PLANARITY TESTING ALGORITHMS IN GRAPH THEORY

Master of Science Thesis

Mohamed Muhumed HASSAN

Eskişehir 2020

PLANARITY TESTING ALGORITHMS IN GRAPH THEORY

Mohamed Muhumed HASSAN

MASTER OF SCIENCE THESIS

Department of Mathematics
Supervisor: Prof. Dr. Handan AKYAR

Eskişehir
Anadolu University
Institute of Graduate Programs
July 2020

ABSTRACT

PLANARITY TESTING ALGORITHMS IN GRAPH THEORY

Mohamed Muhumed HASSAN

Department of Mathematics

Program in Applied Mathematics

Anadolu University, Institute of Graduate Programs, July 2020

Supervisor : Prof. Dr. Handan AKYAR

Determining the planarity or displanarity of a graph and possibly embedding that graph is one of the most intriguing and fascinating algorithmic problems of the fields of graph theory and graph drawings. This thesis studies the simplest ways to do exactly that by using simple nevertheless efficient algorithms. Section 4 gives a short but comprehensive historical background on the subject by introducing a list of almost all planarity algorithms of all time.

In this master thesis two linear-time algorithms and one quadratic-time algorithm are discussed. The presented algorithms are applied on two examples each to see which algorithm comes in handy when testing the planarity of a graph manually. According to the findings, the Boyer-Myrvold algorithm is the fastest algorithm to test the planarity of a certain graph, with few vertices, manually. However, based on tests done in LEDA (A Library of Efficient Data Types and Algorithms) on graphs with many vertices, the Left-Right planarity algorithm is the fastest algorithm to test planarity.

Keywords: Planarity, Testing planarity, planarity algorithms, Embedding planar graph

ÖZET

ÇİZGE KURAMINDA DÜZLEMSELLİĞİ TEST ETME ALGORİTMALARI

Mohamed Muhumed HASSAN
Matematik Anabilim Dalı
Uygulamalı Matematik Tezli Yüksek Lisans Programı
Anadolu Üniversitesi, Fen Bilimleri Enstitüsü, Temmuz 2020

Danışman : Prof. Dr. Handan AKYAR

Bir çizgenin düzlemsel olup olmadığını belirlemek ve düzlemsel bir çizgeyi gömmek, çizge kuramı ve çizge çizimleri alanlarının en ilgi çekici ve büyüleyici algoritmik sorunlarından biridir. Bu tezde, basit ancak verimli algoritmalar yardımıyla bu tür problemler çözmenin basit yöntemleri ele alınmaktadır. Tezin 4. bölümünde, literatürde yer alan ve dümleselliği test etmek için kullanılan algoritmaların neredeyse tamamı listelenerek konuyla ilgili kısa ancak kapsamlı bir tarihsel süreçte sunulmakadır.

Bu yüksek lisans tezinde iki doğrusal-zaman algoritması ve bir kuadratik-zaman algoritması tartışılmıştır. Sunulan algoritmalar bir çizgenin düzlemselliğini manuel olarak test ederken hangi algoritmanın daha kullanışlı olduğunu görmek için her algoritma iki örnek üzerinde de uygulanmıştır. Elde edilen bulgulara göre, Boyer-Myrvold algoritması, az sayıda köşeye sahip belirli bir çizgenin düzlemselliğini manuel olarak test etmek için en hızlı algoritmadır. Bununla birlikte, LEDA'da (Verimli Veri Tipleri ve Algoritmalar Kütüphanesi) çok sayıda köşe noktası olan çizgelerde yapılan testlere dayanarak, Sol-Sağ düzlemsellik algoritmasi en hızlı algoritmadır.

Anahtar Sözcükler: Düzlemsellik, Düzlemselliği test etme, Düzlemsel algoritmalar, Düzlemsel çizge gömme

ACKNOWLEDGMENT

First I would like to express my deep sense of gratitude and appreciation to my supervisor Prof. Dr. Handan AKYAR for her encouragement, expert advice and continuous support throughout this research. I would also like to extends my thanks to Prof. Dr. Emrah AKYAR without his guidance this would not have been possible. Besides My supervisors I would like to thank the members of my thesis committee Assistant Prof. Dr. İlknur ATASEVER GÜVENÇ and Assistant Prof. Dr. Özlem ESEN.

Furthermore, I would like to extend my appreciation and gratitude to my family members specially my parents whose love made many life changing opportunities possible for me. Without your support and guidance, I would not have been able to complete this research. So, to all of you, who helped me during this research, I would like to say thank you very much.

Mohamed Muhumed HASSAN

July 2020

27/07/2020

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Anadolu University, and that "it does not have any plagiarism "whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

.....

(Signature)

Mohamed Muhumed HASSAN

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Graph Theory commenced in the early 1700s when the Swiss Graph Theorist Leonhard Euler (1707—1783), one of the most prolific mathematicians of the 18th century, wrote an article about the famous long-standing 17th century *"Königsberg bridge problem"*. Many mathematicians believe that Euler's article on the seven bridges of Königsberg laid the foundations for graph theory.

In the beginning, the importance of Euler's theories on the subject and the usefulness of 'graph theory' as a whole was limited only to solving puzzles, recreations, and analyzing games. However, after almost a century since Euler proved that the Könisberg bridge problem has no solution, it dawned on mathematicians that graphs could serve as a mathematical models to solve and analyze many real-world problems efficiently and successfully. In our present-day, concepts in graph theory are used to solve certain problems in Chemistry, Physics, Engineering, Genetics, and many more. Also, certain branches of mathematics such as matrix theory, and topology have close connections with graph theory.

In the same year that graph theory was born (1736), the father of modern-day graph theory, Leonard Euler, found his essential formula which relates the size (number of edges), the order (number of vertices) and regions (faces) of polyhedrons. Since that time dozens of results have been found on planar graphs. These studies of planar and non-planar graphs have greatly contributed to the growth of graph theory. As a matter of fact, these studies have been instrumental to the development of topological, algebraic, and computational techniques in modern graph theory.

In this research, we present some basic results on planar graphs. In particular, we focus on algorithms to test and embed planar graphs and discuss several algorithms, three to be exact, in detail, compare them in terms of; efficiency, speed, time and space complexity, and simplicity of the algorithms in question. But, first let us grasp the main introductory concepts of graph theory by defining the basic terminology of the subject, which will serve as the chief aim of the first chapter which is to establish the necessary basis for understanding graph theory.

2 THE BASICS

The fundamental concepts of graphs theory are extraordinarily easy and can be used as a model for a wide variety of problems. The main purpose of this introductory chapter is to fully equip the reader with the terminology and notations that we use in this research so that the reader has complete mastery of the basic concepts and terminology of graph theory. At this point, we should mention that the terminology of graph theory is not standard, but the one used in this chapter is well adapted.

2.1 What is a graph?

A graph can be thought of as a diagram consisting of two sets; a nonempty finite set V of objects (which could be people, towns, numbers, or functions, etc.) called **vertex-set** together with a possibly empty set E of unordered pair (2-element) subsets of V called **edges-set.** Mathematically, we write $G = (V, E)$ to indicate that a graph has **edge set** E and **vertex set** V, each edge is an unordered pair of v vertices. The two points (vertices) associated with an edge are called its **end-vertices**. The two end-vertices of an edge are said to be **adjacent** i.e., they have an edge in common. Two edges are said to be **incident** if they have a common end-vertex. For simplicity, the convention to abbreviate the edge $\{a,\ b\}$ as just ab is adapted.

The number of edges in a graph is said to be its **size**, and the number of vertices is known as the **order** of G. For example, the size of the graph in Figure 2.1 is 8 and its order is 5. A **trivial graph** is a graph with order 1. An **empty graph** is a graph with no edges. i.e., size 0.

In the form of a picture, a graph can be represented by a diagram in which small circles represent vertices and line segments represent edges, where each line segment (edge) connects exactly two circles (vertices). For example, let us draw a graph $H = (V(H), E(H))$ with a vertex set $V(H) = \{a, b, c, x, y\}$ and edge set $E(H) = \{ab, ac, ax, ay, bc, by, cx, xy\}$ as shown in Figure 2.1.

Multiple (two or more) edges that connect the same distinct pair of vertices are said to be **parallel edges**. A **self-loop** is defined as an edge that joins a single

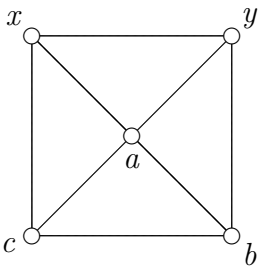


Figure 2.1: *A simple graph*

vertex to itself. A **simple graph** is a graph with neither loops nor parallel edges, and a graph with multiple edges and/or self-loops is called a **multigraph**. for instance, Figure 2.2 is a multigraph because it does not only contain parallel edges (between y and z) but also a self-loop (vertex x).

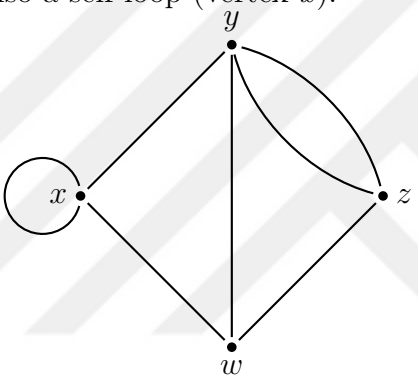


Figure 2.2: *A multigraph*

The **degree** of a vertex v, which is denoted as $deg(v)$, is the cardinality of the set of all vertices adjacent to v. For example, in Figure 2.2 the degrees of vertices w, z and y are 3, 3, and 4 respectively. At this point, we should mention that when calculating the degree of a vertex, it is standard to count **self-loops** twice, so $deg(x)$, in Figure 2.2, is 4. A graph G is called a **regular graph** if the degree of every vertex in G is r where r is any whole number. The graphs in Figure 2.3 are $0-regular$, $1-regular$, $2-regular$, and $3-regular$ respectively.

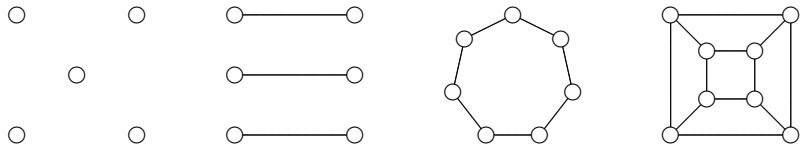


Figure 2.3: *A few regular graphs*

Theorem 2.1 (Handshaking lemma)**.** *The total sum of the degrees of all vertices of any graph H is twice the number of its edges (size). Mathematically,*

$$\sum_{v \in V(H)} deg(v) = 2m$$

where m represents the size of the graph H.

Proof. Each edge has two distinct vertices as its end-vertices, so each edge contributes 2 to the total sum of degree of the vertices of H. □

Corollary 2.1.1. *The cardinality of the set of all vertices of odd degree is even.*

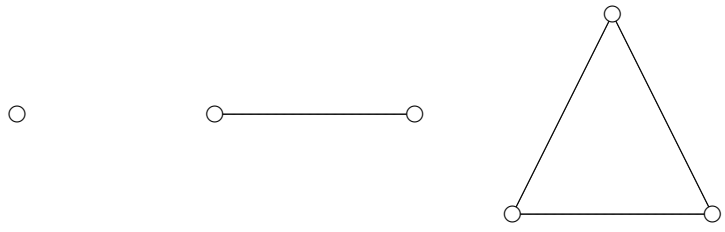
Proof. Since the sum of all degrees of any graph of size m is $2m$ which is, as anticipated, an even number. This implies that the total sum of degrees of all odd degree vertices is even, which guarantees that the number of odd degree vertices is an even number. □

2.2 Some Special Graphs

In this subsection we scrutinize some of the most common types of graphs that we encounter frequently in our research.

Definition 2.1. *A simple undirected graph is said to be a **complete graph** if every distinct pair of vertices are neighbors (adjacent).* A complete graph of order n is represented by K_n and it has $\binom{n}{2} = \frac{n(n-1)}{2}$ edges (see Figure 2.4).

Definition 2.2. *A simple graph is said to be a **bipartite graph** if the vertex-set of that graph can be divided into two non-empty disjoint subsets (called **partite sets**) U and V in such a way that each and every edge connects a vertex in U and a vertex in V.* Two simple bipartite graphs are shown in Figure 2.5.



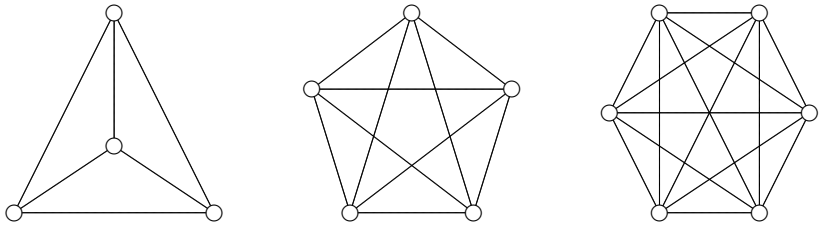


Figure 2.4: *The complete graphs (K_n) for $1 \leq n \leq 6$*

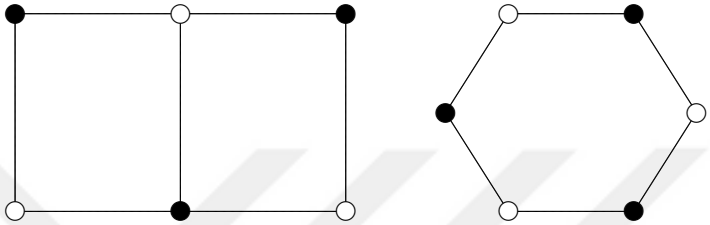


Figure 2.5: *Two bipartite Graphs*

If there is exactly one edge that joins every vertex in V to every vertex in U, then the graph is said to be a ***complete bipartite graph***, which is denoted as $K_{s,t}$ where s and t are the orders of U and V. The complete bipartite graph $K_{r,s}$ contains $r + s$ vertices and $r \cdot s$ edges. Two such graphs are shown in Figure 2.6.

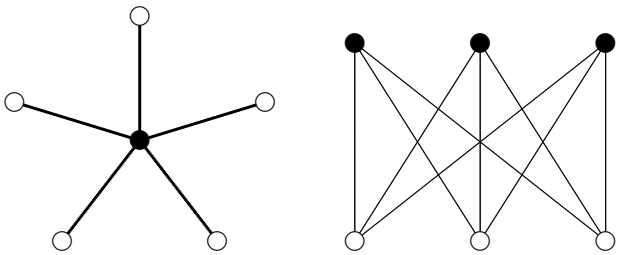


Figure 2.6: *$K_{1,5}$ and $K_{3,3}$*

Definition 2.3. A **walk** *(W)* in a graph H is a finite sequence of vertices in H denoted by $v_0 \longrightarrow v_1 \longrightarrow v_2 \cdots \longrightarrow v_{m-1} \longrightarrow v_m$ where the consecutive vertices are neighbors. A **path** (P_n) on n vertices is a walk from u to v (u and v are distinct) in which neither vertices nor edges are repeated. A **cycle** C_n is a closed path with the same initial and terminal vertices, i.e., $v_0 = v_m$. In other words, a connected

2-regular graph is called a **cycle.** Some path graphs and cycle graphs are shown in Figure 2.7.

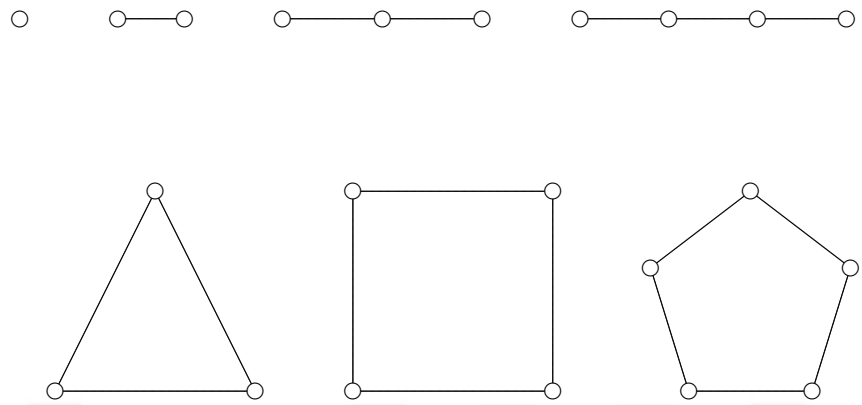


Figure 2.7: *Paths of order 4 or less and Cycles of order 5 or less*

Definition 2.4. *A **forest** (also known as an **acyclic graph**) is a graph that does not contain any cycles. A connected forest is called a **tree**.* A tree G on n vertices has size one less than its order. i.e., $m = n-1$ and contains exactly one path between every two vertices of G. An example of a tree is the simple alkane *isobutane* shown in Figure 2.8.

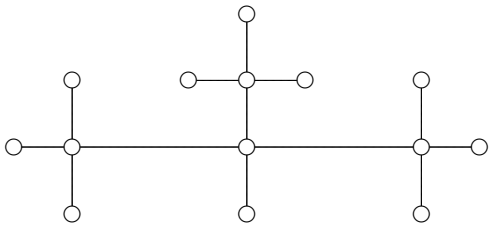


Figure 2.8: *A tree*

3 BASIC CONCEPTS OF PLANAR GRAPHS

In this chapter, we succinctly present the ABC of the heart of this research, i.e., planar graphs. In section 3.1 we concisely discuss one of the most important identities in graph theory, i.e. Euler's formula (Theorem 3.1). In the second section, we shortly explore how unplanar a graph is by examining its *crossing number*, *thickness* and *genus*. In section 3.3 we present one of the oldest results that characterizes planarity. In the last section of this introductory chapter, we study another way to characterize planar graphs, i.e., duality. But first, let us define what a planar graph is, and present few fundamental concepts of planarity.

A simple graph, usually denoted as G, is said to be planar (or embeddable) if it can be drawn on a surface of genus 0, in such a way that; each and every vertex is at a unique position on that surface, i.e., no two vertices overlap each other; every edge is a simple curve[1]; and no two edges intersect, except possibly at a common end-point [40, 28]. Simple graphs drawn in this particular way are said to be plane (or embedded) graphs. Both graphs in Figure 3.1 are planar (embeddable) but only the last one is a plane (embedded) graph.

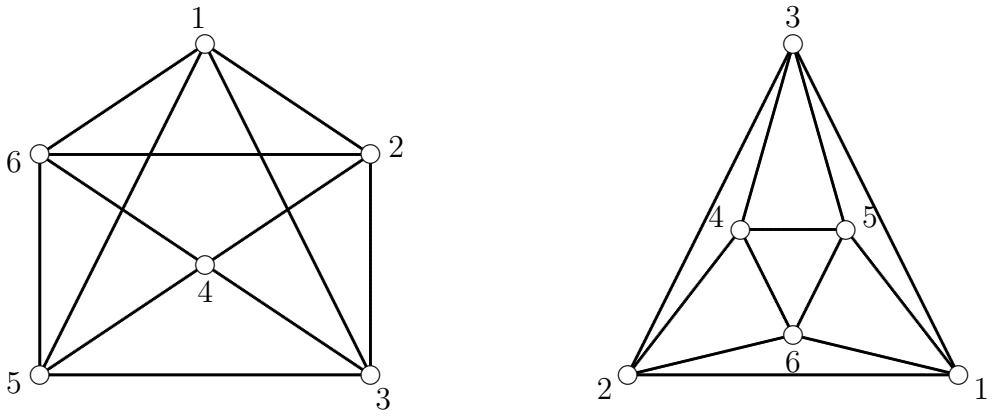


Figure 3.1: *(a) A planar graph and (b) its embedding*

The straight-line embedding of the graph in Figure 3.1 raises the question of whether we can embed every simple planar graph into a plane such that each and

[1]A continuous non-self-intersecting rectifiable (having finite length) curve.

every edge is represented by a straight-line segment. The answer to this question is yes and it is called **Fray's theorem** [12, 41].

3.1 Euler's Formula

If we embed a graph G into a plane, G divides the plane into different connected pieces called *faces* (*regions*). The unbounded, infinite, external face in every plane graph is called the *exterior face*. In Figure 3.2, for example, R_7 is unbounded and thus is the external face.

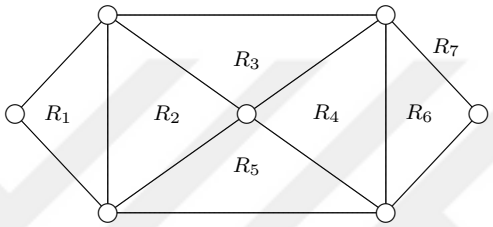


Figure 3.2: *A planar graph with 7 faces*

The eminent graph theorist Leonard Euler formulated an identity that relates the number of faces, edges, and vertices of a simple planar graph. Despite the fact that many indispensable books of Graph Theory including [43, 41, 18] call the equation $f = |E| - n + 2$ Euler's Formula, Chartrand *et al.* state firmly, in [12], that the expression, under consideration, is an identity rather than a formula.

Theorem 3.1 (**Euler's Formula**). *Let G be a simple, connected planar graph and let the order, size and the number of faces of G be n, $|E|$ and f respectively, then the following equation holds*

$$f + n = |E| + 2.$$

Proof. Euler's formula can be proved by induction on either the order of the graph, like in [43, 41, 12], or on the number of faces of G as in [4, 18].

We proceed by induction on f, the number of faces of G. If $f = 1$, then $|E| = n - 1$. G is a tree since it is connected and thus the theorem holds.

Now suppose that the formula is valid for all connected simple planar graphs with at most $f(\geq 2)$ faces. G is not a tree since it has more than one face and hence it contains a cycle. Let e be an edge that is on that cycle. Deleting e would result

$G' = G - e$. By induction hypothesis

$$f' + n' = |E|' + 2$$

using the relations between G and G'

$$|E|' = |E| - 1,\ n' = n,\quad f' = f - 1$$

we obtain

$$f + n = |E| + 2.$$

□

In Figure 3.2, for example, $n = 7$, $|E| = 12$, $f = 7$, and $f + n = |E| + 2 \Longrightarrow 7 + 7 = 12 + 2 \Longrightarrow 14 = 14$.

Before we proceed any further, let us present a few corollaries to Theorem 3.1.

Corollary 3.1.1. Let G be a simple connected planar graph with n (≥ 3) vertices, and $|E|$ edges.

$$|E| \leq 3n - 6.$$

Proof. If we combine Euler's identity with the fact that the degree of each face is at least 3, the corollary follows. □

Corollary 3.1.2. If G is a triangle-free (bipartite) simple connected planar graph, then

$$|E| \leq 2n - 4.$$

Proof. The process is similar to that of Corollary 3.1.1 except that $d(f) \geq 4$, where $d(f)$ is the degree of the face f which is the number of edges bounding that face. □

Corollary 3.1.3. Every simple planar graph contains a vertex with a degree of at most 5.

Proof. Substituting

$$n = \sum_i n(i)$$

$n(i)$ is the number of vertices of degree i and

$$2|E| = \sum_i i \cdot n(i)$$

into $|E| \leq 3n - 6$, produces the desired result. □

Corollary 3.1.4. All plane drawings of a simple connected planar graph have exactly the same number of faces.

Proof. Let G_1 and G_2 bet two plane drawings of a given simple connected planar graph. Since $G_1 \cong G_2$, $|E_1| = |E_2|$ and $n_1 = n_2$

$$\begin{aligned} f(G_1) &= |E_1| - n_1 + 2 \\ &= |E_2| - n_2 + 2 = f(G_2). \end{aligned}$$

□

Corollary 3.1.5. If G is a simple planar graph with k components, $|E|$ edges, n vertices and f faces, then

$$n - |E| + f = k + 1.$$

Proof. If we apply Euler identity separately to each component, keeping in mind not to overcount the unbounded face, the corollary follows. □

Corollary 3.1.6. Both $K_{3,3}$ and K_5 are nonplanar.

Proof. Assuming that K_5 is planar and applying Corollary 3.1.1 we have $9 \geq 10$ which is obviously impossible and therefore K_5 is nonplanar.
Similarly, suppose that $K_{3,3}$ is embeddable, applying Corollary 3.1.2 yields $9 \leq 8$ which is again a contradiction. Hence $K_{3,3}$ is nonplanar.

□

3.2 Measuring Displanarity

In this subsection, we consider parameters that measure how close a given graph is to planarity. These parameters include crossing number, thickness, and genus of a graph.

3.2.1 Crossing number

The smallest possible number of edge crossings that can occur in any simple drawing[2] of G in the plane is called *crossing number* and it is denoted $cr(G)$. The

[2]A simple drawing is characterized as: (1) No two edges cross more than once (2) Any two edges sharing a vertex do not intersect (3) A maximum of two edges can cross at any given point of the plane (4) There are no self-intersecting edges.

crossing number of a planar graph is zero and that of a nonplanar graph is at least one. Thus, $cr(K_{3,3}) \geq 1$ and $cr(K_5) \geq 1$. The graph in Figure 3.3 shows a simple drawing of k_6 where $cr(K_6) = 3$.

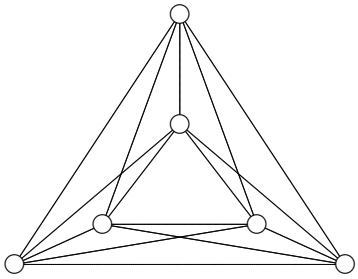


Figure 3.3: *A simple drawing of k_6 with 3 crossings*

Theorem 3.2. [21] *The number of edge crossings of K_4 in the plane, in any simple drawing, is either one or zero.*

Proof. Applying the properties of simple drawing on K_4 produces the required result. □

The *rectilinear crossing number* $\overline{cr}(G)$ of a simple graph G is the smallest number of edge crossings that can occur in a rectilinear[3] drawing of G. In general, crossing number and rectilinear crossing number are different in that $\overline{cr}(K_n) > cr(K_n)$ for $n > 7$ except for $n = 9$. For example, $\overline{cr}(K_8) = 19$ but $cr(K_8) = 18$. Generally, very little is known about the crossing number of graphs. However, there are many theorems that place a ceiling on crossing number and rectilinear crossing number of special classes of graphs. Table 3.1 shows the rectilinear crossing number [34] and crossing number [33] of K_n for $n \leq 12$. For more details on crossing number and rectilinear crossing number of graphs refer to [12, 21, 30, 38].

Table 3.1: *$cr(K_n)$ and $\overline{cr}(K_n)$ for $1 \leq n \leq 12$*

n	1	2	3	4	5	6	7	8	9	10	11	12
$cr(K_n)$	0	0	0	0	1	3	9	18	36	60	100	150
$\overline{cr}(K_n)$	0	0	0	0	1	3	9	19	36	62	102	153

[3]Rectilinear drawing is a drawing in which only straight lines are used

3.2.2 Thickness

The smallest possible number of subgraphs of G required to decompose G into planar subgraphs is called *thickness* of G, and it is denoted by *T(G)*. A graph G is planar if and only if it's thickness is one ($T(G) = 1$). If G is nonplanar, then $T(G) \geq 2$. For instance, $T(K_9) = 3$ as shown in Figure 3.4.

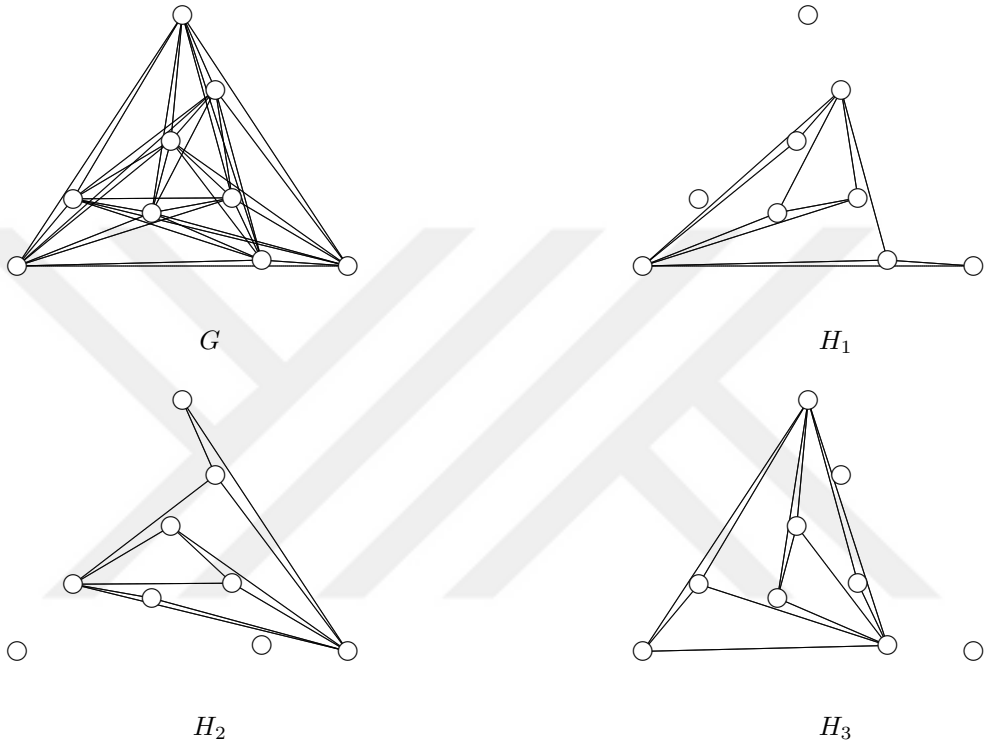


Figure 3.4: $K_9 = H_1 \cup H_2 \cup H_3$

Corollary 3.2.1. *For any simple graph with T thickness, n vertices and $|E|$ edges, the following holds:*

$$T(G) \geq \left\lceil \frac{|E|}{3n-6} \right\rceil .$$

Proof. This result easily follows form Corollary 3.1.1. □

Theorem 3.3. [2] *The thickness T of a complete graph K_n satisfies*

$$T(K_n) = \begin{cases} 1, & 1 \leq n \leq 4 \\ 2, & 5 \leq n \leq 8 \\ 3, & n = 9 \text{ or } n = 10 \\ \left\lfloor \frac{n+7}{6} \right\rfloor, & \text{otherwise.} \end{cases}$$

A similar result for the thickness of complete bipartite graphs was proposed in 1964 by Beineke, Harary and Moon, this result is summarized in Theorem 3.4.

Theorem 3.4. [3] For $s \leq t$, the thickness of $K_{s,t}$ is

$$\left\lceil \frac{st}{2(s+t-2)} \right\rceil$$

except possibly when st is odd and there exists an even integer r such that

$$t = \left\lceil \frac{r(s-2)}{s-r} \right\rceil.$$

Theorem 3.5. [20] *The thickness of* $K_{m,m}$ *is*

$$K_{m,m} = \left\lfloor \frac{m+5}{4} \right\rfloor.$$

A simple graph G is called *biplanar* if there are two planar subgraphs G_1 and G_2 such that $G = G_1 \cup G_2$. i.e. $T(G) = 2$. For example, observe from Theorem 3.3 that K_n for $5 \leq n \leq 8$ is biplanar.

3.2.3 Genus

We have seen that the two building blocks of nonplanar graphs (K_5 and $K_{3,3}$) cannot be planarly drawn on the plane (or sphere). However, both K_5 and $K_{3,3}$ can be planarly drawn on the surface of a torus, and thus they are called *toroidal graphs*. This is because torus has a different genus than sphere.

The genus is a property of a topological surface. A surface has a *genus* g if g non-intersecting Jordan Curves do not divide the surface into 2 regions, but $g+1$ non-intersecting Jordan Curves always divide the surface into two distinct regions. For example, every Jordan Curve drawn on a sphere guarantees a separation of the surface. However, one single Jordan Curve drawn on a torus can never cause a separation. Thus, the genus of spherical surfaces is 0, and that of the torus is 1.

A graph is said to be of *genus* g if it can be embedded on a surface of genus g, but can not be drawn planarly on a surface of genus $g-1$. Since the genus of a torus is one, the genus of every toroidal graph is also 1. Thus $g(K_5) = g(K_{3,3}) = 1$

Theorem 3.6. *The genus of a graph* G *is always less than or equal to its crossing number. i.e.* $g(G) \leq cr(G)$.

Proof. We draw G in such a way that the number of crossings is the smallest possible and hence equals to its $cr(G) = c$. Now, if we build a handle (bridge) at each crossing and draw one edge under it and one over it, we get c handles. Thus $g(G) \leq cr(G)$. □

Theorem 3.7. *Let G be a connected graph with genus g, $|E|$ edges, f faces, and n vertices, then:*

$$f = |E| - n + 2 - 2g.$$

Proof. We proceed by induction on the genus, g. For $g = 0$ the equation transforms to Euler's identity. Now, suppose that the theorem is valid for every graph G' with genus $g - 1$. and let's construct a graph G with genus g by adding one single edge which requires an additional handle. By the induction hypothesis we have

$$f' = m' - n' + 2 - 2g'$$

and

$$f' = f + 1,\ n' = n,\ m' = m - 1 \text{ and } g' = g - 1$$

and substituting these relations into the above equation of G' we obtain

$$f = |E| - n + 2 - 2g.$$

□

Corollary 3.7.1. *The genus g of a graph with m edges and $n(\geq 4)$ vertices satisfies:*

$$g \geq \left\lceil \frac{1}{6}(m - 3n) + 1 \right\rceil.$$

Proof. Using the fact that $3f \leq 2m$ and Theorem 3.7, we obtain the inequality for g. □

Theorem 3.8. *The genus of a complete graph K_n satisfies:*

$$g(K_n) = \left\lceil \frac{1}{12}(n - 3)(n - 4) \right\rceil.$$

For a general graph G, the genus $g(G)$ is greater than or equal to the quantity given in Theorem 3.8.

3.3 Kuratowski's Theorem

We devote this subsection to Kuratowski's Theorem and related characterizations of planarity. We have proved in Corollary 3.1.6 that both K_5 and $K_{3,3}$ are nonplanar. These two graphs are called *Kuratowski graphs*.

To present Kuratowski's theorem effectively we first need to define *homeomorphism*. Two graph G and H are said to be *homeomorphic* to each other if either G or H can be made isomorphic to the other by inserting or removing vertices of degree 2. The four graphs drawn in Figure 3.5 are homeomorphic to each other. Clearly every graph is, by definition, homeomorphic to itself.

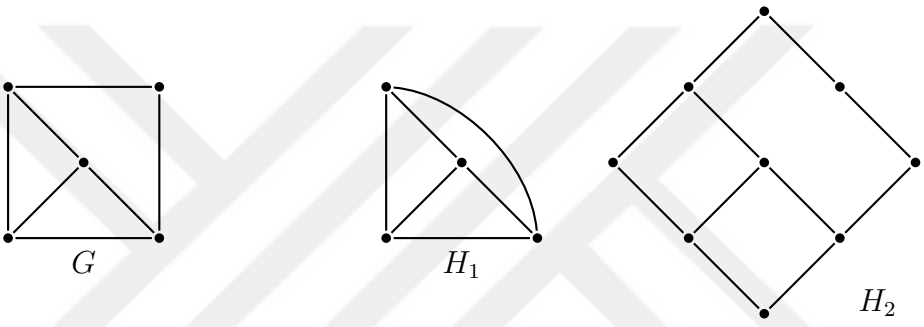


Figure 3.5: *H_1 and H_2 are homeomorphic to G*

The following observations are straightforward and easy to prove.

Corollary 3.8.1. *If a graph G is not planar, then every graph H that is homeomorphic to G is also nonplanar.*

Corollary 3.8.2. *If a simple graph G is planar, then every graph that is homeomorphic to G is also planar.*

The two graphs shown in Figure 3.6 are obviously homeomorphic to K_5 and $K_{3,3}$ respectively and hence by Corollary 3.8.1 are nonplanar.

Now, we briefly present Kuratowski's theorem. However, we omit the proof of Kuratowski's Theorem since it is long involved and beyond the scope of this research. For proof of this theorem see [4, 12, 18, 41].

Theorem 3.9 (**Kuratowski's Theorem**)**.** *A simple graph G is planar if and only if it does not contain any subgraph homeomorphic to $K_{3,3}$ or K_5.*

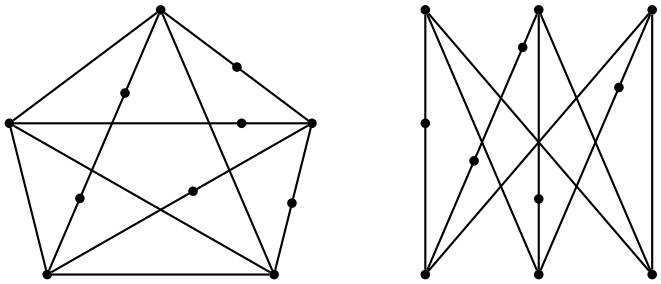


Figure 3.6: *Two graphs homeomorphic to K_5 and $K_{3,3}$*

Other than Kuratowski's theorem there are many other characterizations of planarity. For instance, Wagner's theorem [1937] characterizes planar graphs using contractible graphs. A graph G is said to be *contractible*[4] to another graph H, if H can be obtained from G by one or more edge contractions. If we want to contract edge $e = \{a, b\}$, for example, first we delete the edge e and then identify its end-vertices a and b such that the degree of the new vertex is $deg(a) + deg(b) - 2$. In Figure 3.7 contracting the 5 spooks that connect the outer and inner cycles of length 5 in the Petersen graph, we obtain K_5.

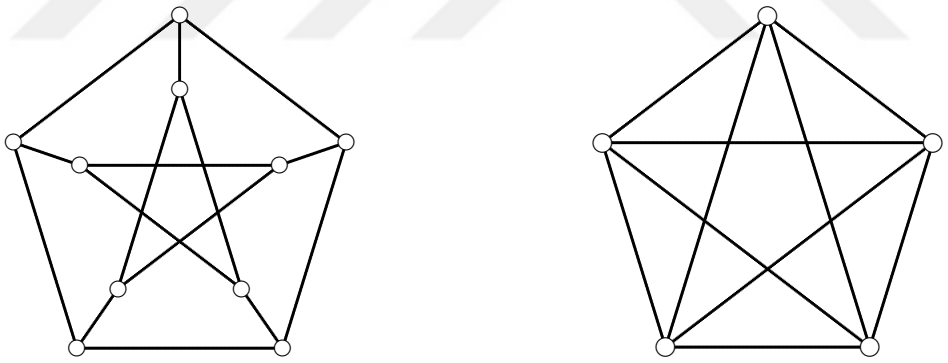


Figure 3.7: *Petersen graph is contractible to K_5*

Theorem 3.10 (**Wagner's Theorem**). *A simple graph G is planar if and only if it does not contain a subgraph contractible to $K_{3,3}$ or K_5.*

3.4 Duality

We conclude this chapter with one last characterization of planarity based on *duality*. Given a particular plane graph G, we construct its *geometric dual* G^* in

[4]Also known as **minor** of a graph

the following stages:

- Inside every face (region) f of G we put a vertex f^*. These are the vertices of the dual graph G^*.
- To each edge e, separating two faces f and h, of G we assign another edge e^* which joins the corresponding vertices f^* and h^* of G^* in such a way that e^* crosses exactly one edge which is the corresponding edge, i.e. e. The edges formed this way are the edges of G^*.
- If the edge e does not separate two regions of G, i.e., e is a bridge, then e^* is a self-loop incident with the vertex of G^* corresponding to that region in which the edge e lies.
- If two faces of G share more than one edge, then G^* has multiple edges.

The procedure of constructing the dual G^* of a graph G is illustrated in Figure 3.8. The dotted simple curves and the small square vertices represent G^*. If G_1^* and G_2^*

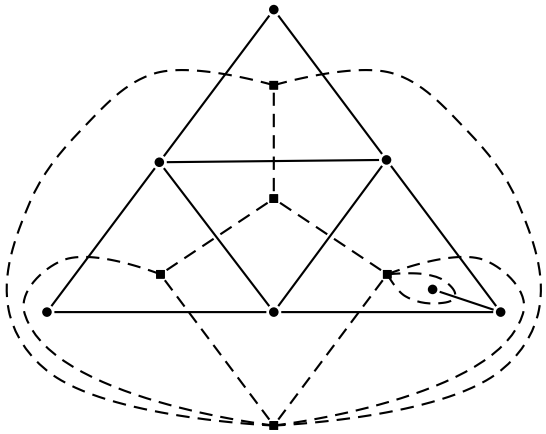


Figure 3.8: *A graph G and its dual G^**

are two graphs constructed from G using the construction rules for G^*, then G_1^* and G_2^* are isomorphic. However, if G and H are two isomorphic graphs, then G^* and H^* are not necessarily isomorphic.

Now, we present few results on dual graphs, but since duality does not contribute much to understanding and implementing the algorithms presented in

the next chapter, we omit the proofs of theorems and lemmas. A complete discussion on duality and proofs of theorems can be obtained in [4, 18, 43].

Lemma 3.1. *If G is connected embedded graph with $|E|$ edges, n vertices, and f faces, and G^* has $|E^*|$ edges, n^* vertices, and f^* faces. Then the following simple relations hold*

$$|E^*| = |E| \qquad n^* = f \qquad \text{and} \qquad f^* = n.$$

Proof. The first two relations are straightforward and last one can be proved with the help of Euler's formula and the fact that both G and G^* are planar.
By Euler's formula

$$f = m - n + 2 \qquad\qquad f^* = m^* - n^* + 2$$
$$f - m + n = 2 \qquad\qquad f^* - m^* + n^* = 2$$

which implies that

$$f^* - m^* + n^* = f - m + n.$$

Simplifying and using $m^* = m \qquad n^* = f$ we obtain

$$n = f^*.$$

□

Theorem 3.11. *If G is a connected plane graph the dual of the dual (double dual) G^{**} is isomorphic to G.*

Proof. Applying the construction guidelines for G^* and Lemma 3.1 we obtain the $G \cong G^{**}$. □

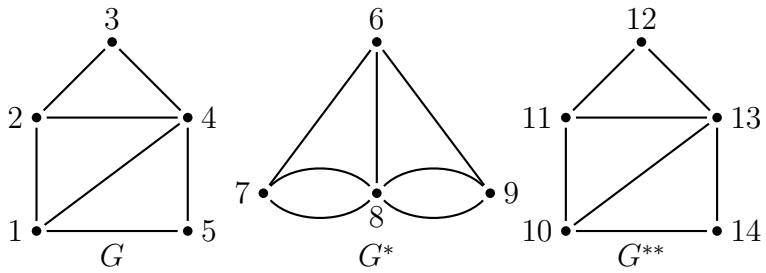


Figure 3.9: *G is isomorphic to G^{**}*

Any connected embaddable graph G is always isomorphic to the dual of its dual. The graph in Figure 3.9 is a case in point.

Theorem 3.12. *Every embeddable graph has a planar dual.*

Proof. If G is planar the construction guidelines for G^* guarantees the planarity of G^*. □

Theorem 3.13. *If G is an embeddable graph and if G^* is it's dual, then a group of edges in G forms a cut-set in G if and only if the corresponding group of edges in G^* forms a cycle in G^*.*

Lastly, we present the last and perhaps the most important theorem of this subsection.

Theorem 3.14. *A graph has a geometric dual if and only if it is planar.*

Proof. Using Theorem 3.13 together with Kuratowski's theorem (Theorem 3.10) delivers the desired result. □

4 PLANARITY TESTING AND EMBEDDING

Graph theorists and computer scientists have been trying to test and characterize planarity since the early 1900s when the famous Polish logician and mathematician **K. Kuratowski** published his well-known theorem (**Theorem 3.9**). It was not long after Kuratowski's theorem that many algorithms to test planarity were published. In 1970, P. Mei and N. Gibbs [27] developed an algorithm based on Kuratowski's theorem that runs in exponential time. Similarly, Wagner's characterisation of planar graphs yields a factorial-time algorithm. Evidently, the algorithms based on these two theorems (**Theorem 3.9** and **Theorem 3.10**) take relatively a long time to determine if a large graph is planar or not.

Due to the nature of the time-complexity [5] of the (then-available) algorithms, mathematicians felt obligated to spare no effort in trying to develop new polynomial-time algorithms. In 1961, the hard work of Auslander and Parter paid off, when they published the first polynomial-time planarity test algorithm [1]. Following in the footsteps of Auslander and Parter, Hopcroft and Tarjan (H *&* T) in 1974 proposed the first-ever linear-time algorithm that tests planarity [22]. Two years after Hopcroft and Tarjan published their algorithm, Booth and Lueker (B *&* L) followed their breakthrough and proposed the second linear-time planarity test algorithm [5].

The linear-time algorithms of H *&* T and B *&* L are so complex and hard to implement that many mathematicians strove to simplify these algorithms. For example, in 1993 Mehlhorn, Mutzel and Naher extended H *&* T's algorithm to produce an embedment of the graph, under consideration, rather than just determining if it is planar or not. Similarly, in 1985, the same was done for B *&* L's algorithm by Chiba *et al.* [13].

According to Ulrik Brandes [10] and R. Tamassia [39], all linear-time planarity testing algorithms can be categorized into two groups:

(1) **Vertex-addition** based algorithms introduced, in 1967, by Lempel, Even, and Cederbaum.

[5] Quantity of time it takes to run the algorithm

(2) **Cycle-based**[6] algorithms pioneered, in 1974, by Hopcroft and Tarjan.

Numerous planarity testing algorithms have been published throughout the years, many of which run in linear time. Some of these algorithms are mentioned in Table 4.1. A large number of these algorithms are modified and/or extended versions of older algorithms, few are newly discovered algorithms.

Table 4.1: *Some Planarity testing and embedding algorithms*

Year	Title	Author(s)	Time
1961	On imbedding graphs in the sphere	L. Auslandar and S. Parter	$O(n^c)$
1964	Graphes planaires: reconnaissance et construction de représentations planaires topologiques	Demoucron et al.	$O(n^2)$
1970	A new planarity test based on 3-connectivity	Bruno et al.	$O(n)$
1970	A planarity algorithm based on the Kuratowski theorem	P. Mei and N. Gibbs	$O(c^n)$
1974	Efficient planarity testing	J. Hopcroft and R. Tarjan	$O(n)$
1976	Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-Tree algorithms	K. Booth and G. Lueker	$O(n)$
1976	An algorithm for planarity testing of graphs	Lempel et al.	$O(n)$
1980	Embedding graphs in the plane — Algorithmic aspects	S. Williamson	$O(n)$
1984	Embedding planar graphs using PQ-Tree algorithms	Chiba et al.	$O(n)$
1985	A characterization of planar graphs by trémaux orders	H. Fraysseix and P. Rosenstiehl	$O(n)$
1985	A Linear algorithm for embedding planar graphs using PQ-Trees	Chiba et al.	$O(n)$

[6]Sometimes called the **path-addition** approach

1989	An optimal parallel algorithm for graph planarity	V. Ramachandran and J. Reift	$O(n)$
1991	A certain embedding of a planar graph in the plane	T. Ozawa	$O(n)$
1992	A fast linear time embedding algorithm based on the Hopcroft-Tarjan planarity test	P. Mutzel	$O(n)$
1992	An Implementation of the Hopcroft and Tarjan planarity test and embedding algorithm	K. Mehlhorn et al.	$O(n)$
1996	On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm	K. Mehlhorn and P. Mutzel	$O(n)$
1999	Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm	J. Boyer and W. Myrvold	$O(n)$
2004	On the cutting edge: simplified $O(n)$ planarity by edge addition	J. Boyer and W. Myrvold	$O(n)$
2004	Stop minding your P's and Q's: implementing a fast and simple DFS-Based planarity testing and embedding algorithm	J. Boyer, P. Cortese, M. Patrignani, and G. Battista	$O(n)$
2004	Lempel, Even, and Cederbaum Planarity Method	Boyer et al.	$O(n)$
2008	Planarity algorithms via PQ-Trees	B. Haeupler and R. Tarjan	$O(n)$
2008	Tremaux trees and planarity	H. Fraysseix	$O(n)$
2009	The left-right planarity test	Ulrik Brandes	$O(n)$

In the remainder of this chapter, we prioritize the answer to the question "How to test planarity" and elaborately discuss several algorithms.

4.1 Demoucron *et al.* Algorithm

Since Auslandar and Parter published the first-ever planarity testing algorithm that runs in polynomial-time, many similar and somewhat different algorithms have been proposed. In 1964, Demoucron, Malgrange, and Pertuiset published an algorithm that tests and embeds a given graph in quadratic time ($O(n^2)$).

Demoucron *et al.* algorithm starts with a random cycle C in G and adds (embeds) paths that have the initial and terminal vertices in C one path at a time until the whole graph is embedded. Before we discuss the intricate algorithm in detail, we need to present a few definitions that will really come in handy when implementing the algorithm.

Definition 4.1 (**Piece**)**.** Let $H = (V_1, E_1)$ be a subgraph of $G = (V, E)$. A piece[7] of the graph G with respect to H is either

- a single edge $e = \{x, y\}$ such that $e \in E$ but $e \notin E_1$ and the end-vertices of e belong to V_1.
- a connected component of $G - H$ along with the edges (and points of contact) that joins it to H.

The graph in Figure 4.1, for example, has 4 pieces relative to the subgraph H. H is the hexagon with the black colored vertices. The 4 pieces are shown at the bottom of the graph in Figure 4.1. Each piece B shares some vertices with H, those vertices are called *points of contact*[8]. For example, the fourth piece has 4 points of contact represented by the black colored vertices. A piece B is said to be a **bridge** if B has at least two points of contact. So, the last two pieces are bridges (the third and fourth pieces have 2 and 4 points of contact respectively), while the first two pieces are not bridges because both have only one contact point i.e., they both have one vertex in common with H.

A fairly large number of the algorithms in Table 4.1 rely heavily on Jordan Curve theorem, one of the central theorems in topological graph theory. Although

[7] Also known as fragment

[8] Otherwise known as contact vertices

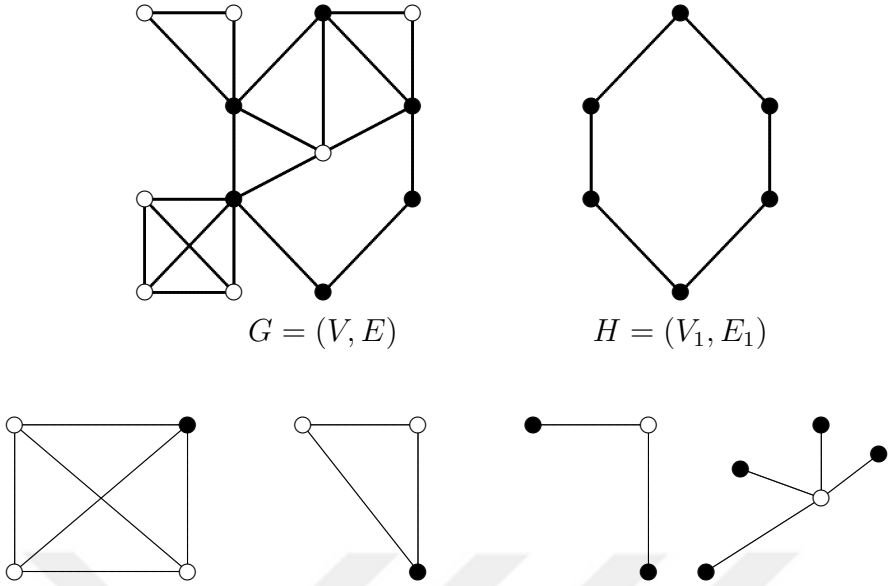


Figure 4.1: *Pieces of a graph G relative to H*

the theorem per se is intuitively obvious, a formal proof of the theorem is nonetheless far from obvious.

Theorem 4.1 (**Jordan Curve Theorem**). *Let C be a Jordan Curve drawn on the plane, then Jordan Curve Theorem states that C divides the plane into two distinct regions, an inside and an outside.*

The correctness of this result is intuitively obvious since it is pretty clear that any cycle C that is drawn on a plane divides the plane into two regions; an *interior region* and an unbounded region called the *exterior region.*

Definition 4.2. *If two bridges B_1 and B_2 cannot be planarly drawn on the same face of a Jordan Curve, then the two bridges are called* ***incompatible*** and denoted as $B_1 \not\approx B_2$.

Some authors (e.g. [39]) prefer *conflicting segments* rather than incompatible bridges, and others prefer calling incompatible bridges *competing fragments.* The graph in Figure 4.2, for example, has two competing fragments (incompatible bridges), meaning that whenever we draw both bridges on the same region of the cycle (the square on the left) at least two edges (one edge from the first bridge and one edge from the second bridge) cross.

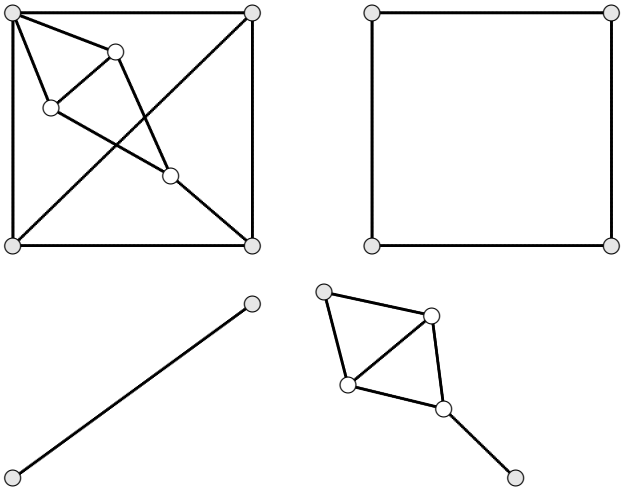


Figure 4.2: *Incompatible bridges of a graph G (top-left) w.r.t. a circuit (top-right)*

Definition 4.3. *An auxiliary graph $G^+(C)$ with respect to a cycle C is a graph whose vertices are the bridges of G and has an edge between two bridges if and only if they are incompatible.*

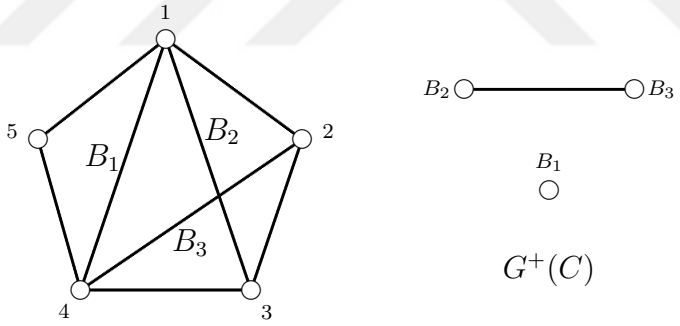


Figure 4.3: *Auxiliary graph relative to the pentagon 12345*

The auxiliary graph of the graph on the left in Figure 4.3 is shown on the right side of Figure 4.3. Observe that B_2 and B_3 are incompatible. However, B_1 is compatible with both B_2 and B_3 and hence in the auxiliary graph, there are no edges incident with the vertex that represents B_1.

Theorem 4.2. *A sufficient and necessary condition for a graph to be planar is that the auxiliary graph $G^+(C)$ with respect to any cycle C is bipartite.*

Proof. The necessity of the condition is straightforward because if the auxiliary graph is bipartite with partite sets A and B. We put all the bridges in A on one

face of the circuit and all the bridges in B on the other.

To prove that the condition is sufficient we assume that the graph is not planar and thus contains a subdivision of K_5 and/or $K_{3,3}$. Showing that the auxiliary graph of any graph that contains a subgraph of Kuratowski graph is not bipartite is enough. We will do exactly that.

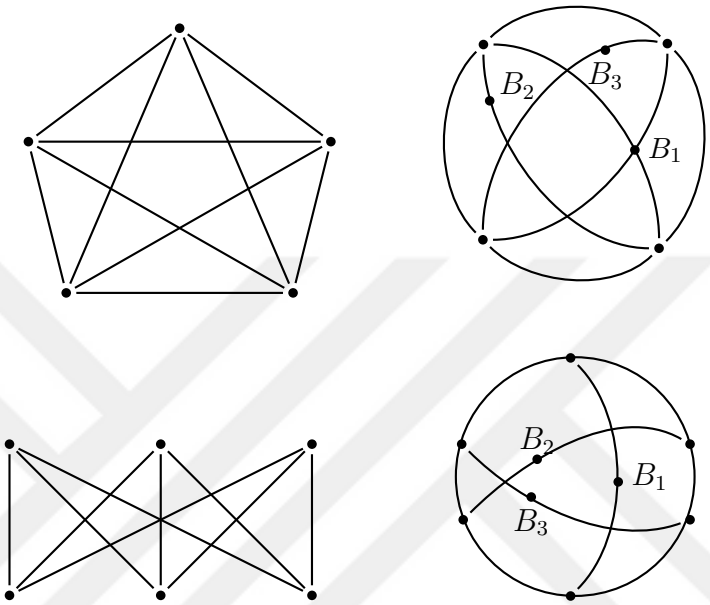


Figure 4.4: *Figure for the proof of Theorem 4.2*

In both cases, we deliberately choose a cycle so that the auxiliary graphs of both K_5 and $K_{3,3}$ are not bipartite as shown in Figure 4.4. In both cases, there are 3 bridges (B_1, B_2 and B_3) any two of which are conflicting bridges and thus for both K_5 and $K_{3,3}$ the auxiliary graph relative to the chosen cycle is the complete graph on 3 vertices i.e., $G^+(C) = K_3$. □

Definition 4.4. *A **2-connected**[9] graph is a graph that cannot be disconnected by the removal of one single vertex, i.e., the graph has no articulation points (cut-vertices).*

The first graph in Figure 4.5 is 1-connected because removing one vertex will disconnect it. The second graph, however, is 2-connected because only by deleting at least 2 vertices could be attained a disconnected graph.

[9]Better-known as **block**

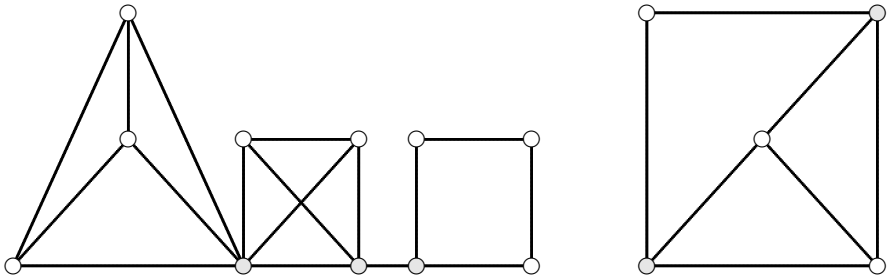


Figure 4.5: *A 1-connected graph (left) and a 2-connected graph (right)*

Definition 4.5. *Let G be a connected graph and let C a circuit in G. A path in G which has its initial and final vertices in C is called an α-path.* For example, the last two pieces on the bottom left of Figure 4.1 contain *α-path*(s). Obviously, every *α-path* is a bridge.

Definition 4.6. *Let G be a planar graph and let $H \subseteq G$. If $\widetilde{H}$ and $\widetilde{G}$ are the embeddings of H and G respectively and if $\widetilde{H} \subseteq \widetilde{G}$, then $\widetilde{H}$ is called G-**admissible**.*

Consider, for example, the graphs in Figure 4.6. In (a) A graph G is shown and two different planar representation of the same graph ($H = G - e$) is shown in (b) and (c). The graph in (c) is $G - admissible$ because the removed edge can be added to the graph in such a way that the graph remains planar and without moving any vertices, the same is not true for the figure in (b) and thus, it is not $G - admissible$.

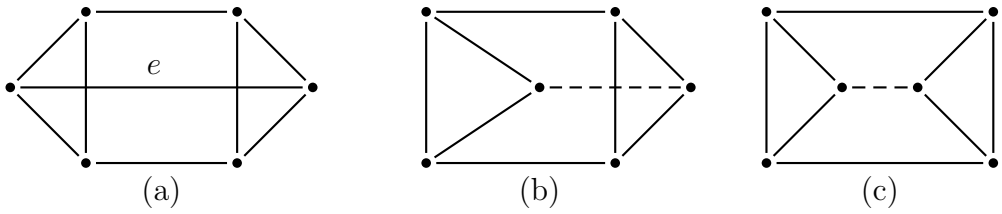


Figure 4.6: *(a) G; (b) G-inadmissible; (c) G-admissible*

An **admissible face** of a bridge B is a region of G in which the bridge B can be drawn planarly. In other words, it contains all points of contact of B, and the bridge B is said to be drawable in that region. We denote $F(B, \widetilde{H})$ the set of regions of $\widetilde{H}$ in which B can be planarly drawn.

Theorem 4.3. Let $\widetilde{H}$ be G-admissible, then for each and every bridge B the

cardinality of the set of region in which B is drawable is different from zero. In other words, $F(B, \widetilde{H}) \neq 0$.

Proof. Since $\widetilde{H}$ is G-admissible, there exist an embedding $\widetilde{G}$ of G in which $\widetilde{H}$ is a subset of $\widetilde{G}$, and hence there is at least one face in which B can be drawn and thus $F(B, \widetilde{H}) \neq 0$. □

4.1.1 Elementary reduction

At this point, we should mention that *Demoucron et al.* algorithm presumes that the graph under consideration is 2-connected. In other terms, the algorithm deals with blocks since its first step is to look for a random circuit. In light of this fact, there are some preprocessings through which we can accomplish a great deal of task simplification. Taking this into consideration, we present these simplifications:

(1) If the graph contains many components, then each component is separately subjected to the test.

(2) If the graph contains one or more cut-vertices (G is separable), then we deal with each block separately.

(3) Multiple edges (parallel edges) and self-loops can be deleted with zero effect on planarity.

(4) Each graph can be reduced to some homeomorphic graph with the minimum order by deleting all degree 2 vertices and replacing them with a single edge, again without affecting the planarity of the graph.

(5) If the size of a graph G is less than or equal to 8, then G is planar. This is so because the smallest size of any nonplanar graph is 9 and that is $K_{3,3}$.

(6) If the order of a graph is less than or equal to 4, then the graph is planar. This is true because the minimum order of a nonplanar graph is 5, and that non-planar graph is K_5.

(7) If in any graph $m > 3n - 6$, then that graph is nonplanar by Corollary 3.1.1

Applying these preprocessings before subjecting the graph to the intricate algorithm, can result in a great deal of simplifications and save a lot of time in the processing. A graph that seems to be complex and difficult to test for planarity at first, can sometimes be easily tested for planarity with the help of the above simplifications. For example, consider the graph in Figure 4.7. If we apply the Demoucron et. al algorithm before we preprocess the graph, we have to deal with a very large 20-vertex graph, but if we disconnect the graph at its articulation points, we have 3 blocks and we deal with each block separately, as shown in Figure 4.7.

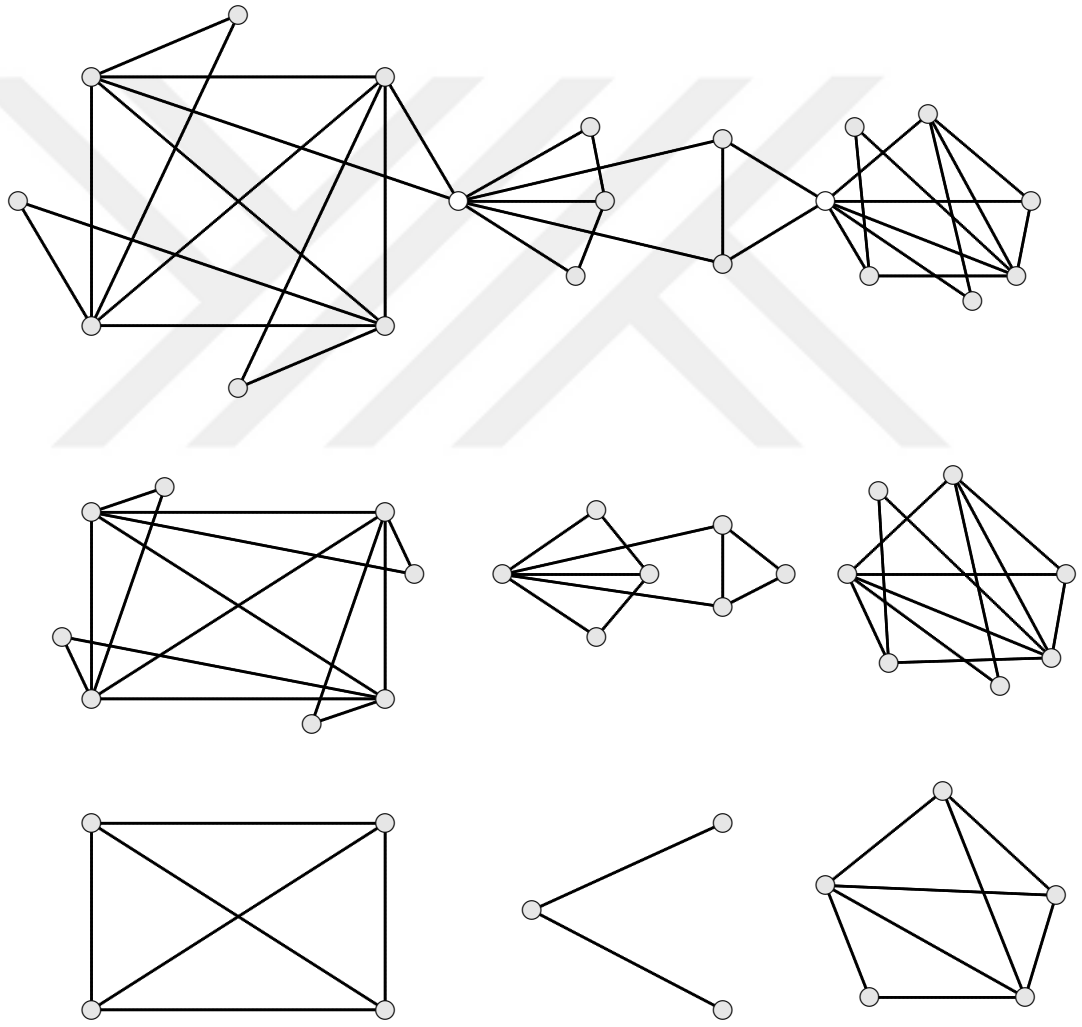


Figure 4.7: *Demonstrating preprocessings*

The order of the first and second components of the graph after removing the articulation points is less than 5, and the size of the third component is less than 9 and therefore all the blocks of the graph in Figure 4.7 are planar. Thus we conclude

that the graph under consideration is planar. In general, each and every block of a graph is planar if and only if the graph is planar.

If these preprocessings fail to determine the planarity of the graph in question, we subject the graph to **Demoucron et al. algorithm**, a more elaborate test.

4.1.2 The algorithm

Demoucron *et al.* algorithm tests graph planarity in **quadratic time**. i.e., the algorithm determines the planarity of a graph of order n in a time frame of at most n^2. This quantity is the maximum amount of time the algorithm may take to run and find a result and its commonly known as the **worst-case time-complexity**. On the other hand, the smallest amount of time in which an algorithm executes an order is called the **best-case** scenario.

Now, before we present the algorithm in its entirety, let us see a general overview of the algorithm by considering the main steps of the algorithm:

- ✓ Find an arbitrary cycle C of G.
- ✓ Find all bridges of G with respect to C.
- ✓ Compute $F(B, \widetilde{H})$ for each bridge B.
- ✓ If there is a bridge B such that $F(B, \widetilde{H}) = \emptyset$, then the graph is nonplanar.
- ✓ If there exists a bridge B such that $F(B, \widetilde{H}) = 1$, go to the last step.
- ✓ If for each bridge $F(B, \widetilde{H}) \geq 2$, then choose a random bridge.
- ✓ Find an $\alpha - path$ and embed it. Repeat the process until all bridges are embedded.

The graph that we are trying to determine its planarity is, of course, a nonseparable (block) graph, since every separable graph, is preprocessed first. In other words, the input graph of the algorithm is a 2-connected graph and the output of the algorithm is either a plane embedding if the graph is planar or a declaration that the graph is not planar.

The number of **iterations** in the algorithm is one less than the number of regions of the graph, this is because the algorithm commences with one region already embedded. Thanks to Euler's formula we know that the number of faces (regions) of a planar graph G is $f = m - n + 2$, and hence the number of iterations in the algorithm is exactly $m - n + 1$ (m: edges, n: vertices and f: regions). Each iteration can be done in $O(n)$-time, since finding bridges, and computing the number of admissible faces of a bridge can be achieved in linear-time. Furthermore, finding an $\alpha-$path and embedding it can be done in constant-time.

Algorithm 1 is self-explanatory. Besides the above general overview of the Algorithm 1 sheds some light on it. Now, we take one more step to make sure that the steps in the algorithm are completely and utterly comprehended by giving examples on how to apply the algorithm manually. But first, let us reiterate that we embed one $\alpha-$path at a time and not a whole bridge as this might result in the algorithm failing to correctly test planarity.

Illustrative example 1: Determine whether the graph given in Figure 4.8 is planar or nonplanar.

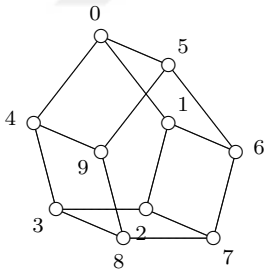


Figure 4.8: *Input graph G for illustrative example 1*

Before we commence applying the algorithm, first let us note that the number iterations in the algorithm for this particular graph is

$$m - n + 1 \Longrightarrow 15 - 10 + 1 = 6 \text{ iterations.}$$

The iterations in the algorithm are listed in Table 4.2.

In iteration 1, we chose and embedded a random cycle $\widetilde{H}_1 = 01234$. Obvious $\widetilde{H}_1$ has two faces (f_1 and f_2) and the original graph G has only one bridge (B_1) with respect to $\widetilde{H}_1$. All the operations related to iteration 1 are shown in the row with iteration number 1 of Table 4.2.

Algorithm 1 *Demoucron et al. algorithm*

Input: a block graph

Output: `true` if the graph is planar, `false` if not

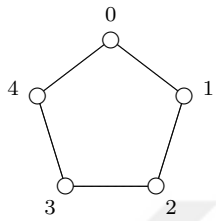
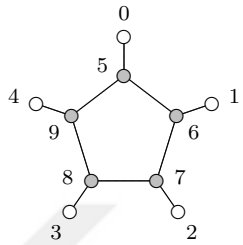
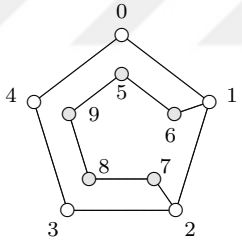
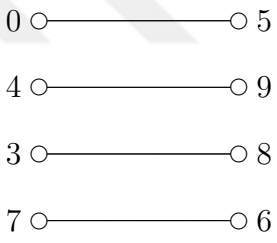
1: **if** $m > 3n - 6$ **then**
2: return `false`
3: **else**
4: H $\leftarrow$ any circuit C of G
5: $f \leftarrow 2$ ▷ Inner and Outer faces
6: **while** $f \neq m - n + 2$ **do** ▷ $H \neq G$
7: $B \leftarrow$ all bridges of G
8: **for** each brige B **do**
9: find $F(B, \widetilde{H})$
10: **if** $\exists B$ such that $F(B, \widetilde{H}) \neq \emptyset$ **then**
11: return `false`
12: **if** $\exists B$ such that $F(B, \widetilde{H}) = 1$ **then**
13: $B_0 \leftarrow B$
14: **else**
15: $B_0 \leftarrow$ any bridge B
16: $P \leftarrow$ any $\alpha-$path of B_0
17: $H_{i+1} \leftarrow H_i \cup P$ ▷ update the graph H_i
18: $f_{i+1} \leftarrow f_i + 1$ ▷ update the faces of H_i
19: **if** $f = m - n + 2$ **then**
20: return `true`
21: **end**

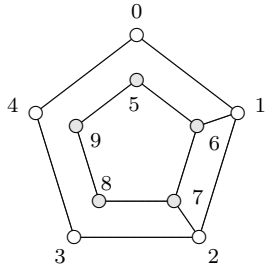
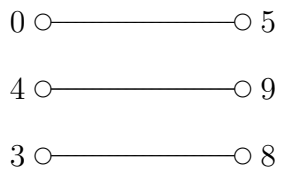
Before we start iteration 2, we choose an α-path ($P_1 = 1659872$) form B_1 in iteration 1.

In iteration 2, we first embed the α-path P_1, and after embedding P_1 we determine the bridges of G with respect to $\widetilde{H}_2$, and we notice that G has 4 bridges relative to $\widetilde{H}_2$. G also has 3 faces. Now, we determine the admissible faces of each bridge and then before begin iteration 3 we choose an α-path $P_2 = 67$.

In iterations 3,4,5 we go through similar steps and in iteration 6 we find a plane embedding of G.

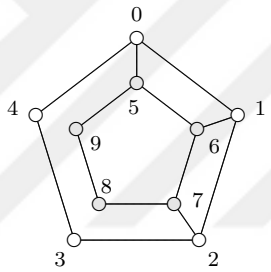
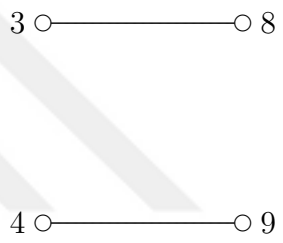
Table 4.2: *Application of Demoucron et al. algorithm on a planar graph*

<table>
<tr><td>Iteration
(i)</td><td>$\widetilde{H}_i$</td><td>Bridges of G w.r.t. $\widetilde{H}_i$</td></tr>
<tr><td>1</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">$F_1 = 01234$, $F_2 = 01234$ $F(B, \widetilde{H}_1) = f_1, f_2$
Since the bridge has multiple admissible faces we pick
any α-path, say $P = 1659872$</td></tr>
<tr><td>2</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">$F_1 = 0165987234$, $F_2 = 1659872$, $F_3 = 01234$
Again we list the admissible faces of G. $F(\{0,5\}, \widetilde{H}_2) = F_1$,
$F(\{4,9\}, \widetilde{H}_2) = F_1$, $F(\{3,8\}, \widetilde{H}_2) = F_1$, $F(\{7,6\}, \widetilde{H}_2) = F_2$.
Now we choose $P = 67$ as the second $\alpha-$path.</td></tr>
</table>

3		

$F_1 = 0165987234, \quad F_2 = 56789, \quad F_3 = 1672, \quad F_4 = 01234$

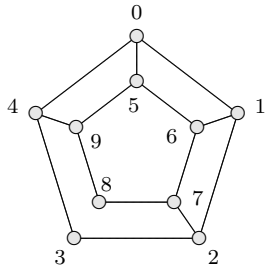
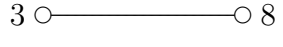
Once more we list the admissible faces of G. $F(\{0,5\}, \widetilde{H}_3) = F_1$, $F(\{4,9\}, \widetilde{H}_3) = F_1, \quad F(\{3,8\}, \widetilde{H}_3) = F_1$, and once again we choose $P = 05$ as the third $\alpha-$path.

4		

$F_1 = 05987234, \quad F_2 = 0165, \quad F_3 = 1672, \quad F_4 = 56789,$

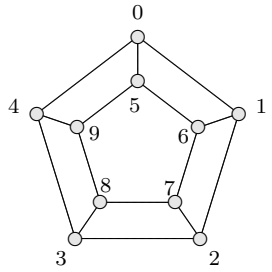
$F_5 = 01234.$

Now we compute the admissible faces of G. $F(\{4,9\}, \widetilde{H}_4) = F_1$, $F(\{3,8\}, \widetilde{H}_4) = F_1$, and once again we pick $P = 49$ as the fourth α-path.

5		

$F_1 = 0594, \quad F_2 = 0165, \quad F_3 = 1672, \quad F_4 = 234987,$

$F_5 = 56789\ , \quad F_6 = 01234$

<table>
<tr><td colspan="3">Now we compute the admissible faces of G. $F(\{3,8\},\widetilde{H}_4) = F_4$, and lastly we embed $P = 38$ as the fifth $\alpha-$path.</td></tr>
<tr><td>6</td><td> 
</td><td></td></tr>
<tr><td colspan="3">At iteration 6, as expected, the algorithm terminates. All edges of the original graph have been planarly embedded. In other words, $E(G) - E(\widetilde{H}_6) = \emptyset$. We notice that $m = 15$, $n = 10$, $f = 7$ and hence $f = m - n + 2 \Longrightarrow 7 = 15 - 10 + 2 \Longrightarrow 7 = 7$. Therefore the graph is planar. Furthermore, G and $\widetilde{H}_6$ are isomorphic since they are basically the same graph.</td></tr>
</table>

Illustrative example 2: Determine whether the graph given in Figure 4.9 is planar or nonplanar

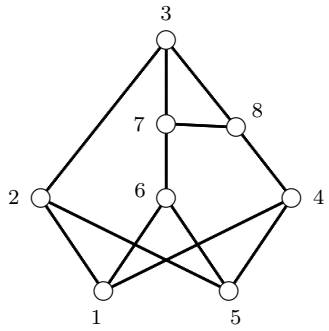


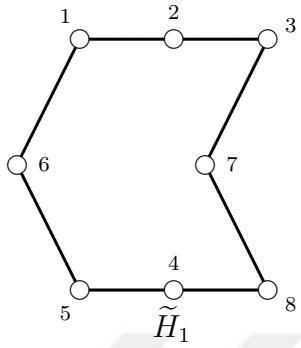
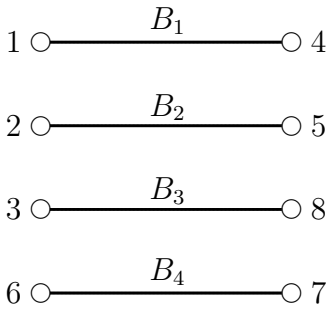
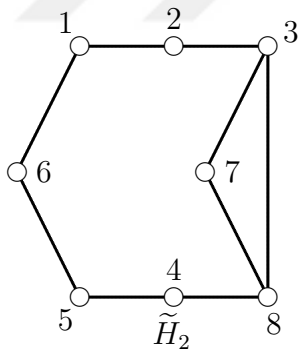
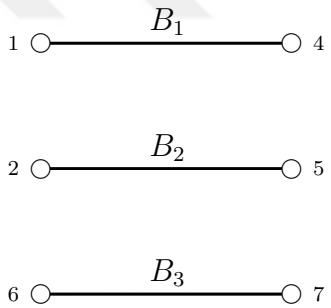
Figure 4.9: *Input graph G for illustrative example 2*

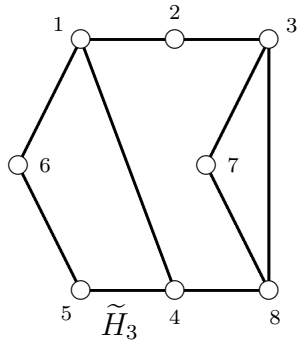
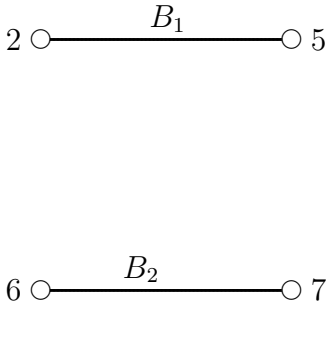
Before subjecting the graph to the intricate algorithm, let us calculate the number of iteration in the algorithm which is

$$m - n + 1 \Longrightarrow 12 - 8 + 1 = 5 \text{ iterations.}$$

Just like in the illustrative example 1, the main steps are summarized in Table 4.3. As shown in the table the algorithm runs smoothly through iteration 1, 2, and 3. However, $F(B_3, \widetilde{H}_3) = \emptyset$ at iteration 4, and hence the algorithm stops.

Table 4.3: *Application of Demoucron et al. algorithm on a nonplanar graph*

<table>
<tr><th>Iteration # (i)</th><th>$\widetilde{H}_i$</th><th>Bridges of G w.r.t. $\widetilde{H}_i$</th></tr>
<tr><td>1</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">$F_1 = 12378456, \quad F_2 = 12378456.$
Now we compute the admissible faces of G. $F(B_1, \widetilde{H}_1) = F_1, F_2$, $F(B_2, \widetilde{H}_1) = F_1, F_2$, $F(B_3, \widetilde{H}_1) = F_1, F_2$, $F(B_4, \widetilde{H}_1) = F_1, F_2$, . Next we embed $P = 38$ on F_2, the outer face.</td></tr>
<tr><td>2</td><td> 
</td><td> 
</td></tr>
<tr><td colspan="3">$F_1 = 12378456, \quad F_2 = 378, \quad F_3 = 1238456.$
Now we compute the admissible faces of G. $F(B_1, \widetilde{H}_2) = F_1, F_3$, $F(B_2, \widetilde{H}_2) = F_1, F_3$, $F(B_3, \widetilde{H}_2) = F_1$. Next we embed $P = 14$ on F_1. We should note that $F(B_3, \widetilde{H}_2) = 1$ and according to the algorithm we should choose our α-path from this bridge, However, we choose another more convenient bridge ,in the end, it makes no difference for this particular graph.</td></tr>
</table>

3		

$F_1 = 12384, \quad F_2 = 1456, \quad F_3 = 378, \quad F_4 = 1238456.$

Now we compute the admissible faces of G. $F(B_1, \widetilde{H}_3) = F_4$, $F(B_2, \widetilde{H}_3) = \emptyset$. Since the bridge B_2 has no admissible faces, we conclude that the graph is nonplanar.

Theorem 4.4. *Every iteration in the Demoucron et al. algorithm generate a partial plane drawing $\widetilde{H}_i$ and thus the Demoucron et al. algorithm for testing planarity is valid.*

Proof. We prove this theorem by induction on the number of iterations in the algorithm, say n.

- **Base case:** $n = 1$.

  H is just a circuit of G and can be simply redrawn by deleting vertices and edges from any plane drawing of G.

- **Induction hypothesis:** from G_{n-1} to G_n

  Assuming that we begin with a plane embedding of H, let us examine steps of the algorithm. Since we are assuming that G is planar, for each bridge there is at least one admissible face. If this was not the case, there would be an $\alpha-$path in a bridge joining distinct faces of the partial plane drawing, which would be a partial plane drawing of a nonplanar graph G.

  After computing the bridges of the graph, the algorithm builds a partial plane

drawing G_n during the steps 12—16. During these steps, we consider the following two cases:

(1) There is a bridge with exactly one admissible region. In this particular case, there exist exactly one way to draw an $\alpha-$path, and the algorithm simply generates a partial plane drawing.

(2) Each bridge has at least two admissible regions. In this case, every bridge has multiple admissible faces and no matter which admissible face we choose to embed a given bridge the algorithm will still generate a partial embedding of G. Since each iteration generates a partial embedding of G and the algorithm is a repetition of these iterations we conclude that the Demoucron et al. algorithm is valid.

□

For more information on Demoucron et al. algorithm refer to [4, 14, 15, 18, 20, 24, 31, 41].

4.2 The Left-Right Planarity Algorithm

The Left-right algorithm is one of the oldest $O(n)$-time planarity algorithms and was first proposed by Hubert de Fraysseix and Pierre Rosenstiehl in 1985 and later modified and simplified in 2008 and 2009 by Fraysseix [16] and Ulrik Brandes [10] respectively. Just like Algorithm 1 the Left-right algorithm (also known as *The Fraysseix and Rosenstiehl* algorithm) also builds on the Jordan Curve, but more importantly it heavily relies on the Depth-first search algorithm.

4.2.1 Depth-first search (DFS) algorithm

As the name suggests Depth-first search (DFS) algorithm is a systematic way of searching graphs, i.e., visiting all vertices of a given graph without visiting any vertex too many times.

In depth-first search, we explore as many vertices as possible along each path before backtracking. For example, consider Figure 4.10. In order to perform DFS, we begin at vertex 1 and go down to 2, 5 and 9. Since there are no new vertices to explore from 9, we backtrack to 5 and then go down to 10. Once again we backtrack to 2 through 5 from which we visit 6 and 11, then we backtrack to 1 via 6 and 2, from which we move down to 3, 15, and 16. Backtracking to 1 through 15 and 3 takes us to 4 and eventually to 12, 13, and 14.

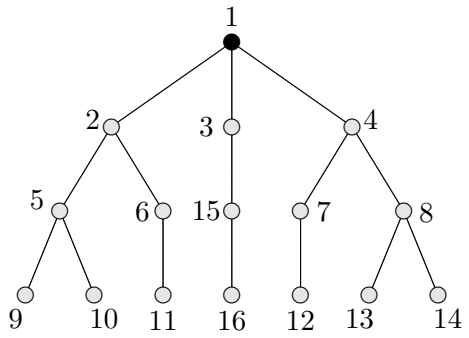


Figure 4.10: *Depth-first search*

Executing DFS on a graph divides the edges of the graph into two distinct sets of edges [18]:

(1) The edges that the algorithm uses to discover (visit) new nodes (vertices) are called **tree edges**, because they form a tree.

(2) The edges that the DFS algorithm does not use to discover new edges are called **back edges** (also known as *co-tree edges*).

It can be easily seen that $E = T \cup B$, where E represents the set of edges of the graph, T represents the tree edges and B represents the back edges. For example, consider the graph G in Figure 4.11, combining the tree edges in graph T and the back edges removed from T makes G.

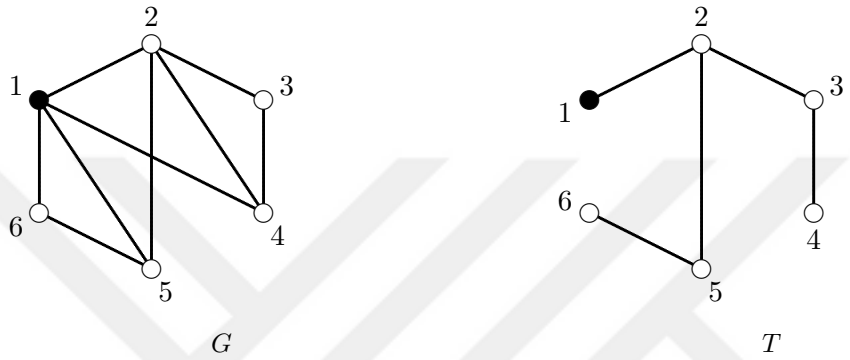


Figure 4.11: *(a) A Graph G and (b) its DFS-tree*

When performing the Depth-first search algorithm, each vertex is assigned to a number in the order in which they are visited. We call these number DFS *index* and we write $DFS(v)$ to denote the DFS index of the vertex v. For example, in Figure 4.10, $DFS(5) = 3$ because vertex 5 is the third vertex reached during the Depth-first search visit. Performing DFS on the graph in Figure 4.10 and labeling vertices in the order in which they are visited gives us the graph in Figure 4.12.

In any tree, there is a vertex called the **root** (vertex 1 in Figure 4.10, Figure 4.11, and Figure 4.12) which is the vertex that DFS visits start with. A vertex u is said to be an *ancestor* of another vertex v if DFS visits u before v, otherwise, u is said to be a **descendant** of v. In Figure 4.12, for example, the descendants of 2 are 3, 6, 4, 5, and 7; the ancestors of 13 are 12, 11, and 1. A vertex can not be a descendant or ancestor of itself. So, vertex 1, doesn't have any parents (ancestors) but, obviously, it has many descendants.

The **height** of a vertex v is defined as the distance between that vertex and the root of the DFS tree. In other words, it is the number of tree edges it takes to

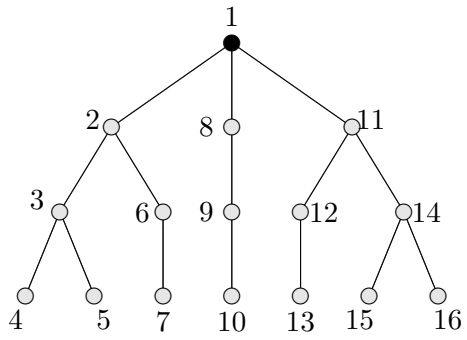
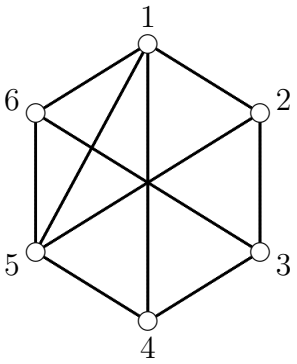
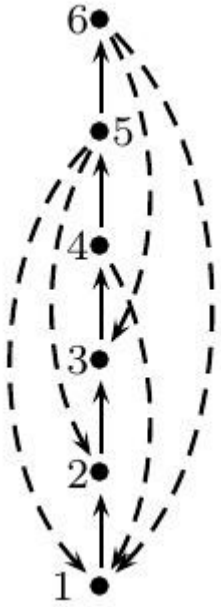


Figure 4.12: *DFS index*

travel from the root to that vertex v. For example, in Figure 4.13(b) it is clear that vertex 3 is closer to the root then vertex 4. We say vertex 3 is lower than vertex 4. **Tree edges** connect ancestors to descendants, i.e., tree edges join a vertex with a lower DFS index to a vertex with a higher DFS index (see the tree edges represented by the solid line in Figure 4.13(b)). On the other hand, **back edges** join a descendant to an ancestor, which means that back edges connect a vertex with a higher DFS index to a vertex with a lower DFS index (see the back edges represented by the dashed lines in Figure 4.13(b)). Mathematically, if $e = \{x, y\}$ and $e \in T$, then $DFS(x) < DFS(y)$ and if $e \in B$ then $DFS(x) > DFS(y)$.



(a) G



(b) T

Figure 4.13: *(a) A Graph G; (b) it's DFS orientation*

We define the *lowpoint* of a vertex v as the ancestor of v with the lowest DFS

index accessible via a back edge from a descendant of v [32]. In Figure 4.13, for instance, $lowpt(3) = 1$ because from vertex 3 we can reach as low as vertex 1 through a back edge from 5 or 6 (both are descendants of 3).

A back edge $e = \{a, b\}$ is said to be a **returning edge** for itself and for every tree edge $f = (u, v)$ such that $DFS(u) > DFS(b)$ and $DFS(a) \geq DFS(v)$ [10].
A cycle in a DFS oriented graph is said to be a **fundamental cycle** if it is formed by one single back edge and several tree edges. For example, in Figure 4.13(b), the cycle 123451 is a fundamental cycle because it is made of 4 tree edges and 1 back edge. Two fundamental cycles having at least one edge in common are called **overlapping cycles**, and we should be careful the way we draw these overlapping cycles because they can cause crossings. If two fundamental cycles do not overlap they do not cause planarity problems. Two fundamental cycles can joint and fork a maximum of one time because they can only share tree edges. The last shared edge by two overlapping cycles plus the succeeding two edges is said to be a ***fork***. A tree edge $e = \{a, b\}$ and $e_1 = \{b, x\}$ and $e_2 = \{b, y\}$ together form a fork provided e is the last shared edge. In Figure 4.14, for example, the two cycles 12341 and 23452 share the tree edges $\{2, 3\}$ and $\{3, 4\}$. Therefore the tree edge e together with e_1 and e_2 constitute a fork (A vertex with one incoming tree edge and at least 2 outgoing edges).

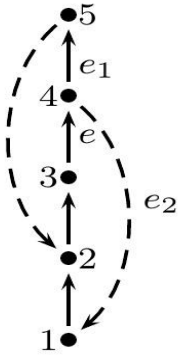


Figure 4.14: *A fork*

In Figure 4.14, the cycle enclosed by e_1 is oriented left of the DFS tree and we say it is oriented counter-clockwise, and the back edge e_1 is said to be a *left edge*, while the cycle enclosed by e_2 is oriented right of the DFS tree and it is said to be oriented clockwise (e_2 is called *right edge*). For every two overlapping cycles,

there are four distinct possible orientations as shown in Figure 4.15. In the first two cases of Figure 4.15 both the edges are drawn on the same side of the DFS tree. If two back edges are drawn on the same side in every planar embedding of the graph they are said to be **T-alike**, and if they are drawn on different sides in every planar embedding of the graph they are said to be **T-opposite**. For instance, the two back edges in Figure 4.15 are drawn on the same side of the DFS tree and hence they are T-alike. On the other hand, the edges $\{5, 2\}$ and $\{6, 3\}$ in Figure 4.13 are T-opposite.

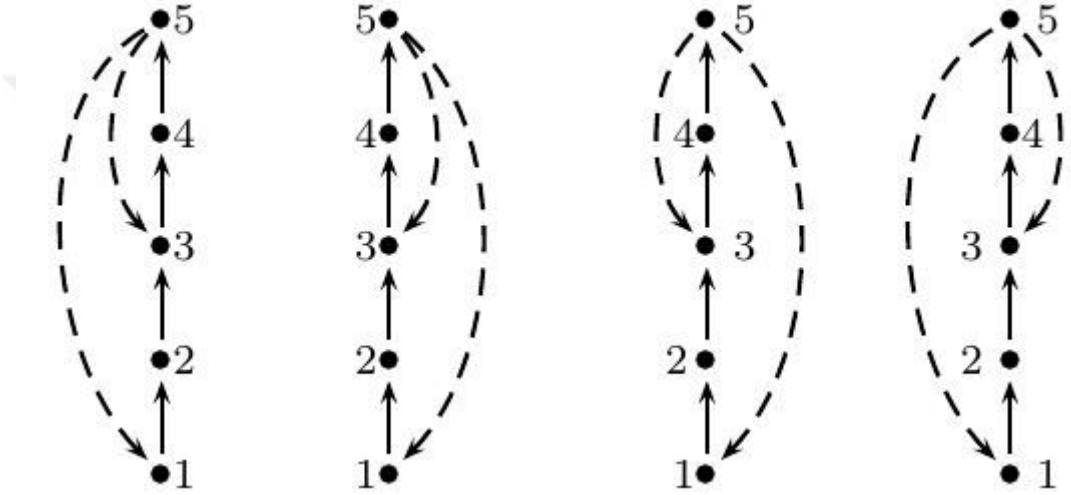


Figure 4.15: *Possible orientations of two overlapping cycles*

Two overlapping cycles are said to be **nested** if the non-shared edges of one cycle can be drawn entirely inside the other cycle.

Observation 4.2.1. *In any planar embedding of a DFS-oriented graph, two overlapping cycles are nested (one enclosed inside the other) if and only if they are T-alike.*

It is quite clear that if two back edges are T-opposite i.e., they can not be drawn on the same side in any planar drawing of the graph, then obviously they can not be nested.

Observation 4.2.2. *In any planar embedding of the DFS-oriented graph, overlapping cycles (back edges) are nested according to their lowpoint order, provided that the root is on the external face.*

Referring back to Figure 4.15 it is clear that overlapping cycles should be oriented according to Observation 4.2.2. If this was not the case, the two back edges would be forced to cross.

Observation 4.2.3. *In any planar drawing (drawn with the help of The Left-Right algorithm) of a DFS-oriented graph, the root is always on the outer face.*

Observe that the back edge $\{5, 2\}$ in the first graph of Figure 4.16 has a higher lowpoint than $\{5, 1\}$ ($lowpt(\{5, 2\}) = 2$, and $lowpt(\{5, 1\}) = 1$). Hence, according to Observation 4.2.2, circuit 23452 has to be drawn completely inside the cycle 123451, should they be drawn on the same side. Similarly in the second graph, the root is inside a bounded face which is not allowed according to observation 4.2.3.

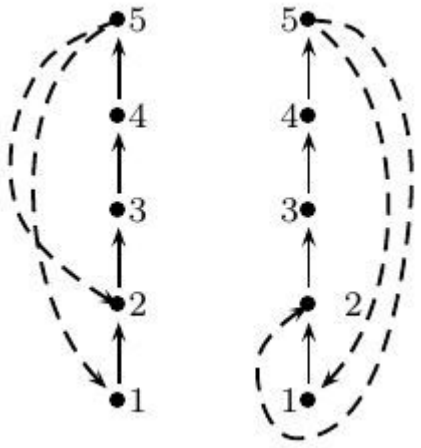


Figure 4.16: *Two forbidden cases*

The next definition is unrivaled in terms of significance, since determining planarity using the Left-Right algorithm basically boils down to assigning each back edge to one class (left or right). This definition is based on the concept of *fork* and thus **LR-partition** is sometimes called the **fork condition.**

Definition 4.7 (**LR-partition**). [39] *If G is a DFS-oriented graph. A partition $B = L \cup R$ (where B: back edges, L: left edges, R: right edges) is said to be a left-right partition if for every fork containing $e \in T$, and $e_1, e_2 \in B$*

(1) *All return edges of e_1 whose lowpoints are strictly higher than the $lowpt(e_2)$ belong to one side (either left or right) and*

(2) *All return edges of e_2 with lowpoints strictly higher than lowpoint of e_1 belong to the other side.*

We now present the left-right planarity algorithm criterion.

Theorem 4.5 (**LR planarity algorithm criterion**). *A necessary and sufficient condition for a graph to be planar is that it admits a left-right partition.*

The necessity of the above theorem is quite straightforward. If the back edges of G can be planarly divided into two classes (left and right), then we have a planar embedding of G. The proof of the sufficiency condition is long, complicated, and thus excluded (curious reader could refer to [10, 16, 39]).

Lemma 4.1. *Given an LR partition of back edges of a graph, a left-right ordering yields a planar drawing.*

If we are given an LR partition of back edges of a graph G, applying the fork condition and taking the above observations into account we can easily deduce a planar drawing of the graph G. As a matter of fact, the fork condition is the center where the left-right planarity algorithm revolves.

4.2.2 The algorithm

We now present the linear-time left-right planarity algorithm. The algorithm consists of three phases; first, it orients the graph using depth-first search; second, the algorithm traverses seeking an LR partition; lastly, it computes a planar embedding.

The input of the algorithm is a simple, undirected graph and the output is either a plane embedding of the graph or a declaration that the graph is not planar.

Phase 1: *Orientation:*

Using depth-first search we orient the input graph.

Phase 2: *Traversing:*

Applying the fork condition we divide the back edges of the graph into two classes (left and right). If we successfully obtain an LR partition, we go to phase 3. If, however, we arrive at the conclusion that one back edge belongs to both classes, we declare that the graph is nonplanar.

Phase 3: *Embedding:*

Embed the left back edges on the left side of the DFS tree, and the right back edges on the right side.

The best way to complete the discussion and guarantee a full comprehension of the left-right planarity algorithm is almost certainly to execute it on some graphs, and hence we conclude the discussion of the left-right planarity algorithm by running it through two graphs: a planar graph and a nonplanar graph.

Illustrative example 1 Using the left-right planarity algorithm discuss the planarity of the graph in Figure 4.17.

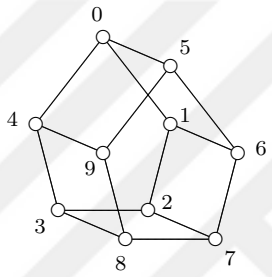


Figure 4.17: *Input graph G for illustrative example 1*

In **phase 1**, we orient the graph using the depth-first search algorithm to find a DFS orientation of the graph in Figure 4.17. The tree path 0-1-2-3-4-9-5-6-7-8 represents the tree edges and the dashed downward curves represent back edges. It is these dashed lines (back edges) that we need to classify as either left or right edges, this is done in phase 2.

In **phase 2**, we determine on which side of the DFS graph each back edge has to be drawn in every planar embedding of the graph. We clearly state which back edges must be drawn on the left, which must be drawn on the right and which back edges does not matter which side they are drawn.

As we have mentioned earlier, a fork is a vertex with at least one incoming edge and at least two outgoing edges. Looking back at Figure 4.18, we see that vertex 0 can not contain a fork because it has only one outgoing edge. The same is true for vertices 1, 2, 3, and 9. However, vertices 4, 6, 5, 7, and 8 all have 1 incoming and at least 2 outgoing edges and thus constitute a fork. We examine those forks one by one, using fork condition, to determine which side each back edge goes. We

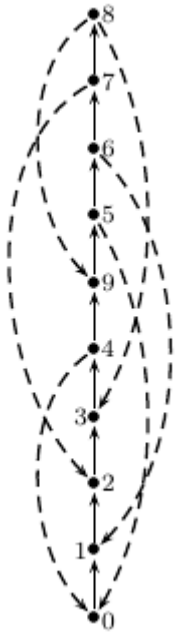


Figure 4.18: *DFS oriented input graph*

start with the first fork ($e = \{3, 4\}$, $e_1 = \{4, 0\}$ and $e_2 = \{4, 9\}$). Since the lowpoint of e_1 and e_2 is 0 and 1 respectively, we put all back edges of e_2 with lowpoint strictly higher than 0 (the lowpoint of e_1) on one side, say left. Those edges are $\{6, 1\}$, $\{7, 2\}$ and $\{8, 3\}$. However, there are no edges to put on the other side (the right side) since there are zero back edges of e_2 with lowpoint higher than the lowpoint of e_1. Applying the fork condition on the remaining forks we obtain the LR-partition given in Table 4.4.

Table 4.4: *Summary of the application of the LR planarity algorithm on a graph*

#	Fork			$lowpt(e_1)$	$lowpt(e_2)$	L edges	R edges
	e	e_1	e_2				
1	$\{3, 4\}$	$\{4, 0\}$	$\{4, 9\}$	0	1	$\{6, 1\}, \{7, 2\}, \{8, 3\}$	None
2	$\{5, 6\}$	$\{6, 1\}$	$\{6, 7\}$	1	2	$\{7, 2\}, \{8, 3\}, \{8, 9\}$	None
3	$\{6, 7\}$	$\{7, 2\}$	$\{7, 8\}$	2	3	$\{8, 3\}, \{8, 9\}$	None
4	$\{7, 8\}$	$\{8, 3\}$	$\{8, 9\}$	3	9	$\{8, 9\}$	None

As clearly stated in Table 4.4, the back edges $(6,1),(7,2),(8,3)$, and $(8,9)$ must be drawn on the same side of the DFS graph in any planar embedding of the input graph. The side we choose to embed these edges is irrelevant, it could be left or right. On the other hand, the back edges $(4,0)$ is a free edge, it can be drawn on any side. The embedded version of the input graph with the help of the left-right planarity algorithm, summarized in Table 4.4, is shown in Figure 4.19.

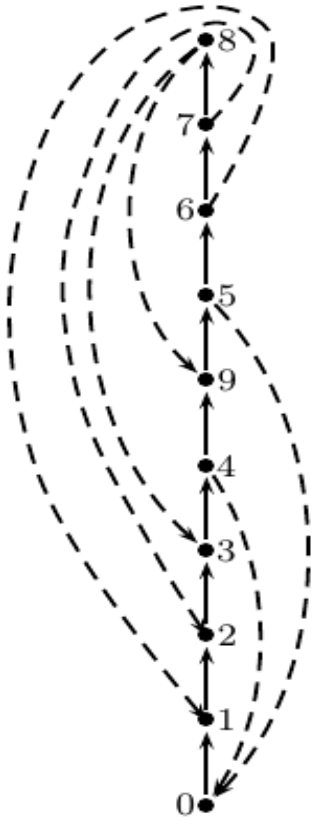


Figure 4.19: *Embedment of the input graph in Figure 4.17*

Illustrative example 2: Determine the planarity of the graph shown in Figure 4.20.

Just like in illustrative example 1, we apply the algorithm on the graph through its three phases. First, we start DFS orienting the graph as shown in Figure 4.21.

In the second phase of the algorithm, we try to obtain an LR partition, if possible. If it turns out that an LR partition of the back edges is not attainable, then we obtain a contradiction that one back edge belongs to both left and right side of the DFS tree.

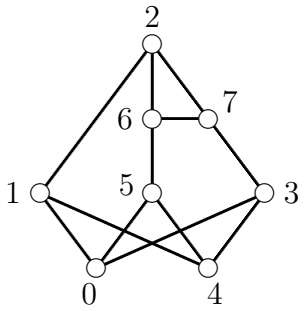


Figure 4.20: *Input graph for illustrative example 2*

Figure 4.21: *DFS orientation of Figure 4.20*

Table 4.5: *Application of the LR planarity algorithm on a graph*

#	Fork			$lowpt(e_1)$	$lowpt(e_2)$	L edges	R edges
	e	e_1	e_2				
1	$\{6,5\}$	$\{5,0\}$	$\{5,4\}$	0	0	$\{4,1\}$, $\{7,2\}$, $\{7,6\}$	$\emptyset$
2	$\{5,4\}$	$\{4,1\}$	$\{4,3\}$	1	0	$\{7,2\}$, $\{7,6\}$	$\{4,1\}$

Considering the first two iterations of the algorithm, As shown in Table 4.5, we observe that the edge $(4,1)$ belongs to both classes, left and right, which is impossible. Consequently, the results of the two iterations contradict each other. At this point, there is no need to further continue applying the algorithm, and we simply deduce that the input graph in Figure 4.20 is nonplanar.

Lastly, before we bring an end to the discussion of the left-right planarity algorithm, we should point out that the number of iterations in the algorithm is linearly related to the number of forks the DFS oriented version of the graph contains, as each fork represents one iteration. Although it is pointless to continue with the iterations if we come face to face with a contradiction as in illustrative example 2.

For additional information on this algorithm, see [10, 16, 23, 39].

4.3 Boyer-Myrvold Planarity Algorithm

Boyer-Myrvold planarity test algorithm [6] is another linear-time algorithm that heavily relies on the edge addition approach. Boyer-Myrvold algorithm alludes to Booth & Leuker [5], Hopcroft & Tarjan [22] and Williamson [42]. This algorithm commences with several preprocessings, the first of which is to perform depth-first search (DFS) which consequently produces a DFS tree, and in the process, as described earlier, creating a partition in the edges of the graph into two groups: *tree edges* and *back edges*.

Each vertex v in the DFS oriented graph has some incoming and outgoing edges. Let us denote $B_{in}(v)$ as the set of incoming back edges entering the vertex v, while $B_{out}(v)$ represents the set of outgoing back edges leaving the vertex v.

The Boyer-Myrvold algorithm, after creating a DFS tree, embeds the back edges and processes the vertices one by one in reverse DFS-index order as shown in Figure 4.22.

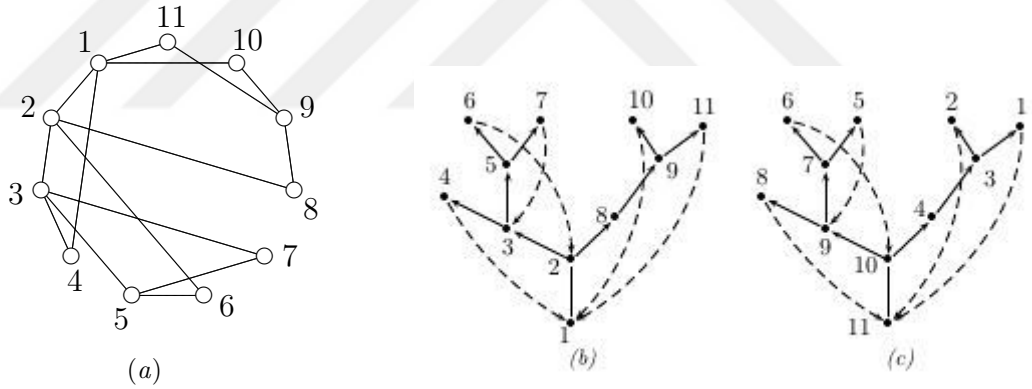


Figure 4.22: *(a) A Graph G; (b) Its DFS-tree; (c) Its reverse DFS-index*

If the set $B_{in}(v)$ is empty for some vertex v_i, then at iteration i there are no operations needed. An operation at iteration k is needed only if the vertex v_k has one or more incoming back edges. Consequently executing the algorithm on the graph in Figure 4.22, for instance, would perform zero operations at iterations 1 through 8, since $B_{in}(v_i) = \emptyset$ for $1 \leq i \leq 8$ where $i \in \mathbb{Z}$. This is so because at each iteration we embed incoming back edges. So if, for example, there are no back edges to embed at a certain iteration, then there are zero operations needed at that iteration, and we move on to the next iteration.

During the processing of a vertex v_i all back edges entering vertex v_k (provided v_k