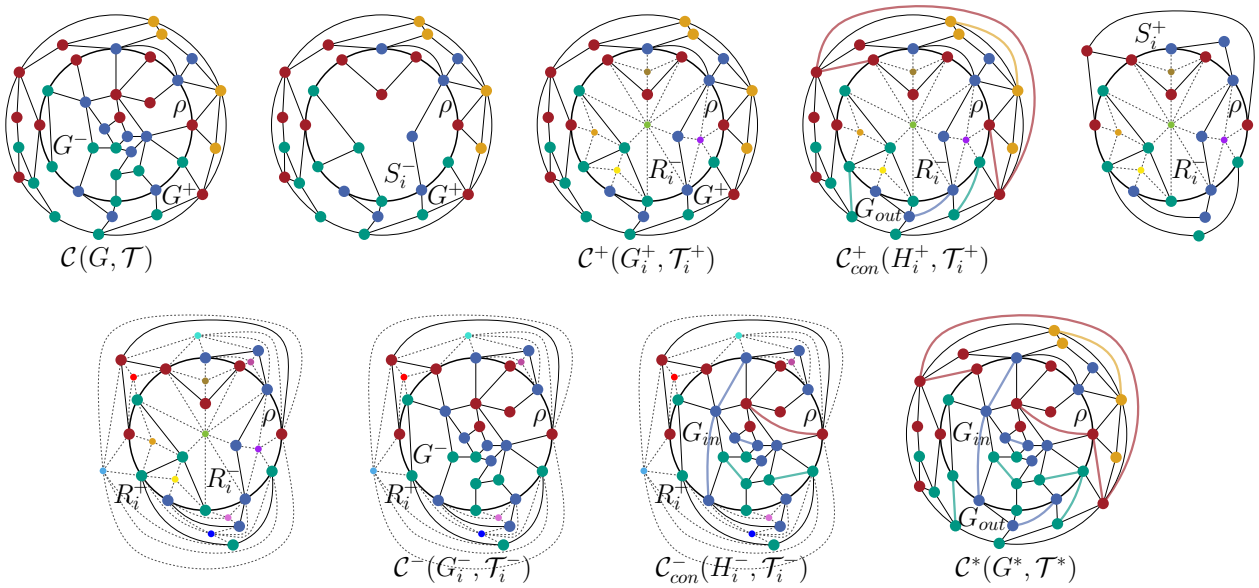


Figure 4.5: Illustrations of all of the c-graphs constructed by Algorithm TestCP.

be the case that $\mathcal{C}^-_{con}$ has a different internal-cluster connectivity than that represented by S_i^-, but this is not a problem, because the different cluster connectivity (which necessarily corresponds to a different admissible cycle-star) still provides a c-planar drawing of the whole graph.

- If no admissible cycle-star passes Procedure InnerCheck, $\mathcal{C}$ is not c-planar.

It is crucial in this algorithm that ρ be a cycle-separator. Because it is a cycle, no candidate saturating edges can connect vertices in the interior of ρ to vertices in the exterior of ρ, as such vertices do not share any face. That is, the interaction between G^-_ρ and G^+_ρ only happens through vertices of ρ. This allows us to split the instance into smaller instances recursively along ρ and model the interaction via cycle-stars (by Lemma 4.3 and Remark 1) whose universal cycle is ρ.

The complete listing of Algorithm TestCP is given on the next page.

[1]The merging operations are well defined as cycle ρ bounds the outer face of R_i^- and an inner face of G^+, as well as an inner face of R_i^+ and the outer face of G^-.

[2]As $\mathcal{C}^+(G_i^+,\mathcal{T}_i^+)$ and $\mathcal{C}^-(G_i^-,\mathcal{T}_i^-)$ are 2-connected, TestCP can be recursively applied.

Algorithm TestCP(C-GRAPH $\mathcal{C}(G,\mathcal{T})$):

BASE CASE

If $|V(G)| = O(\ell)$, then we can test C-PLANARITY for $\mathcal{C}(G,\mathcal{T})$ in $O(1)$ time when ℓ is a constant, by performing a brute force search to find a subset E' of the candidate saturating edges of $\mathcal{C}$ such that c-graph $\mathcal{C}'(G \cup E', \mathcal{T})$ satisfies Condition (iii) of Theorem 4.2.

RECURSIVE STEP

1. Select a cycle separator ρ of G. If ρ is a cluster-separator, then **return** `NO`; otherwise, construct c-graphs $\mathcal{C}^+_\rho(G^+,\mathcal{T}^+)$ and $\mathcal{C}^-_\rho(G^-,\mathcal{T}^-)$ as defined in Lemma 4.3.
2. OUTERCHECK
   (a) Construct the set $\mathcal{S}$ of all cycle-stars such that, for every $S \in \mathcal{S}$, it holds that (i) ρ is the universal cycle of S, (ii) ρ bounds the outer face of S, and (iii) every star vertex of S is incident only to vertices of ρ belonging to the same cluster.
   (b) For each cycle-star $S^-_i \in \mathcal{S}$:
      i. Construct a c-graph $\mathcal{C}^-(S^-_i,\mathcal{K}^-_i)$ as follows. First, initialize $\mathcal{K}^-_i$ to the subtree of $\mathcal{T}$ whose leaves are the vertices of S^-_i. Then, for each star vertex v of S^-_i, assign v to the cluster $\mu \in \mathcal{K}^-_i$ to which all its neighbours belong.
      ii. Augment $\mathcal{C}^-(S^-_i,\mathcal{K}^-_i)$ to an internally triangulated c-graph $\mathcal{C}^-_\Delta(R^-_i,\mathcal{J}^-_i)$ by introducing new vertices, each belonging to a distinct cluster, and by adding edges only between vertices in $V(S^-_i)$ and the newly introduced vertices (that is, no two non-adjacent vertices in S^-_i are adjacent in R^-_i).
      iii. Merge $\mathcal{C}^-_\Delta(R^-_i,\mathcal{J}^-_i)$ and $\mathcal{C}^+_\rho(G^+,\mathcal{T}^+)$ to obtain a c-graph $\mathcal{C}^+(G^+_i,\mathcal{T}^+_i)$[1].
      iv. Run TESTCP($\mathcal{C}^+(G^+_i,\mathcal{T}^+_i)$) to test whether $\mathcal{C}^+(G^+_i,\mathcal{T}^+_i)$ is c-planar[2].
   (c) If no c-graph $\mathcal{C}^+(G^+_i,\mathcal{T}^+_i)$ is c-planar, then **return** `NO`; otherwise, initialize $\mathcal{S}'$ as the set of *admissible* cycle-stars, i.e., the cycle-stars in $\mathcal{S}$ whose corresponding c-graph $\mathcal{C}^+$ is c-planar.

3. InnerCheck

   (a) For each admissible cycle-star $S_i^- \in \mathcal{S}'$:

      i. Let $\mathcal{C}^+_{con}(H_i^+, \mathcal{T}_i^+)$ be the c-planar c-connected super c-graph of $\mathcal{C}^+$ returned by TestCP($\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$) (step 2(b)iv). Apply the construction of Lemma 4.3 to c-graph $\mathcal{C}^+_{con}$ and cycle ρ to obtain a c-graph $\mathcal{C}^+(S_i^+, \mathcal{K}_i^+)$ satisfying Properties (2, 3) of the lemma.

      ii. Augment $\mathcal{C}^+(S_i^+, \mathcal{K}_i^+)$ to a c-graph $\mathcal{C}^+_\Delta(R_i^+, \mathcal{J}_i^+)$ by introducing new vertices, each belonging to a distinct cluster, and by adding edges only between the vertices in $V(S_i^+)$ and the newly introduced vertices in such a way that cycle ρ bounds an inner face of R_i^+ and all the other faces of R_i^+ are triangles.

      iii. Merge $\mathcal{C}^+_\Delta(R_i^+, \mathcal{J}_i^+)$ and $\mathcal{C}^-_\rho(G^-, \mathcal{T}^-)$ to obtain a c-graph $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$[1].

      iv. Run TestCP($\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$) to test whether $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$ is c-planar[2].

      v. If 3(a)iv returns `YES`, then construct a c-planar c-connected super c-graph $C^*(G^*, \mathcal{T})$ of $\mathcal{C}(G, \mathcal{T})$ as follows. Let $\mathcal{C}^-_{con}(H_i^-, \mathcal{T}_i^-)$ be the c-planar c-connected c-graph returned by TestCP($\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$). Remove all the vertices and edges of H_i^- in the exterior of cycle ρ, thus obtaining a new c-graph $\mathcal{C}_{in}(G_{in}, \mathcal{T}_{in})$ in which cycle ρ bounds the outer face. Similarly, remove all the vertices and edges of H_i^+ in the interior of cycle ρ, thus obtaining a new c-graph $\mathcal{C}_{out}(G_{out}, \mathcal{T}_{out})$ in which cycle ρ bounds an inner face. Finally, merge $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$ to obtain c-graph $\mathcal{C}^*(G^*, \mathcal{T})$ and **return** `YES` along with c-graph $\mathcal{C}^*(G^*, \mathcal{T})$.

4. **return** `NO` if no c-graph $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$, constructed at step 3(a)iii, is c-planar.

Base Case of the algorithm. The base case occurs when $\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$ and $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$ are no longer smaller than $\mathcal{C}(G, \mathcal{T})$.

Observe that, we obtained G_i^+ (G_i^-) by merging G^+ (G^-) and R_i^- (R_i^+) along cycle ρ, which has size $s(n)$. The size of G^+ and G^- is bounded by $\frac{2n}{3} + s(n)$, while the size of R_i^- and R_i^+ is bounded by $3s(n)$. Therefore, since cycle ρ is shared by all the mentioned graphs by construction, we have

that the size of G_i^+ and G_i^- is at most $\frac{2n}{3} + 3s(n)$. Thus, with $s(n) \leq 2\sqrt{\ell n}$ [67], we can set the base case of Algorithm TestCP when $n \leq \frac{2n}{3} + 6\sqrt{\ell n}$, that is, $n \leq 324\ell$.

Correctness of the algorithm. We show that, given a 2-connected c-graph $\mathcal{C}(G, \mathcal{T})$, Algorithm TestCP returns `YES`, which happens when both procedures OuterCheck and InnerCheck succeed, if and only if $\mathcal{C}(G, \mathcal{T})$ is c-planar.

($\Rightarrow$) Suppose that OuterCheck and InnerCheck succeed for a cycle-star $S_\omega^- \in \mathcal{S}$ constructed at step 2a. We show that $\mathcal{C}(G, \mathcal{T})$ is c-planar. Consider the c-graph $\mathcal{C}^*(G^*, \mathcal{T})$ constructed at step 3(a)v from $\mathcal{C}_{con}^-(H_\omega^-, \mathcal{T}_\omega^-)$ and $\mathcal{C}_{con}^+(H_\omega^+, \mathcal{T}_\omega^+)$. The proof of this direction follows by the next claim about $\mathcal{C}^*$ and from Theorem 4.2.

Claim 1. *C-graph $\mathcal{C}^*(G^*, \mathcal{T})$ is a c-planar c-connected super c-graph of $\mathcal{C}(G, \mathcal{T})$.*

Proof. Graphs G_{in} and G_{out} are planar, as they are subgraphs of H_ω^- and H_ω^+, respectively (step 3(a)v). By construction, cycle ρ bounds an inner face of G_{out} and the outer face of G_{in}. Therefore G^*, obtained by merging G_{in} and G_{out}, is planar. Also, observe that, G_{in} and G_{out} are supergraphs of G^- and G^+, respectively, therefore graph G^* is a super graph of G.

We now show that $\mathcal{C}^*$ is c-connected, that is, for each cluster $\mu \in \mathcal{T}$, graph $G^*(\mu)$ is connected.

First, let μ be a cluster in $\mathcal{T}$ such that $V(\mu)$ lies in the interior of ρ in G. Since $\mathcal{C}_{con}^-(H_\omega^-, \mathcal{T}_\omega^-)$ is c-connected, we have that $H_\omega^-(\mu)$ is connected. Also, $V(\mu)$ lie in the interior of ρ in H_ω^-. By construction, G_{in} contains all the vertices and the edges in the interior of ρ, therefore we also have that $G_{in}(\mu)$ is connected. Hence, $G^*(\mu)$ is connected. The proof that graph $G^*(\mu)$ is connected, for each cluster μ in $\mathcal{T}$ such that $V(\mu)$ lies in the exterior of ρ in G, is analogous.

Then, let μ be a cluster such that $V(\mu) \cap V(\rho) \neq \emptyset$. Clearly, if $V(\mu) \subseteq V(\rho)$, then $G^*(\mu)$ is connected since both $G_{in}(\mu)$ and $G_{out}(\mu)$ are connected. Otherwise, the following three cases are possible: either $G_{in}(\mu)$ is disconnected, or $G_{out}(\mu)$ is disconnected, or both $G_{in}(\mu)$ and $G_{out}(\mu)$ are disconnected.

We show that all the vertices in $G_{in}(\mu)$ and in $G_{out}(\mu)$ are connected in $G^*(\mu)$.

We first prove that all the vertices in $G_{in}(\mu)$ are connected in $G^*(\mu)$.

Consider two connected components c' and c'' of $G_{in}(\mu)$. Observe that, by construction, c-graph $\mathcal{C}^-_{con}(H^-_\omega, \mathcal{T}^-_\omega)$ (step 3(a)v) is a merge of $\mathcal{C}_{in}(G_{in}, \mathcal{T}_{in})$ and of $\mathcal{C}^+_\Delta(R^+_\omega, \mathcal{J}^+_\omega)$. Since $\mathcal{C}^-_{con}$ is c-connected and since R^+_ω is an augmentation of cycle-star S^+_ω such that edges in $E(R^+_\omega) \setminus E(S^+_\omega)$ do not have endpoints in the same cluster, the c-graph $\mathcal{C}^\#(G^\#, \mathcal{T}^\#)$ obtained by merging $\mathcal{C}_{in}$ and $\mathcal{C}^+(S^+_\omega, \mathcal{K}^+_\omega)$ is also c-connected. Since $\mathcal{C}^\#$ is c-connected, the vertices of c' and c'' are connected via star vertices of S^+_ω and vertices of G_{in} belonging to cluster μ in $G^\#(\mu)$. Observe that, by construction, c-graph $\mathcal{C}^+_{con}(H^+_\omega, \mathcal{T}^+_\omega)$ is a merge of $\mathcal{C}_{out}(G_{out}, \mathcal{T}_{out})$ and of $\mathcal{C}^-_\Delta(R_\omega, \mathcal{J}_\omega)$. Further, S^+_ω has been obtained by applying the construction of Lemma 4.3 to c-graph $\mathcal{C}^+_{con}(H^+_\omega, \mathcal{T}^+_\omega)$ (step 3(a)i) and cycle ρ. Therefore, each connected component of μ in G_{out} corresponds to a star vertex of S^+_ω. Hence, we have that the vertices of c' and c'' are also connected in G^* via vertices of G_{out} and G_{in} belonging to cluster μ.

Now, we prove that all the vertices in $G_{out}(\mu)$ are connected in $G^*(\mu)$.

Consider two connected components c' and c'' of $G_{out}(\mu)$. Observe that, as shown above, each connected component of μ in G_{out} corresponds to a star vertex of S^+_ω. Recall that $\mathcal{C}^\#$ is c-connected. Therefore, the star vertices of S^+_ω corresponding to c' and c'' are connected via other star vertices of S^+_ω and vertices of G_{in} belonging to cluster μ in $G^\#(\mu)$. Hence, the vertices of c' and c'' are also connected in G^* via vertices of G_{out} belonging to connected components of μ corresponding to star vertices of S^+_ω and vertices of G_{in} belonging to cluster μ in $G^*(\mu)$. □

($\Leftarrow$) Suppose that $\mathcal{C}(G, \mathcal{T})$ is c-planar. We show that Procedure OuterCheck and InnerCheck succeed. Since $\mathcal{C}(G, \mathcal{T})$ is c-planar, there exists a super c-graph $\mathcal{C}^*(G^*, \mathcal{T})$ of $\mathcal{C}$ such that G^* is planar and $\mathcal{C}^*$ is c-connected, by Theorem 4.2. By using the construction of Lemma 4.3 on c-graph $\mathcal{C}^*$, we can obtain a cycle-star S^- whose universal cycle is ρ that represents the connectivity of clusters inside ρ in $\mathcal{C}^*$. The proof of this direction follows from the next claim.

Claim 2. *Procedures* OuterCheck *and* InnerCheck *succeed for* $S^-_i = S^-$.

Proof. Procedure OuterCheck succeeds if, for a cycle separator ρ of G selected at step 1 of the algorithm, there exists a cycle-star S^-_i whose universal cycle is ρ such that the corresponding

c-graph $\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$, constructed at steps 2(b)i, 2(b)ii, and 2(b)iii of the algorithm, is c-planar. Recall that, cycle-star S^- has the following properties: 1. Cycle ρ is the universal cycle of S^- and bounds the outer face of S^-, and 2. for each star vertex v of S^-, the neighbours of v belong to the same cluster $\mu \in \mathcal{K}^-$vertex v belongs to. Since, steps 2a and 2(b)i construct all c-graphs $\mathcal{C}^-(S_i^-, \mathcal{K}_i^-)$ with the above properties, when $S_i^- = S^-$ we are guaranteed to compute c-graph $\mathcal{C}^-(S^-, \mathcal{K}^-)$. First, observe that the c-graph obtained by merging c-graphs $\mathcal{C}^-(S^-, \mathcal{K}^-)$ and $\mathcal{C}_\rho^+(G^+, \mathcal{T}^+)$ is c-planar, since S^- has been obtained by applying the construction of Lemma 4.3 to a super c-connected c-planar c-graph $\mathcal{C}^*$ of $\mathcal{C}$. This together with Remark 1 imply that c-graph $\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$ is c-planar. Thus, the invocation of TestCP on $\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$ at step 2(b)iv will return `YES`. Hence, Procedure OuterCheck succeeds.

Procedure InnerCheck succeeds if, there exists a c-graph $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$, constructed at steps 3(a)i, 3(a)ii, and 3(a)iii of the algorithm, that is c-planar. By Theorem 4.2, a c-graph $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$ is c-planar if and only if there exists a super c-graph $\mathcal{C}'(G', \mathcal{T}_i^-)$ of $\mathcal{C}^-(G_i^-, \mathcal{T}_i^-)$ such that G' is planar and $\mathcal{C}'$ is c-connected. As Procedure OuterCheck succeeds, the c-graph $\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$ corresponding to S^- is c-planar. Therefore, Procedure OuterCheck provides us with a c-planar c-connected c-graph $\mathcal{C}_{con}^+(H_i^+, \mathcal{T}_i^+)$ (see steps 2(b)iv and 3(a)i) that is a super c-graph of $\mathcal{C}^+(G_i^+, \mathcal{T}_i^+)$. Consider the c-graph $\mathcal{C}^+(S_i^+, \mathcal{K}_i^+)$ constructed at step 3(a)i by applying the construction of Lemma 4.3 to $\mathcal{C}_{con}^+$. Observe that, the c-graph obtained by merging $\mathcal{C}^+(S_i^+, \mathcal{K}_i^+)$ and $\mathcal{C}_\Delta^-(R_i^-, \mathcal{J}_i^-)$ is a c-connected c-planar c-graph. This is due to the fact that, since R_i^- is internally triangulated, there exists no edge in the interior of ρ in H_i^+ that belongs to H_i^+ and does not belong to R_i^-, that is, no candidate saturating edges connect two vertices in the interior of ρ in $\mathcal{C}_{con}^+$. Since $S_i^+ \subseteq R_i^+$, we also have that the c-graph obtained by merging $\mathcal{C}_\Delta^+(R_i^+, \mathcal{J}_i^+)$ (constructed at step 3(a)ii) and $\mathcal{C}_\Delta^-(R_i^-, \mathcal{J}_i^-)$ is a c-connected c-planar c-graph. Also, since each of the vertices added to obtain R_i^- from S^- belongs to a different cluster and since the edges added to internally triangulate S^- do not connect vertices of the same cluster, we have that the c-graph obtained by merging $\mathcal{C}_\Delta^+(R_i^+, \mathcal{J}_i^+)$ and $\mathcal{C}^-(S^-, \mathcal{K}^-)$ is also a c-connected c-planar c-graph.

Let $\mathcal{A}$ be the subgraph of G^* induced by the edges in the interior and on the boundary of ρ in $\mathcal{C}^*$. Since S^- exactly represents the cluster connectivity of $\mathcal{A}$, the c-graph obtained by merging

$\mathcal{C}^+_\Delta(R^+_i, \mathcal{J}^+_i)$ and $\mathcal{A}$ is also a c-connected c-planar c-graph. The fact that, such a c-graph is a super c-graph of $\mathcal{C}^-(G^-_i, \mathcal{T}^-_i)$ shows that $\mathcal{C}^-(G^-_i, \mathcal{T}^-_i)$ is c-planar. Hence, Procedure InnerCheck succeeds. □

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

Theorem 4.3. *The* C-Planarity *problem can be solved in* $2^{O(\sqrt{\ell n}\cdot \log n)}$ *time for* n*-vertex c-graphs with maximum face size* ℓ*.*

Proof. Given an n-vertex c-graph $\mathcal{C}(G, \mathcal{T})$ with maximum face size ℓ, by Lemma 4.2, we can construct in linear time a 2-connected, in fact 3-connected, c-graph $\mathcal{C}'$ equivalent to $\mathcal{C}$. Therefore, we can apply Algorithm TestCP to $\mathcal{C}'$ to determine whether $\mathcal{C}$ is c-planar.

We now argue about the running time.

Since G' is 2-connected and since, by Lemma 4.2, $|V(G')| = O(|V(G)|)$ and the maximum face size ℓ' of G' is $O(\ell)$, we can construct a cycle separator ρ of G of size $s(n) = O(\sqrt{\ell n})$ that separates the vertices in the interior of ρ from the vertices in the exterior of ρ in such a way that both such sets contain at most $\frac{2n}{3}$ vertices [67]. Also, since all cycle-stars whose universal cycle is ρ have size $O(s(n))$ and the augmentations at steps 2(b)ii and 3(a)ii can be done by introducing at most $s(n)$ new vertices, graphs G^+_i (step 2(b)iv) and G^-_i (step 3(a)iv) have $O(\frac{2n}{3} + O(s(n)))$ size. Further, by construction, G^-_i and G^+_i are 2-connected and their maximum face size is ℓ'; thus, the cycle separators of G^-_i and G^+_i have size bounded by $s(|V(G^-_i)|)$ and by $s(|V(G^+_i)|)$, respectively.

Moreover, observe that each cycle-star $S^-_i \in \mathcal{S}$ satisfying the properties described at step 2a can be constructed in $O(s(n))$ time. Also, each cycle-star S^-_i is in one-to-one correspondence with a non-crossing partition of a set containing $s(n)$ elements. This is due to the fact that each vertex of ρ is incident to at most a star vertex of S^-_i and that, by the planarity of S^-_i, the neighbours of any two star vertices do not alternate along ρ. The number of all such partitions is expressed by the Catalan number $C_{s(n)} \leq 4^{s(n)}$.

The non-recursive running time $f(n)$ is bounded by the time taken by steps 1 and 3(a)i, that is, $O(n)$

time. In fact, the cycle-separator of an n-vertex graph can constructed in $O(n)$ time [67]. Testing whether a cycle is a cluster-separator can be done by performing a visit of the graph to detect if there exist a cluster whose vertices lie inside and outside of ρ, but not along ρ; this can clearly be done in $O(n)$ time. Finally, applying the construction of Lemma 4.3 to obtain a cycle-star only requires finding the connected components of each cluster inside (or outside) ρ and their respective connections to cycle ρ, which can be done in $O(n)$ time by performing a DFS-visit of G^- (or G^+).

By the above arguments, the running time of Algorithm TestCP is expressed by by the following recurrence:

$$T(n) = 2C_{s(n)}\Big(T\big(\frac{2n}{3} + O\big(s(n)\big)\big) + f(n)\Big) \tag{4.1}$$

Since equation (4.1) solves to $T(n) = 2^{O(\sqrt{\ell n}\cdot \log n)}$ for $s(n) = O(\sqrt{\ell n})$, $C_{s(n)} = 4^{s(n)}$, $f(n) = O(n)$, the statement follows. □

In the next section, we show how to adapt algorithm TestCP to obtain an XP algorithm with parameter h for generalized h-simply nested graphs, which extend simply-nested graphs with bounded face size.

4.3.1 Generalized h-Simply-Nested Graphs

A plane graph is *h-simply-nested* if it consists of nested cycles of size at most h and of edges only connecting vertices of the same cycle or vertices of adjacent cycles; refer to Fig. 4.6. We extend the class of h-simply-nested graphs to the class of *generalized h-simply-nested graphs*, by allowing the inner-most cycle to contain a plane graph consisting of at most $2h$ vertices in its interior and the outer-most cycle to contain a plane graph consisting of at most $2h$ vertices in its exterior.

See [23] for a related graph class, in which the vertices in the interior of the inner-most cycle can

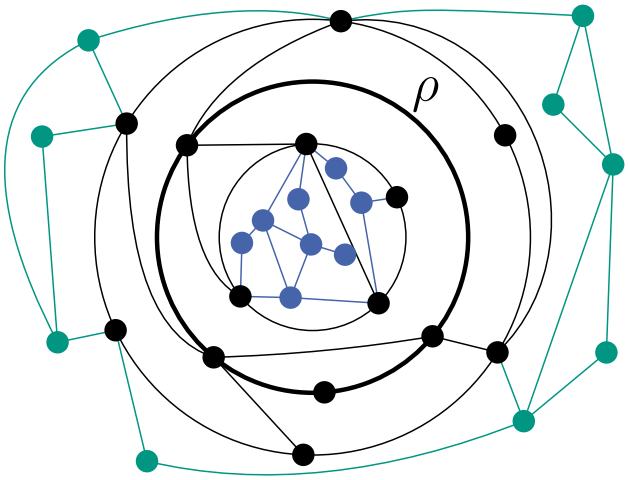


Figure 4.6: A generalized 6-simply-nested graph.

only form a tree, there exist no other vertices in the exterior of the outer-most cycle, and chords are not allowed for the remaining cycles.

Let G be a generalized h-simply-nested plane graph with $n > 5h$ vertices. We have the following simple observation about the structure of G; refer to Fig. 4.6.

Observation 2. *Graph G contains a cycle ρ with $|V(\rho)| \leq h$ that separates G into two generalized h-simply-nested graphs G^+ and G^- with $|V(G^+)| \leq \frac{n}{2}$ and $|V(G^-)| \leq \frac{n}{2}$ such that G^+ (G^-) does not contain any vertex in the exterior (interior) of its outer-most cycle (inner-most cycle). Further, such a cycle can be computed in $O(n)$ time.*

By Observation 2, we can use a cycle separator of size at most h in Algorithm TestCP to test the c-planarity of a c-graph whose underlying graph is a generalized h-simply-nested plane graph G (instead of a cycle separator of size $O(\sqrt{\ell n})$, where ℓ is the maximum face size of G). Observe that, graphs G_i^+ and G_i^- obtained at steps 2(b)iii and 2(b)iii of the algorithm, respectively, also belong to the family of generalized h-simply-nested plane graphs. Therefore, Observation 2 also holds for such graphs. Altogether, we obtain the following recurrence relation for the running-time:

$$T(n) = 2C_h\Big(T\Big(\frac{n}{2} + O(h)\Big) + O(n)\Big) \tag{4.2}$$

Equation (4.2) immediately implies the following theorem.

Theorem 4.4. *The* C-PLANARITY *problem can be solved in $n^{O(h)}$ time for n-vertex c-graphs whose underlying graph is a generalized h-simply-nested graph.*

4.4 An MSO_2 formulation for C-Planarity

In this section, we show that the property of a c-graph of admitting a c-planar drawing can be expressed in extended monadic second-order (MSO_2) logic. We use this result and the fact that graph properties definable in MSO_2 logic can be verified in linear time on graphs of bounded treewidth, by Courcelle's Theorem [26], to build an FPT algorithm for testing the c-planarity of embedded flat c-graphs.

First-order graph logic deals with formulae whose variables represent the vertices and edges of a graph. *Second-order graph logic* also allows quantification over k-ary relations defined on the vertices and edges. *MSO_2 logic* only allows quantification over elements and *unary relations*, that is, sets of vertices and edges. Given a graph G and an MSO_2 formula ϕ, we say that G *models* ϕ, denoted by $G \models \phi$, if the logic statement expressed by ϕ is satisfied by the vertices, edges, and sets of vertices and edges in G. We will apply Courcelle's theorem not to the underlying graph G of the clustered planarity instance, but to the supergraph $G^\diamond$ of G that includes all the candidate saturating edges of G. This will allow us to quantify over sets of candidate saturating edges, but in exchange we must show that $G^\diamond$, and not just G, has low treewidth (Lemma 4.2).

Let H be a graph and let $E_1, E_2 \subseteq E(H)$. The following logic predicates can be expressed in MSO_2 logic (refer, e.g., to [12, 27] for their detailed formulation):

- $\diamond$ $\textsc{planar}_H(E_1, E_2)$:= the subgraph $(V(H), E_1 \cup E_2)$ of H is planar, and
- $\diamond$ $\textsc{conn}_H(U, E_1, E_2)$:= vertices in $U \subseteq V(H)$ are connected by edges in $E_1 \cup E_2$.

Let $\mathcal{C}(G, \mathcal{T})$ be a c-graph and let E^* be the set of all the candidate saturating edges of $\mathcal{C}$. By Property(iii) of Theorem 4.2, c-graph $\mathcal{C}$ admits a c-planar drawing if and only if there exists a super c-graph $\mathcal{C}'(G', \mathcal{T})$ of $\mathcal{C}$ such that G' is planar and $\mathcal{C}'$ is c-connected. Testing Property(iii) amounts to determining the existence of a set $E^+ \subseteq E^*$ such that (i) the subgraph G' of $G^\diamond$ obtained by adding the edges in E^+ to G is planar and (ii) graph $G'(\mu)$ is connected, for each cluster $\mu \in \mathcal{T}$.

We remark that in an MSO_2 formula it is possible to refer to given subsets of vertices or edges of a graph, provided that the elements of such subsets can be distinguished from the elements of other subsets by equipping them with labels from a constant finite set [7]. Therefore, in our formulae we use the unquantified variables V_i to denote the set of vertices belonging to cluster μ_i, for each disconnected cluster $\mu_i \in \mathcal{T}$, E^* to denote the set consisting of all the candidate saturating edges of $\mathcal{C}$, and E_G to denote $E(G)$.

Let c be the number of disconnected clusters in $\mathcal{T}$. We have the formula:

$$\textsc{c-planar}_{\mathcal{C}(G,\mathcal{T})} \equiv \exists (E^+ \subseteq E^*) \Big[\textsc{planar}_{G^\diamond}(E_G, E^+) \wedge \bigwedge_{i=1}^{c} \textsc{conn}_{G^\diamond}(V_i, E_G, E^+) \Big]$$

It is easy to see that formula $\textsc{c-planar}_{\mathcal{C}(G,\mathcal{T})}$ correctly expresses Condition(iii) of Theorem 4.2 only if G admits a unique combinatorial embedding (up to a flip). In fact, if G has more than one embedding, formula $\textsc{c-planar}_{\mathcal{C}(G,\mathcal{T})}$ might still be satisfiable after a change of the embedding, as formula $\textsc{planar}_{G^\diamond}(E_G, E^+)$ models the planarity of an abstract graph rather than the planarity of a combinatorial embedding. We formalize this fact in the following lemma.

Lemma 4.4. *Let $\mathcal{C}(G, \mathcal{T})$ be a c-graph such that G has a unique combinatorial embedding and let $\mathcal{C}^\diamond(G^\diamond, \mathcal{T}^\diamond)$ be the c-graph obtained by augmenting $\mathcal{C}$ with all its candidate saturating edges. Then, $\mathcal{C}$ is c-planar iff $G^\diamond \models \textsc{c-planar}_{\mathcal{C}(G,\mathcal{T})}$.*

Since changes of embedding are not allowed in our context, as we aim at testing the c-planarity of a c-graph given a prescribed embedding, we combine Lemmata 4.2 and 4.4, and then invoke Courcelle's Theorem to obtain the following main result.

Theorem 4.5. *The* C-Planarity *problem can be solved in* $f(\overline{\text{emw}}, c)O(n)$ *time for* n*-vertex* c*-graphs with* c *disconnected clusters and whose underlying graph has embedded-width* $\overline{\text{emw}}$*, where* f *is a computable function.*

Proof. To test that $\mathcal{C}(G, \mathcal{T})$ admits a c-planar drawing with the given embedding we proceed as follows. First, we apply Lemma 4.2 to obtain a c-graph $\mathcal{C}^*(G^*, \mathcal{T}^*)$ that is equivalent to $\mathcal{C}(G, \mathcal{T})$ such that G^* is 3-connected. Note that, the 3-connectivity of G^* implies that it has a unique combinatorial embedding (up to a flip) [80]. Then, we construct formula $\phi = \text{C-PLANAR}_{\mathcal{C}^*(G^*, \mathcal{T}^*)}$ and the super c-graph $\mathcal{C}^\diamond(G^\diamond, \mathcal{T}^\diamond)$ of $\mathcal{C}^*$ obtained by augmenting $\mathcal{C}^*$ with all its candidate saturating edges. Finally, we use Courcelle's Theorem to test whether $G^\diamond \models \phi$. The correctness immediately follows from Lemmata 4.2 and 4.4.

We now argue about the running time. By Lemma 4.2, c-graph $\mathcal{C}^*(G^*, \mathcal{T}^*)$ can be constructed in $O(n)$ time. Let κ be the maximum face size of G^*. The number of candidate saturating edges of $\mathcal{C}^*$ is $O(\kappa^2 n)$. By Lemma 4.2, $\kappa = O(\ell)$. Hence, we can augment $\mathcal{C}^*(G^*, \mathcal{T}^*)$ to obtain $\mathcal{C}^\diamond(G^\diamond, \mathcal{T}^\diamond)$ in $O(\ell^2 n)$ time.

By Courcelle's theorem [26], it is possible to verify whether $G^\diamond \models \phi$ in $g(tw(G^\diamond), len(\phi))O(|V(G^\diamond)| + |E(G^\diamond)|)$ time, where g is a computable function. By Lemma 4.2, $|V(G^\diamond)| = |V(G^*)| = O(n)$ and $\text{tw}(G^\diamond) = \overline{\text{emw}}(G)$. Also, by the discussion above, $|E(G^\diamond)| = O(\ell^2 n)$ and, by definition of embedded-width, $\ell = O(\overline{\text{emw}})$; thus, $|E(G^\diamond)| = O(\overline{\text{emw}}^2 n)$. Further, formula ϕ can be constructed in time proportional to its length $len(\phi)$, which is $O(c)$. Therefore, the overall running time can be expressed as $f(\overline{\text{emw}}, c)O(n)$, where f is a computable function. □

4.5 Conclusions and Open Problems

In this chapter, we provide subexponential-time, XP, and FPT algorithms to test C-Planarity of fairly-broad classes of c-graphs.

Several interesting questions arise from this research: (1) Can our results be generalized from flat to non-flat c-graphs? (2) Is there a fully polynomial-time algorithm to test C-Planarity of c-graphs whose underlying graph is a generalized h-simply-nested graph? (3) Are there interesting parameters of the underlying graph such that C-Planarity is FPT with respect to a single one of them (e.g., outerplanarity index, maximum face size, notable graph width parameters)? (4) Are there interesting parameters of the c-graph such that C-Planarity is FPT with respect to a single one of them (e.g., number of clusters, number of vertices of the same cluster incident to the same face[3], maximum distance between two faces containing vertices of the same cluster)?

[3]This question has also been previously asked by Chimani *et al.* [19]

Bibliography

[1] M. Ajtai, V. Chvátal, M. Newborn, and E. Szemerédi. Crossing-free subgraphs. In *Theory and Practice of Combinatorics*, volume 60 of *North-Holland Mathematics Studies*, pages 9–12. North-Holland, 1982.

[2] H. A. Akitaya, R. Fulek, and C. D. Tóth. Recognizing weak embeddings of graphs. In A. Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 274–292. SIAM, 2018.

[3] P. Angelini and G. Da Lozzo. Clustered planarity with pipes. In S. Hong, editor, *27th International Symposium on Algorithms and Computation, ISAAC 2016*, volume 64 of *LIPIcs*, pages 13:1–13:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[4] P. Angelini, G. Da Lozzo, G. Di Battista, and F. Frati. Strip planarity testing for embedded planar graphs. *Algorithmica*, 77(4):1022–1059, 2017.

[5] P. Angelini, G. Da Lozzo, G. Di Battista, F. Frati, M. Patrignani, and V. Roselli. Relaxing the constraints of clustered planarity. *Comput. Geom.*, 48(2):42–75, 2015.

[6] P. Angelini, F. Frati, and M. Patrignani. Splitting clusters to get c-planarity. In D. Eppstein and E. R. Gansner, editors, *Graph Drawing, 17th International Symposium, GD 2009, Revised Papers*, volume 5849 of *LNCS*, pages 57–68. Springer, 2009.

[7] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[8] V. Arvind, J. Köbler, S. Kuhnert, and Y. Vasudev. Approximate graph isomorphism. In B. Rovan, V. Sassone, and P. Widmayer, editors, *37th Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 100–111, 2012.

[9] J. C. Athenstädt and S. Cornelsen. Planarity of overlapping clusterings including unions of two partitions. *J. Graph Algorithms Appl.*, 21(6):1057–1089, 2017.

[10] L. Babai. Graph isomorphism in quasipolynomial time. *arXiv preprint arXiv:1512.03547*, 2015.

[11] M. J. Bannister, S. Cabello, and D. Eppstein. Parameterized complexity of 1-planarity. In *13th Int. Symp. Algorithms and Data Structures*, volume 8037 of *Lect. Notes in Comput. Sci.*, pages 97–108. Springer, 2013.

[12] M. J. Bannister and D. Eppstein. Crossing minimization for 1-page and 2-page drawings of graphs with bounded treewidth. In C. A. Duncan and A. Symvonis, editors, *GD 2014*, volume 8871 of *LNCS*, pages 210–221. Springer, 2014.

[13] T. Biedl. Drawing planar partitions III: Two Constrained Embedding Problems. Tech. Report RRR 13-98, Rutcor Research Report, 1998.

[14] T. Bläsius and I. Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. *Theor. Comput. Sci.*, 609:306–315, 2016.

[15] J.-D. Boissonnat and F. P. Preparata. Robust plane sweep for intersecting segments. *SIAM J. Comput.*, 29(5):1401–1421, 2000.

[16] G. Borradaile, J. Erickson, H. Le, and R. Weber. Embedded-width: A variant of treewidth for plane graphs, 2017.

[17] V. Bouchitté, F. Mazoit, and I. Todinca. Treewidth of planar graphs: connections with duality. *Electronic Notes in Discrete Mathematics*, 10:34–38, 2001.

[18] U. S. Census Bureau. Tiger/line shapefiles and tiger/line files. `https://www.census.gov/geo/maps-data/data/tiger-line.html`.

[19] M. Chimani, G. Di Battista, F. Frati, and K. Klein. Advances on testing c-planarity of embedded flat clustered graphs. In C. A. Duncan and A. Symvonis, editors, *Graph Drawing - 22nd International Symposium, GD 2014, Revised Selected Papers*, volume 8871 of *LNCS*, pages 416–427. Springer, 2014.

[20] M. Chimani, C. Gutwenger, M. Jansen, K. Klein, and P. Mutzel. Computing maximum c-planar subgraphs. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing, 16th International Symposium, GD 2008, Revised Papers*, volume 5417 of *LNCS*, pages 114–120. Springer, 2008.

[21] M. Chimani and K. Klein. Shrinking the search space for clustered planarity. In W. Didimo and M. Patrignani, editors, *Graph Drawing - 20th International Symposium, GD 2012, Revised Selected Papers*, volume 7704 of *LNCS*, pages 90–101. Springer, 2012.

[22] F. R. K. Chung. Separator theorems and their applications. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows, and VLSI-Layout*, volume 9 of *Algorithms and Combinatorics*, pages 17–34. Springer-Verlag, 1990.

[23] R. J. Cimikowski. Finding hamiltonian cycles in certain planar graphs. *Inf. Process. Lett.*, 35(5):249–254, 1990.

[24] S. Cornelsen and D. Wagner. Completely connected clustered graphs. *J. Discrete Algorithms*, 4(2):313–323, 2006.

[25] P. F. Cortese, G. Di Battista, F. Frati, M. Patrignani, and M. Pizzonia. C-planarity of c-connected clustered graphs. *J. Graph Algorithms Appl.*, 12(2):225–262, 2008.

[26] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[27] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[28] G. Da Lozzo, D. Eppstein, M. T. Goodrich, and S. Gupta. Subexponential-time and FPT algorithms for embedded flat clustered planarity. In *WG '18*, LNCS, 2018. To Appear.

[29] E. Dahlhaus. A linear time algorithm to recognize clustered graphs and its parallelization. In C. L. Lucchesi and A. V. Moura, editors, *LATIN '98: Theoretical Informatics, Third Latin American Symposium, Proceedings*, volume 1380 of *LNCS*, pages 239–248. Springer, 1998.

[30] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, November 2005.

[31] G. Di Battista and F. Frati. Efficient c-planarity testing for embedded flat clustered graphs with small faces. *J. Graph Algorithms Appl.*, 13(3):349–378, 2009.

[32] W. Didimo, F. Giordano, and G. Liotta. Overlapping cluster planarity. *J. Graph Algorithms Appl.*, 12(3):267–291, 2008.

[33] V. Dujmović, D. Eppstein, and D. R. Wood. Genus, treewidth, and local crossing number. In *Proc. 23rd Int. Symp. Graph Drawing and Network Visualization*, volume 9411 of *Lect. Notes in Comput. Sci.*, pages 87–98. Springer, 2015.

[34] Z. Dvořák and S. Norin. Strongly sublinear separators and polynomial expansion. Electronic preprint arxiv:1504.04821, 2015.

[35] D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *ACM Conf. on Geographic Information Systems (GIS)*, pages 16:1–16:10, 2008.

[36] D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf. Approximate topological matching of quad meshes. *The Visual Computer*, 25(8):771–783, 2009.

[37] D. Eppstein, M. T. Goodrich, and D. Strash. Linear-time algorithms for geometric graphs with sublinearly many crossings. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 150–159. Society for Industrial and Applied Mathematics, 2009.

[38] D. Eppstein, M. T. Goodrich, and L. Trott. Going off-road: transversal complexity in road networks. In *ACM Conf. on Geographic Information Systems (GIS)*, pages 23–32, 2009.

[39] D. Eppstein and S. Gupta. Crossing patterns in nonplanar road networks. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, pages 40:1–40:9, 2017.

[40] P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Math. Hung.*, 17(1–2):61–99, 1966.

[41] A. Fabri and S. Pion. CGAL: The Computational Geometry Algorithms Library. In *Proc. 17th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 538–539, 2009.

[42] Q. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In P. G. Spirakis, editor, *Algorithms - ESA '95, Third Annual European Symposium, Proceedings*, volume 979 of *LNCS*, pages 213–226. Springer, 1995.

[43] R. Fulek. C-planarity of embedded cyclic c-graphs. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing*, LNCS 9801, pages 94–106, 2016.

[44] R. Fulek, J. Kyncl, I. Malinovic, and D. Pálvölgyi. Clustered planarity testing revisited. *Electr. J. Comb.*, 22(4):P4.24, 2015.

[45] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[46] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[47] M. T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. System Sci.*, 51(3):374–389, 1995.

[48] M. T. Goodrich, S. Gupta, and M. R. Torres. A topological algorithm for determining how road networks evolve over time. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, Burlingame, California, USA, October 31 - November 3, 2016*, pages 31:1–31:10, 2016.

[49] M. T. Goodrich, G. S. Lueker, and J. Z. Sun. C-planarity of extrovert clustered graphs. In P. Healy and N. S. Nikolov, editors, *Graph Drawing, 13th International Symposium, GD 2005, Revised Papers*, volume 3843 of *LNCS*, pages 211–222. Springer, 2005.

[50] A. Grigoriev and H. L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007.

[51] M. Grohe. Isomorphism testing for embeddable graphs through definability. In *32nd ACM Symp. on Theory of Computing (STOC)*, pages 63–72, 2000.

[52] C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Advances in c-planarity testing of clustered graphs. In S. G. Kobourov and M. T. Goodrich, editors, *Graph Drawing, 10th International Symposium, GD 2002, Revised Papers*, volume 2528 of *LNCS*, pages 220–235. Springer, 2002.

[53] A. Hagberg, P. Swart, and D. Schult. Exploring network structure, dynamics, and function using NetworkX. In *Proc. 7th Python in Science Conf.*, pages 11–15, 2008.

[54] S.-H. Hong and H. Nagamochi. Simpler algorithms for testing two-page book embedding of partitioned graphs. *Theoretical Computer Science*, 2016.

[55] V. Jelínek, E. Jelínková, J. Kratochvíl, and B. Lidický. Clustered planarity: Embedded clustered graphs with two-component clusters. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing, 16th International Symposium, GD 2008, Revised Papers*, volume 5417 of *LNCS*, pages 121–132. Springer, 2008.

[56] E. Jelínková, J. Kára, J. Kratochvíl, M. Pergel, O. Suchý, and T. Vyskocil. Clustered planarity: Small clusters in cycles and eulerian graphs. *J. Graph Algorithms Appl.*, 13(3):379–422, 2009.

[57] A. Karduni, A. Kermanshah, and S. Derrible. A protocol to convert spatial polyline data to network formats and applications to world urban road networks. *Scientific data*, 3, 2016.

[58] K. Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proc. 50th IEEE Symp. on Foundations of Computer Science*, pages 639–648, 2009.

[59] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *Proc. 26th ACM Symposium on Theory of Computing*, pages 27–37, 1994.

[60] F. T. Leighton. *Complexity Issues in VLSI.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1983.

[61] T. Lengauer. Hierarchical planarity testing algorithms. *J. ACM*, 36(3):474–509, 1989.

[62] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[63] K. Liu, Y. Li, F. He, J. Xu, and Z. Ding. Effective map-matching on the most simplified road network. In *ACM Conf. on Geographic Information Systems (GIS)*, pages 609–612, 2012.

[64] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In *ACM Conf. on Geographic Information Systems (GIS)*, pages 352–361, 2009.

[65] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[66] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[67] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986.

[68] J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

[69] J. Pach, R. Radoičić, G. Tardos, and G. Tóth. Improving the crossing lemma by finding more crossings in sparse graphs. *Discrete Comput. Geom.*, 36(4):527–552, 2006.

[70] J. Pach and G. Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.

[71] G. Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abh. Math. Sem. Univ. Hamburg*, 29:107–117, 1965.

[72] B. Rosen and A. Saalfeld. Match criteria for automatic alignment. In *7th Symp. on Computer-Assisted Cartography (Auto-Carto)*, pages 1–20, 1985.

[73] A. Saalfeld. Conflation automated map compilation. *International Journal of Geographical Information System*, 2(3):217–228, 1988.

[74] L. Savary and K. Zeitouni. Automated linear geometric conflation for spatial data warehouse integration process. In *8th AGILE Conference on GIScience*, 2005.

[75] B. Shumaker and R. Sinnott. Astronomical computing: 1. computing under the open sky. 2. virtues of the haversine. *Sky and Telescope*, 68:158–159, 1984.

[76] S. Stahl. The embeddings of a graph – a survey. *Journal of Graph Theory*, 2(4):275–298, 1978.

[77] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.

[78] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical systems theory*, 10(1):99–127, 1976.

[79] A. Ventura, A. Rampini, and R. Schettini. Image registration by recognition of corresponding structures. *IEEE Trans. on Geoscience and Remote Sensing*, 28(3):305–314, 1990.

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

[81] D. Xiong. A three-stage computational approach to network matching. *Transportation Research Part C: Emerging Technologies*, 8(1):71–89, 2000.

[82] M. Zhang. *Methods and implementations of road-network matching*. PhD thesis, Technical University of Munich, 2009.

[83] Q. Zhang and I. Couloigner. Automatic road change detection and GIS updating from high spatial remotely-sensed imagery. *Geo-Spatial Information Science*, 7(2):89–95, 2004.