Algorithm to find which Circuits could be Faces given a Circuit Matrix in a Planar Graph

Abhinav Mishra
Indian Institute of
Information Technology
iim2014003@iiita.ac.in

Neelanjana Jaiswal
Indian Institute of
Information Technology
ism2014005@iiita.ac.in

Sannihith Kavala
Indian Institute of
Information Technology
ihm2014004@iiita.ac.in

***Abstract*—Planar graphs are frequently used to analyze morphological properties of networks occurring in complex systems. In these networks, an important task is to successfully extract the faces. We introduce a versatile algorithm to extract the faces of a planar graph given the circuit matrix of the graph. This algorithm has many applications in different research fields. For example With a focus on developmental biology and urban street patterning, we apply the algorithm to extracting cells from tissue graphs and areas of building density from urban street networks. We present the theory as well as other possible cases about how the algorithm works.**

***Index Terms*—Circuit Matrix, Planar Graphs, Faces**

I. Introduction

The concept of graphs is widely used in economic, social, and biological systems. It provides us with a tool to illustrate and analyze networks such as transportation networks, relationships between individuals, and biochemical pathways. Planar graphs have been extensively used to model systems ranging from urban street patterns to epithelial tissue formations and vein patterning of plant leaves. In this work, we present an algorithm that can be used to identify which circuits can be faces if we are given a circuit matrix. All of these examples can be formalized as finding faces in a planar embedding of a graph. To our knowledge, there are only few easily accessible algorithms to find faces in planar graphs, e.g., the planar face traversal of the Boost Graph Library First, we present the theoretical background of the algorithm and then explain the algorithm in detail. We then illustrate the performance of the algorithm on graphs of different circuit matrices and different sizes.

II. Motivation

III. Theoretical Background

A. planar Graphs

A planar graph is a graph which can be drawn in the plane without any edges crossing. Some pictures of a planar graph might have crossing edges, but its possible to redraw the picture to eliminate the crossings. For example, although the usual pictures of K_4 and Q_3 have crossing edges, its easy to redraw them so that no edges cross. However, if you fiddle around with drawings of $K_{3,3}$, it quickly becomes clear that its crossings cant be eliminated.

A planar graph $G = (V; E)$ with vertices V and edges E is a graph that can be drawn in a two-dimensional (2D) plane such that no edges cross. We refer to such a drawing as a planar embedding of the graph. In our algorithm, we assume that we know the $CircuitMatrix$ of the graph G. Areas that are enclosed by edges are called faces. This also includes the outer face, which is bounded by all edges at the border of the embedding.

B. Faces

A planar graph divides the plane into a set of regions, also called faces or regions. Each face is bounded by a closed walk (the boundary of the face). If the graph contains a spike sticking into the middle of a region, the boundary of that face will contain two copies of the same edge. Therefore, the boundary of a face is not always a cycle. The degree of the face is the length of its boundary. By convention, we also count the unbounded area outside the whole graph as one region.

The number of faces $\mid F \mid$ (including the outer face) in a planar graph can be calculated using Euler's formula:

$$\mid F \mid = 2 - \mid V \mid + \mid E \mid, \qquad (1)$$

where $\mid V \mid$ is the number of vertices in the graph and $\mid E \mid$ the number of edges.

C. Circuit Matrix

We consider a loopless graph $G = (V, E)$ which contains circuits. We enumerate the circuits of $G : C_1, ..., C_l$. The circuit matrix of G is an m matrix $B = (b_{ij})$ where

$$b_{ij} = \begin{cases} 1, & \text{if the arc} e_j \text{is in the circuit } C_i \\ 0, & \text{otherwise} \end{cases}$$

(as usual, $E = e_1, ..., e_m$). The circuits in the digraph G are oriented, i.e. every circuit is given an arbitrary direction for the sake of defining the circuit matrix. After choosing the orientations, the circuit matrix of G is $B = (b_{ij})$ where

$$b_{ij} = \begin{cases} 1, & \text{if arc } e_j \text{is in circuit } C_i \text{ and are in same direction} \\ -1, & \text{if arc } e_j \text{is in circuit } C_i \text{ and are in opposite direction} \\ 0, & \text{otherwise} \end{cases}$$

Algorithm 1 Find faces

```
1: for all v ∈ V do
2:     for all adjacent v_adj to v do initialize visit
3:         while v ∉ visit do
4:             if |visit| = 3 and counter-clockwise then
5:                 for all candidate vertices v_cand do
6:                     if |v_cand| = 1 then add v_cand to visit
7:                     else
8:                     calculate orientation o and angle α
9:
10:                    find v_cand leading to counterclockwise
11:                    orientation v_ccw
12:                        if |v_ccw| = 1 then add v_ccw to visit
13:                        else if |v_ccw| > 1 then add v_cand
14:                            with minimal α to visit
15:                        else if |v_ccw| = 0 then
16:                            if any v_cand is collinear then
17:                                add e_cand to visit
18:                            else
19:                                add v_cand with maximal α
20:                                to visit
```

IV. Algorithm

We are using Boost C++ library in order to get the planar embedding of the given graph inorder for the algorithm to get the relative orientation of the edges in the graph. A traversal of the faces of a planar graph involves iterating through all faces of the graph, and on each face, iterating through all vertices and edges of the face. The iteration through all vertices and edges of each face follows a path around the border of the face.

In a biconnected graph, like the one shown above, each face is bounded by a cycle and each edge belongs to exactly two faces. For this reason, when $planarfacetraversal$ is called on a biconnected graph, each edge will be visited exactly twice: once on each of two distinct faces, and no vertex will be visited more than once on a particular face. The output of $planarfacetraversal$ on non-biconnected graphs is less intuitive - for example, if the graph consists solely of a path of vertices (and therefore a single face), $planarfacetraversal$ will iterate around the path, visiting each edge twice and visiting some vertices more than once. $planarfacetraversal$ does not visit isolated vertices.

Like other graph traversal algorithms in the Boost Graph Library, the planar face traversal is a generic traversal that can be customized by the redefinition of certain visitor event points. By defining an appropriate visitor, this traversal can be used to enumerate the faces of a planar graph, triangulate a planar graph, or even construct a dual of a planar graph.

The general idea of our algorithm is to iterate over all vertices of the graph and enumerate all faces that contain this vertex. From each vertex, the edges are traversed in a certain orientation (i.e., clockwise or counter-clockwise) in order to and all faces that contain the initial vertex. To determine the edges we need to traverse, we compute the orientation and the angle between the current edge and the next candidate edge, as described next. Algorithm 1 shows the steps to find the faces in a planar graph.

V. Conclusion

We presented a novel algorithm for finding faces in a graph with known two-dimensional embedding. The algorithm is applicable to any such graph, as long as all faces are simple polygons and no vertices of degree less than two exist. We have also used Boost C++ library for plotting a 2 D embedding of the planar graph in order for our algorithm to work.