Table of Contents

GRAPH THEORY
by
Ronald J. Gould
Emory University

Chapter 1
Graphs

Section 1.0 Introduction

For years, mathematicians have affected the growth and development of computer science. In the beginning they helped design computers for the express purpose of simplifying large mathematical computations. However, as the role of computers in our society changed, the needs of computer scientists began affecting the kind of mathematics being done.

Graph theory is a prime example of this change in thinking. Mathematicians study graphs because of their natural mathematical beauty, with relations to topology, algebra and matrix theory spurring their interest. Computer scientists also study graphs because of their many applications to computing, such as in data representation and network design. These applications have generated considerable interest in algorithms dealing with graphs and graph properties by both mathematicians and computer scientists.

Today, a study of graphs is not complete without at least an introduction to both theory and algorithms. This text will attempt to convince you that this is simply the nature of the subject and, in fact, the way it was meant to be treated.

Section 1.1 Fundamental Concepts and Notation

Graphs arise in many settings and are used to model a wide variety of situations. Perhaps the easiest way to adjust to this variety is to see several very different uses immediately. Initially, let's consider several problems and concentrate on finding models representing these problems, rather than worrying about their solutions.

Suppose that we are given a collection of intervals on the real line, say $C = \{ I_1, I_2, \ldots, I_k \}$. Any two of these intervals may or may not have a nonempty intersection. Suppose that we want a way to display the intersection relationship among these intervals. What form of model will easily display these intersections?

One possible model for representing these intersections is the following: Let each interval be represented by a circle and draw a line between two circles if, and only if, the intervals that correspond to these circles intersect. For example, consider the set

$$C = \{ [-4, 2], [0, 1], (-8, 2], [2, 4], [4, 10) \}.$$

The model for these intervals is shown in Figure 1.1.1.

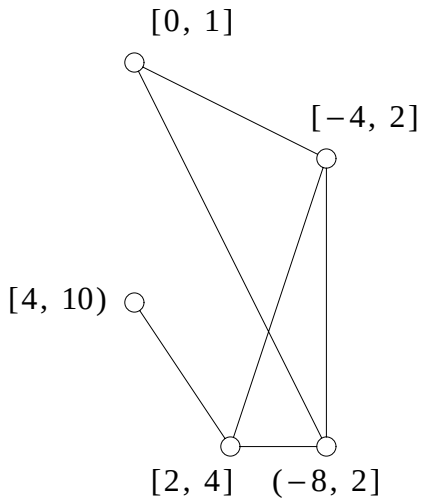


Figure 1.1.1. A model for the intersections of the members of C.

Next, we consider the following old puzzle. Suppose there are three houses (call them h_1, h_2 and h_3) and three utility companies (say gas (g), water (w) and electricity (e)). Our problem is to determine if it is possible to connect each of the three houses to each of the three utilities without crossing the service lines that run from the utilities to the houses. We model this puzzle by representing each house and each utility as a circle and drawing a line between two circles if there is a service line between the corresponding house and utility. We picture this situation in Figure 1.1.2. A solution to this problem would be a drawing in which no lines crossed. The drawing of Figure 1.1.2 is not a solution to the problem, but merely an attempt at modeling the problem.

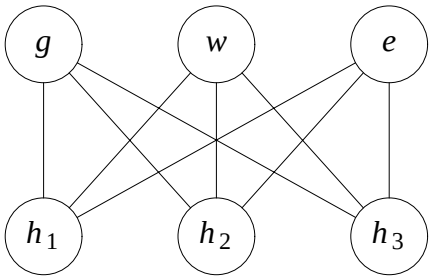


Figure 1.1.2. The three houses and three utilities model.

In our third problem, suppose you are the manager of a company that has four job openings (say j_1, j_2, j_3 and j_4) and five applicants $a_1, \ldots, a_5$ and that some of these applicants are qualified for more than one of your jobs. How do you go about choosing people to fill the jobs so that you will fill as many openings as possible? We picture such a situation in Figure 1.1.3. Again, each job and each applicant can be represented as a circle. This time, a line is drawn from a circle representing an applicant to each of the circles representing the jobs for which the applicant is qualified. A solution to this problem would be a set of four lines joining distinct jobs to distinct applicants, that is, one line joins each job to a distinct applicant. For example, the lines joining j_1 and a_2, j_2 and a_1, j_3 and a_4 and j_4 and a_5 constitute a solution to this problem. Since lines only join jobs to applicants, this is clearly the maximum number of lines possible. Can you find another solution? The real problem is how can we find solutions in general?

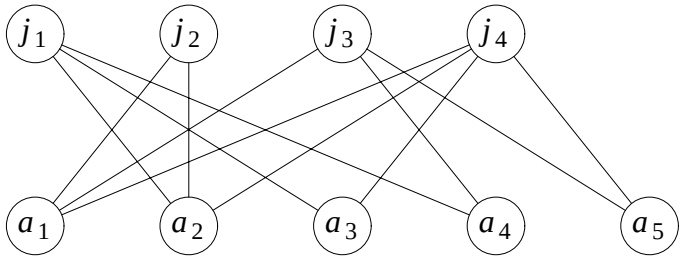


Figure 1.1.3. A job applicant model.

Despite the fact that these problems seem very different, we have used a similar type of diagram to model them. Such a diagram is called a graph. Formally, a *graph* $G = (V, E)$ is a finite nonempty set V of elements called *vertices*, together with a set E of two element subsets of V called *edges.* In our example diagrams, each circle is a vertex and each line joining two vertices is an edge. If the context is not clear, we will denote V or E by $V(G)$ or $E(G)$, respectively, to show they come from the graph G. In Figure 1.1.2, the vertices are h_1, h_2, h_3, g, w, e and the edges are

$$\{ h_1, g \}, \{ h_1, e \}, \{ h_1, w \}, \{ h_2, g \}$$
$$\{ h_2, e \}, \{ h_2, w \}, \{ h_3, e \}, \{ h_3, g \}, \{ h_3, w \}.$$

For simplicity, we will usually denote edges by consecutively listing the vertices at either end. For example, the edge $\{ h_1, g \}$ would be denoted $h_1 g$ or $g h_1$.

One of the beauties of graphs is that they may be thought of in many ways: formally as set systems, geometrically as the diagrams we have presented and algebraically, as we shall see later. Such diverse representations afford us an opportunity to use many tools in studying graphs and to apply graph models in many ways. To do this effectively, of course, we need to build more terminology and mathematical machinery.

Given a graph $G = (V, E)$, the number of vertices in V is called the *order of G* and the number of edges in E is called the *size of G*. They shall be denoted as $|V|$ and $|E|$, respectively. The interval graph of Figure 1.1.1 has order 5 and size 6. If a graph G has order p and size q, we say G is a (p, q) graph. Two vertices that are joined by an edge are said to be *adjacent,* as are two edges that meet at a vertex. If two vertices are not joined by an edge, we say they are *nonadjacent* or *independent.* Similarly, two edges that do not share a common vertex are said to be *independent.* The set of all vertices adjacent to a vertex v is called the *neighborhood of v* and is denoted $N(v)$. An edge between vertices u and v is said to have u (or v) as an *end vertex.* Further, the edge is said to be *incident* with v (or with u) and v is said to *dominate u* (also, u dominates v). The number of edges incident with a vertex v is called the *degree of v* and is denoted *deg v* or by $deg_G\ v$ if we wish to emphasize that this occurs in the graph G. The minimum degree and maximum degree of a vertex in the graph G are denoted by $\delta(G)$ and $\Delta(G)$, respectively. A graph in which each vertex has degree r is called an *r-regular* graph (or simply regular). We now present the theorem traditionally called The First Theorem of Graph Theory.

Theorem 1.1.1 Let G be a (p, q) graph and let $V = \{ v_1, v_2, \ldots, v_p \}$. Then $\sum_{i=1}^{p} deg\ v_i = 2q$. Consequently, any graph contains an even number of vertices of odd degree.

Proof. Since each edge has exactly two end vertices, the sum of the degrees counts each edge exactly twice. Thus, the sum is obtained. Since $2q$ is even, an even number of vertices of odd degree must then be present in the sum. □

We have considered three problems thus far. The drawing of Figure 1.1.1 is a solution to the first problem, and we have an idea of what a solution for one example of the third problem looks like. But the drawing given for the utilities problem does not provide a solution to that problem. This does not mean there is no solution, only that our drawing fails to provide one. What if we try other drawings (see Figure 1.1.4)? One of the interesting features of these drawings is the freedom we have to shape them. There are no restrictions on the size of the vertices or on the length or even the shape of the edges. These drawings are very much free-form. We are also free to choose an entirely different representation for our graph, for example the set representation we used in defining graphs. But this freedom also presents us with some difficulties. If a graph is presented in different ways, how can we determine if the presentations really represent the same graph?

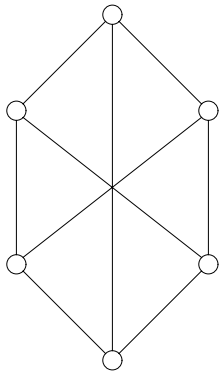


Figure 1.1.4. Another drawing of the house-utilities graph.

Mathematicians use the term *isomorphism* to mean the "fundamental equality" of two objects or systems. That is, the objects really have the same mathematical structure, only nonessential features like object names might be different. For graphs, "fundamentally equal" means the graphs have essentially the same adjacencies and nonadjacencies. To formalize this concept further, we say two graphs G_1 and G_2 are *isomorphic* if there exists a 1-1 and onto function $f : V(G_1) \rightarrow V(G_2)$ such that $xy \in E(G_1)$ if, and only if, $f(x)f(y) \in E(G_2)$ (that is, f preserves adjacency and nonadjacency). We use the function f to express the correspondence between vertices that are "essentially the same" in the two graphs. The function f is called an *isomorphism.*

Example 1.1.1 The two drawings of the house-utilities graph are again shown in Figure 1.1.5. An isomorphism between G_1 and G_2 is determined by the function $f : V(G_1) \rightarrow V(G_2)$ where:

$$f(a) = x, \quad f(b) = r, \quad f(c) = y,$$
$$f(d) = s, \quad f(e) = z, \quad f(g) = t.$$

An isomorphism from G_2 to G_1 is given by f^{-1}, the inverse of f.

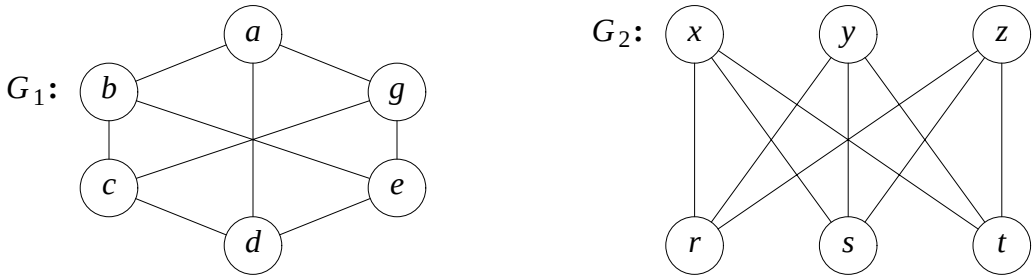


Figure 1.1.5. Two isomorphic graphs.

Can you find other isomorphisms from G_1 to G_2? □

A *subgraph of G* is any graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$; we also say *G contains H*. If H is a subgraph of G and $V(H) = V(G)$, we say that H is a *spanning subgraph* of G. A more restricted but often very useful idea is the following: Given a subset S of $V(G)$, the *subgraph induced by S*, denoted <S>, is that graph with vertex set S and edge set consisting of those edges of G incident with two vertices of S.

The graphs in Figure 1.1.6 illustrate these ideas. Since $V(H) = V(G)$, H is a spanning subgraph of G. Also, I is an induced subgraph of G since all edges of G with both end vertices in $V(I)$ are contained in I. However, J is not an induced subgraph of G since the edge from 1 to 5 is in G, but is not in J.

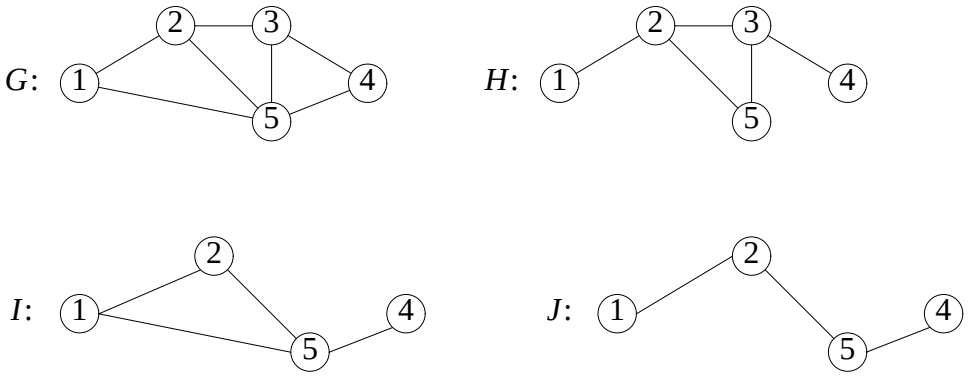
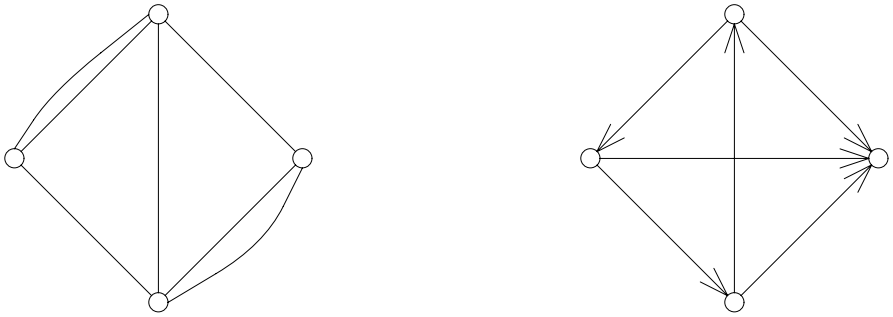


Figure 1.1.6. Spanning subgraph H, induced subgraph I, subgraph J of G.

Several natural and useful variations on graphs will be helpful. The first is the idea of a *multigraph,* that is, a graph with (possibly) multiple edges between vertices. A *pseudograph* allows edges that begin and end at the same vertex (called a *loop*). If we think of the edge between two vertices as an ordered pair rather than a set, a natural direction from the first vertex of the pair to the second can be associated with the edge. Such edges will be called *arcs* (to maintain the historical terminology), and graphs in which each edge has such a direction will be called *directed graphs* or *digraphs.* For digraphs, the number of arcs directed away from a vertex v is called the *outdegree* of v (denoted $od\ v$) and the number of arcs directed into a vertex v is the *indegree* of v (denoted $id\ v$). Often, for emphasis, we denote the arc directed from u to v as $u \to v$. In a digraph, we define the *degree* of a vertex v to be $deg\ v = id\ v + od\ v$. If $u \to v$ is an arc of the digraph, we say that *u dominates v* and that *v* is *dominated by u*. Sometimes we say *u* is *adjacent to v* or *v* is *adjacent from u*.

Clearly, we can produce even more variations such as pseudodigraphs, multidigraphs and pseudomultidigraphs. Although these will not play as significant a role in our study of graphs, at times they will be useful. In this text we will be sure the reader understands the kind of graph under consideration, and the term *graph* will always be as we defined it: finite order, without loops, multiple edges or directed edges.



multigraph digraph

Figure 1.1.7a. A multigraph and a digraph.

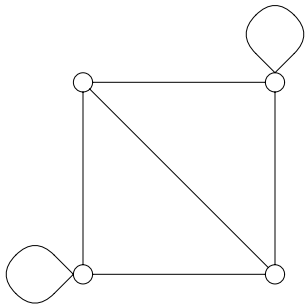


Figure 1.1.7b. A pseudograph.

Section 1.2 Elementary Properties and Operations

A quick inspection of a road map of the southern states shows several important cities and the interstates that connect them. We model a portion of this map in Figure 1.2.1.

It seems natural to think of a graph (or digraph) when trying to model a road system. Vertices represent cities, and edges (or arcs) represent the roads between the cities. In tracing the route from one location to another, we would traverse some sequence of roads, beginning at some starting point, and finally reaching our destination. For example, we could travel from Atlanta to Nashville by first leaving Atlanta and traveling along I75 to Chattanooga, then following I24 to Nashville. Such a model leads us naturally to formally define the following concepts.

Let x and y be two vertices of a graph G (not necessarily distinct vertices). An $x - y$ *walk* in G is a finite alternating sequence of vertices and edges that begins with the vertex x and ends with the vertex y and in which each edge in the sequence joins the vertex that precedes it in the sequence to the vertex that follows it in the sequence. For example, in the graph of Figure 1.2.1, one $b - n$ walk is

$$b, I59, c_2, I75, a, I20, b, I59, c_2, I24, n$$

while another is

$$b, I20, a, I85, c_1, I85, a, I75, c_2, I24, n.$$

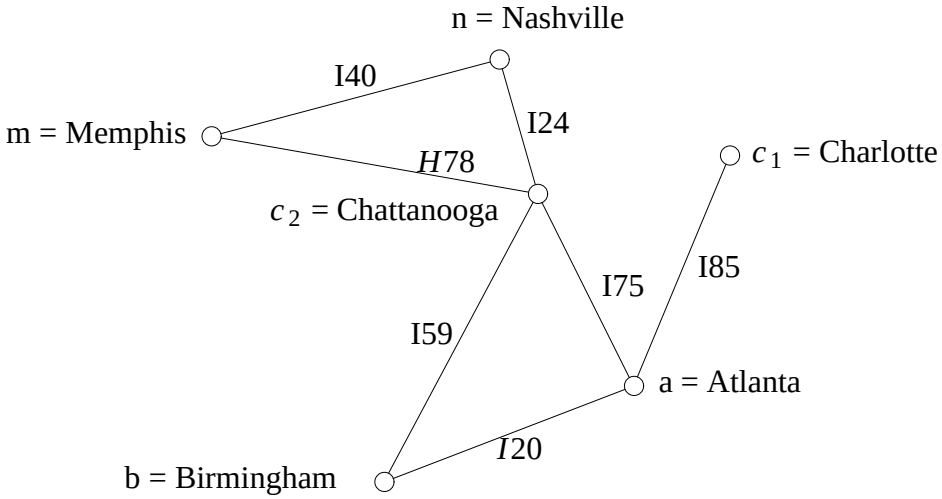


Figure 1.2.1. A model of roads between some southern cities.

The number of edges in a walk is called the *length* of the walk. Note that repetition of vertices and edges is allowed. Ordinarily, we will use a more compact notation for a walk by merely listing the vertices involved, noting that the edge between consecutive vertices (at least in a graph or digraph) is then implied. We will only use the full notation for walks in multigraphs, where there is a choice of edges and this choice is important, or for emphasis of the edges involved. An $x - y$ walk is *closed* if $x = y$ and *open* otherwise. Two walks are equal if the sequences of vertices and edges are identical.

Several stronger types of walks are also important. An $x - y$ *trail* is an $x - y$ walk in which no edge is repeated, and an $x - y$ *path* is an $x - y$ walk in which no vertex is repeated, except possibly the first and the last (if the path is closed). Clearly, a path is also a trail. We consider a single vertex as a trivial path (walk or trail). In the graph of Figure 1.2.1, we see that $a, b, a, c_2, b,$ is an $a - b$ walk of length 4, while a, b, c_2, a, c_1 is an $a - c_1$ trail of length 4, and a, c_2, n, m is an $a - m$ path of length 3.

It is clear that every path is a trail and every trail is a walk and that the converse of each of these statements fails to hold. However, we now present a useful result that relates walks and paths.

Theorem 1.2.1 In a graph G, every $x - y$ walk contains an $x - y$ path.

Proof. Let W be an $x - y$ walk in the graph G. Note that a vertex may receive more than one label if it occurs more than once in W. If no vertex is repeated, then W is

already a path; hence, we assume that at least one vertex is repeated in W. Let i and j be distinct integers with $i < j$ such that $v_i = v_j$. That is, the vertex v_i is repeated as v_j. If we now delete the vertices $v_i, v_{i+1}, \ldots, v_{j-1}$ from W, we obtain an $x - y$ walk W_1 which is shorter than W and has fewer repeated vertices. If W_1 is a path, we are done; if not, we repeat this process to obtain a new $x - y$ walk W_2. If W_2 is a path, we are done; otherwise, repeat this process. Since W is a finite sequence, eventually we must reach the stage where no vertices are repeated and an $x - y$ path is obtained. □

A closed trail is called a *circuit,* while a nontrivial circuit with no repeated vertices (other than the first and last) is called a *cycle.* We allow C_2 as a cycle, but note that it does not occur in graphs. The existence of C_2 is restricted to multigraphs. We do not consider C_1 (a single vertex) as a trivial cycle as this adds more complications than benefits. The *length* of a cycle (or circuit) is the number of edges in the cycle (or circuit). In the graph of Figure 1.2.3, w, x, y, z, w is a cycle of length 4; while t, r, u, t, s, v, t is a circuit of length 6. A graph of order n that consists of only a cycle (or path) is denoted C_n (or P_n) and is called simply an *n-cycle (or n-path).* If a graph contains no cycles it is termed *acyclic.* Cycles and paths are very important ideas and will be studied in much greater detail later. For now, they allow us to continue expanding our terminology. The graph of Figure 1.2.2 is an example of another special class of graphs called *complete graphs,* which contain an edge between all pairs of vertices. We denote the complete graph of order p as K_p.

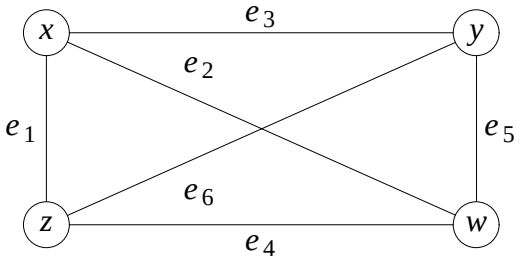


Figure 1.2.2. The complete graph K_4.

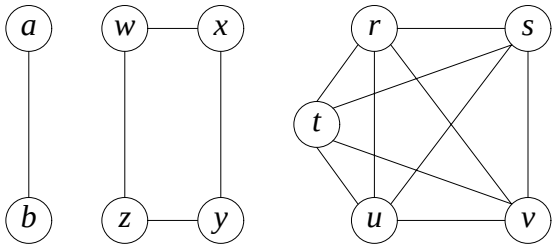


Figure 1.2.3. A disconnected graph *H*.

The graph *H* of Figure 1.2.3 is clearly different from those we considered earlier. For one thing, *H* consists of three "pieces." Each piece should be thought of as being "connected," but *H* should not. We can formalize this idea and obtain a useful way of describing "connected graphs" by using paths. We say a graph *G* is *connected* if there exists a path in *G* between any two of its vertices and *G* is *disconnected* otherwise. Clearly, there is no $a - z$ path in *H*, but there are paths between any two of w, x, y, and z. A *component* of a graph is a maximal connected subgraph. Thus, in this case *H* has three components, a P_2, a C_4 and a K_5.

Connectivity in digraphs is a little more interesting in that there are several possible kinds. A digraph *D* is said to be *strongly connected* (or *strong*) if for each vertex *v* of *D* there exists a directed path from *v* to any other vertex of *D*. We say *D* is *weakly connected* (or *weak*) if, when we remove the orientation from the arcs of *D*, a connected graph or multigraph remains (often, we say that the underlying graph is connected). Of course, *D* is *disconnected* if it is not at least weakly connected. For example, the digraph *E* of Figure 1.2.4 is clearly weakly connected. However, *E* is not strong since there is no directed path in *E* from *x* to any other vertex of *E*. The digraph *D* is also easily seen to be strong.

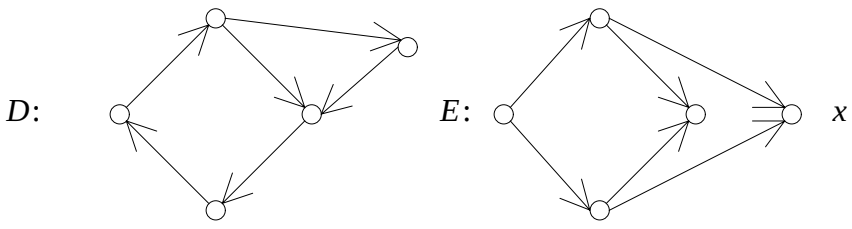


Figure 1.2.4. A strong digraph *D* and weak digraph *E*.

Combining several previous ideas, we define a *tree* to be a connected acyclic graph; a *forest* is an acyclic graph, that is, a graph each of whose components is a tree. (What else would a forest be?) In defining a forest as a collection of trees, what we have done is form a new graph from other graphs. In particular, the *union* of two graphs G_1 and G_2 (denoted $G_1 \cup G_2$) is that graph G with $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2)$. If we form the union of m isomorphic copies of the graph G, we denote the resulting graph as mG. As an example of this concept, the graph of Figure 1.2.3 can be denoted as $H = P_2 \cup C_4 \cup K_5$.

Graph union is an example of one graph operation used to form a new graph from other graphs. There are many other graph operations. Perhaps the simplest and most natural graph operation is the following: Consider a graph $G = (V, E)$ and form a new graph $\overline{G}$ where $V(\overline{G}) = V(G)$ and an edge $e \in E(\overline{G})$ if, and only if, e is not in $E(G)$. We call $\overline{G}$ the *complement of G*. In a sense, all we have done is remove all the edges from G and insert those edges that were originally missing from G. It should also be clear that the complement of $\overline{G}$ is the graph G itself.

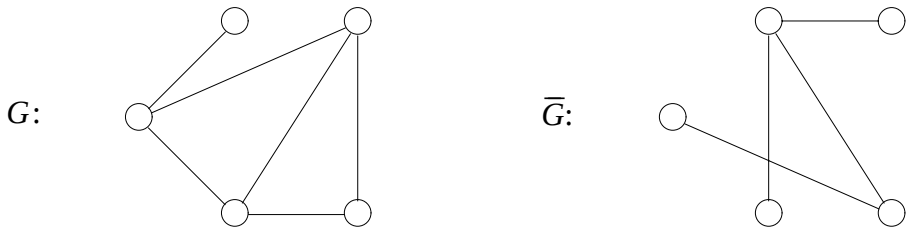


Figure 1.2.5. A graph G and its complement $\overline{G}$.

The *join* of two graphs G and H with disjoint vertex sets, denoted $G + H$, is the graph consisting of $G \cup H$ and all edges between vertices of G and vertices of H. Often, we will have occasion to consider the special case when $H = K_1$ and for simplicity, we will denote this as $G + x$, where K_1 is the vertex x. We define the *complete bipartite* graph $K_{m,n}$ to be the join $\overline{K}_m + \overline{K}_n$. The graph of Figure 1.1.2 is $K_{3,3}$. Complete bipartite graphs are a special case of an important class of graphs. We say a graph G is *bipartite* if it is possible to partition the vertex set V into two sets (called *partite sets*), say V_1 and V_2, such that each edge of G joins a vertex in V_1 to a vertex of V_2. Clearly complete bipartite graphs are bipartite, but complete bipartite graphs contain all possible edges between the partite sets. More generally, a graph G is *n-partite* if it is possible to partition the vertex set of G into n sets, such that any edge of G joins two vertices in different partite sets. Complete n-partite graphs have all possible edges between the partite sets. We denote the complete n-partite graph with partite sets of

order $p_1, p_2, \ldots, p_n$ as $K_{p_1, p_2, \ldots, p_n}$. Can you find a representation for $K_{p_1, p_2, \ldots, p_n}$ as the join of graphs?

On the other hand, at times we will also remove a set S of vertices from a graph G, along with all edges of G incident to a vertex in S. We denote the resulting graph as $G - S$. Again, if $S = \{ x \}$, we denote the resulting graph as $G - x$.

There are several graph operations that result in a graph whose vertex set is the cartesian product of the vertex sets of two graphs. The classic paper of Sabidussi [10] deals with several of these. We shall now consider some of these products.

The *cartesian product* of graphs G_1 and G_2, denoted $G_1 \times G_2$, is defined to be the graph with $V(G_1 \times G_2) = V(G_1) \times V(G_2)$, and two vertices $v = (v_1, v_2)$ and $w = (w_1, w_2)$ are adjacent in the cartesian product whenever $v_1 = w_1$ and v_2 is adjacent to w_2 in G_2 or symmetrically if $v_2 = w_2$ and v_1 is adjacent to w_1 in G_1. Can you show that $P_3 \times P_2$ is isomorphic to $P_2 \times P_3$? In general, is $G_1 \times G_2$ isomorphic to $G_2 \times G_1$?

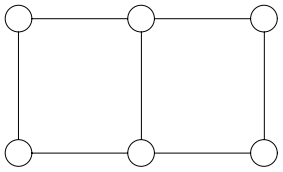


Figure 1.2.6. The cartesian product $P_2 \times P_3$.

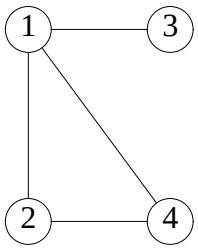
The *lexicographic product* (sometimes called the *composition*) of two graphs G_1 and G_2, denoted $G_1 [G_2]$, also has vertex set $V(G_1) \times V(G_2)$, while (v_1, v_2) is adjacent to (w_1, w_2) if, and only if, either v_1 is adjacent to w_1 in G_1 or $v_1 = w_1$ in G_1 and $v_2 w_2 \in E(G_2)$. In general, is $G_1 [G_2]$ isomorphic to $G_2 [G_1]$?

Figure 1.2.7. The lexicographic products $P_2[P_3]$ and $P_3[P_2]$.

Section 1.3 Alternate Representations for Graphs

Earlier, we considered two ways of representing graphs, as a set system (V, E) and pictorially. Now, we consider several other ways of viewing graphs.

Let $G = (V, E)$ be a (p, q)-graph. Consider the $p \times p$ matrix $A = [a_{ij}]$, where each row (and each column) of A corresponds to a distinct vertex of V. Let $a_{ij} = 1$ if vertex v_i is adjacent to vertex v_j in G and $a_{ij} = 0$ otherwise. Note that $a_{ii} = 0$ for each $i = 1, 2, \ldots, p$. This *adjacency matrix* of G is clearly a symmetric $(0, 1)$-matrix, with zeros down the main diagonal. The adjacency matrix clearly contains all the structural information about G and thus can be used as a representation for G. This representation has several advantages. First, $(0, 1)$-matrices are well-studied objects, and we gain all the power of linear algebra as a tool for studying graphs (although we shall not take full advantage of this here, the interested reader should see [1]). Second, the adjacency matrix representation is a very convenient one for storage within a computer.



$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 1.3.1. A graph and its adjacency matrix.

We now determine a method for finding the number of $x-y$ walks of a given length in a graph.

Theorem 1.3.1 If A is the adjacency matrix of a graph G with vertices $v_1, v_2, \ldots, v_p$, then the (i, j)-entry of A^n is the number of $v_i - v_j$ walks of length n in G.

Proof. We will apply induction on the length n. For $n = 1$, the result is obvious, as this is just the adjacency matrix itself. Now let $A^{n-1} = [a_{ij}^{n-1}]$ and assume that a_{ij}^{n-1} is the number of distinct $v_i - v_j$ walks of length $n - 1$ in G. Also, let $A^n = [\, a_{ij}^n \,]$. Since $A^n = A^{n-1} A$, we have that

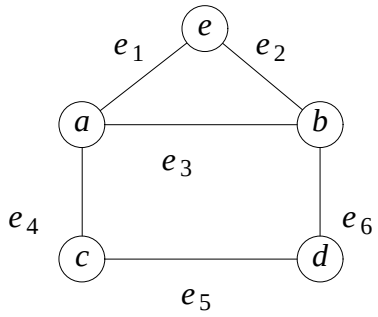
$$a_{ij}^{n} = \sum_{k=1}^{p} a_{ik}^{n-1}\, a_{kj}. \qquad 1.1$$

Every $v_i - v_j$ walk of length n in G must consist of a $v_i - v_k$ walk of length $n - 1$ followed by the edge from v_k to v_j and the vertex v_j. Thus, by induction and equation (1.1), the result follows. □

An idea similar to that of the adjacency matrix is the *incidence matrix.* For a (p, q) graph G, let the $p \times q$ matrix $M = [i_{xe}]$ be defined as follows: $i_{xe} = 1$ if vertex x is incident to edge e and $i_{xe} = 0$ otherwise. Thus, the rows of M correspond to the vertices of G and the columns correspond to the edges. It is easy to see that all the structure of G is contained in M; however, because M is not square, it lacks some of the power of adjacency matrices. Despite this shortcoming, incidence matrices have some valuable uses.

Still other, much simpler representations tend to be useful in computer applications. Probably the most commonly used form is merely to list each vertex in V along with those vertices that are adjacent to the listed vertex. This *adjacency list* contains all the structure of G, but has no extraneous information about nonadjacencies (like the zeros of the adjacency matrix). Nonadjacency is implied by omission from the list. Thus, when using an adjacency list, a program does not need to decide whether or not the next piece of information shows an edge or a nonedge, but rather it just retrieves the next adjacency. Tests with a number of algorithms have shown that adjacency lists will often speed computations. This is especially true in graphs that have many more nonadjacencies than adjacencies (called *sparse* graphs).

Example 1.3.1. The adjacency lists for the graph of Figure 1.3.2 are now shown.



a: b, c, e
b: a, d, e
c: a, d
d: b, c
e: a, b

Figure 1.3.2. A graph and its adjacency lists.

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{bmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 1.3.3. The incidence matrix for the graph of Figure 1.3.2.

Section 1.4 Algorithms

In order to deal with graphs efficiently and actually determine their properties on a computer, we need to develop processes that, when applied to a particular graph, will produce the desired answer. Algorithms are step-by-step procedures for solving problems. Usually, a graph problem will be posed in terms of several *parameters* (or variables). We then describe the problem at hand by giving a specification of the parameters involved and a statement about what constitutes a solution. An *instance* of a problem is obtained when we specify values for the parameters.

When dealing with graphs on a computer, we face several problems, including finding an algorithm that answers the question. But even then, there is no guarantee that we will be able to actually solve the problem. There are several other difficulties that must be faced. The first is the amount of space necessary for the information needed to store the description of the instance of our problem. Such a description might typically be in the form of the graph structure (adjacency matrices, adjacency lists, etc.). We may also need space to keep any partial results or necessary facts. As graphs grow large, this information can simply be too much to deal with. However, since computer memory has become cheaper and external storage (like tapes and discs) can be used, we shall proceed as though space is not the problem of fundamental concern, but rather that we can manage the *space requirements* of our problem.

The problem we will be more concerned with is the *time complexity*, that is, the relative time it will take us to perform the algorithm. We speak of relative time because it is impractical to try to determine the exact running times of algorithms. The computational speeds of machines differ, and the varying skills of programmers can also affect how well an algorithm seems to perform. These concerns make it unreasonable to try to measure time performance exactly. Instead, what we try to measure is the number

of computational steps involved in the algorithm. Since the exact steps we perform might well depend upon the graph being investigated (the particular instance of the problem), ordinarily our estimates are a worst-case measure of performance. Thus, an upper bound on the time complexity of the problem is usually obtained.

Consider as an example the problem of computing the square of the adjacency matrix A of a graph of order p. If we follow the typical rules for matrix multiplication, we perform p multiplications and $p - 1$ additions to compute each entry of A^2. Since there are p^2 entries in A^2, this means the algorithm really requires $p^2 \times (2p - 1) = 2p^3 - p^2$ operations to complete its task. Clearly, as p grows larger, the number of computations we must perform grows even faster.

In judging the quality of an algorithm, we try to measure how well it performs on arbitrary input. In doing this, we try to find an upper bound on the number of computational steps necessary to perform this algorithm. This entails finding some function that bounds the number of computational steps we must perform. As in the description of matrix multiplication, this is usually a function of the *size* of the problem. Here, *size* only refers to the amount of data in the instance of the problem at hand. Since it is clear that we must compute more if we have more data, it only makes sense for us to try to measure our work as a function of the problem size.

Returning to the problem of squaring the adjacency matrix A, if c_1 is the maximum amount of time required to multiply two numbers and c_2 is the maximum amount of time necessary to add two numbers, and c_3 is the maximum amount of time necessary to do all the other steps necessary for us to perform the algorithm (you can think of this as setup time to read in data, etc.), then we can bound the time, $T(p)$, it takes to square the adjacency matrix A (or, for that matter, to perform matrix multiplication of two $p \times p$ matrices) as:

$$T(p) \leq (c_1 p + c_2 (p - 1)) p^2 + c_3 = (c_1 + c_2) p^3 - c_2 p^2 + c_3$$

Thus, the amount of time it takes to square the adjacency matrix is bounded by a cubic function of p, the order of the graph. Since the constants are all dependent on the machines, languages, programmers and other factors outside our control, we simplify our description by saying that the problem of squaring the adjacency matrix of a graph of order p is *On The Order of* p^3, or $O(p^3)$ (read big oh of p^3). Hence, we say that the time complexity of the matrix multiplication algorithm is $O(p^3)$.

Formally, we say that $g(n) = O(f(n))$ if there exist constants k and m, such that $|g(n)| \leq k |f(n)|$ for all $n \geq m$. An algorithm has *polynomial time complexity* if its computational time complexity $T(n) = O(p(n))$ for some polynomial p in the input size n.

Table 1.4.1 compares the times for various bounding functions based on given input sizes. This table assumes that any operation requires .000001 seconds.

	problem size			
time	10	30	50	100
n	.00001 sec	.00003 sec	.00005 sec	.0001 sec
n^2	.0001 sec	.0009 sec	.0025 sec	.01 sec
n^5	.1 sec	24.3 sec	5.2 min	2.7 hrs
2^n	.001 sec	17.9 min	35.7 yrs	2^{48} cent
3^n	.059 sec	6.5 yrs	2 x 10^8 cent	3^{70} cent

Table 1.4.1 Comparisons of several time functions and data sizes.

It is clear that for the same problem size, the smaller the bounding function, the faster the algorithm is likely to run. That is, linear algorithms (those bounded by linear functions) should be "faster" than those with complexity $O(p^5)$, which in turn should be faster than those with complexity $O(p^{50})$ for similar data sets. It is also clear that for some functions (like 2^n), we have no hope of success, even with relatively small data sizes. For example, if the process under consideration has complexity $O(3^n)$, then we can see there is no real hope of success even for values as small as $n = 50$. We call any problem for which no polynomial algorithm can exist an *intractable problem.*

There is yet another kind of problem that at the present time sits between the polynomial problems (those with a known polynomial time solution) and the intractable problems. In order to consider such problems, we must consider *decision problems,* that is, problems that can be answered "yes" or "no". For example, one such decision problem is: Given two graphs G_1 and G_2, does G_1 contain a subgraph isomorphic to G_2?

A problem is said to be *in the class NP* if it can be solved by a nondeterministic polynomial algorithm. We can view such an algorithm as being composed of two parts. The first stage is the *guessing stage.* Here, given an instance *I,* the guessing stage selects some "structure" (graph, subgraph, vertex set, edge set, etc.) as a possible solution. Then, in the second stage, the *checking stage,* the structure is checked to see if it provides a yes or no answer to the decision problem. A nondeterministic algorithm solves a decision problem if it always correctly provides a yes or no answer for the guessed structure (in polynomial time). It should be clear that the set of all problems with polynomial solutions *P* is a subset of *NP*. It is not known if $P = NP$, although many believe that this

is not the case.

There is yet another special type of problem within *NP*, the so called *NP-complete* problems. These problems have been shown to be in a sense the hardest problems in *NP*. That is, a problem *X* is *NP*-complete if a solution for *X* provides a solution for all other problems *Y* in *NP*. By this we mean that there is a polynomial algorithm to "transform" *X* into *Y* (and, hence, to convert a solution of *X* into a solution of *Y*).

A rather interesting theory of computational complexity has arisen around these classes. It is not within the scope of this text to study this theory, but we do point out that there are many graph theoretic problems that are known to be in this elusive class of *NP*-complete problems. The interested reader should see the excellent text of Garey and Johnson [6].

Section 1.5 Degree Sequences

Given any graph, we can easily find the degree of each of its vertices. For example, the graph of Figure 1.3.2 has vertices of degree 2, 3, 3, 2 and 2. Each graph can be associated with such a unique sequence called its *degree sequence.* For convenience, these degrees can be ordered to form a nonincreasing sequence of nonnegative integers. In this case, the sequence 3, 3, 2, 2, 2 is formed. Several interesting questions about degree sequences now come to mind.

The first question you might think of is: Can we reverse this process? By this we mean, given a degree sequence *S*, can we determine a graph with *S* as its degree sequence? Perhaps a better first question is: Can we determine when a sequence of integers represents the degree sequence of a graph? A sequence is said to be *graphical* if it is the degree sequence of some graph. A graph *G* with degree sequence *S* is called a *realization* of *S*. Finally, given a sequence *S* that is graphical, with realization *G*, is *G* uniquely determined, or can there be several nonisomorphic realizations of *S*?

Let’s begin with the question: Are there some obvious restrictions on *S*? Certain conditions are clearly important. First, degrees are nonnegative integers; thus, all the terms of the sequence must be nonnegative integers. Next, if *S* has *p* terms, then no term can be larger than $p - 1$, because no vertex can be adjacent to more than the $p - 1$ other vertices. There are still other conditions that will eliminate some sequences.

Consider the sequence *S*: 1, 1, 1. For *S* to be the degree sequence of some graph, the graph must have exactly three vertices of degree one. But by Theorem 1.1.1, any graph must have an even number of vertices of odd degree, and therefore *S* cannot be a degree

sequence. Remember, Theorem 1.1.1 tells us that the sum of the degrees of the vertices in any graph must be an even number, and hence the sum of the terms of S must be even.

Next, we ask: If the sequence S is graphical, is the graph uniquely determined? That is, must there be only one graph (up to isomorphism) with S as its degree sequence? To answer this question, consider the sequence S: 2, 2, 2, 2, 2, 2. This sequence passes the first test in that all terms are nonnegative integers. It also passes the second test in that it contains only even valued entries. It is easy to find two nonisomorphic graphs that have degree sequence S. The graphs C_6 and $2C_3$ are two such graphs (see Figure 1.5.1). Thus, we have a negative answer to our question, that is, we see that degree sequences do not always provide enough information to uniquely describe a graph.

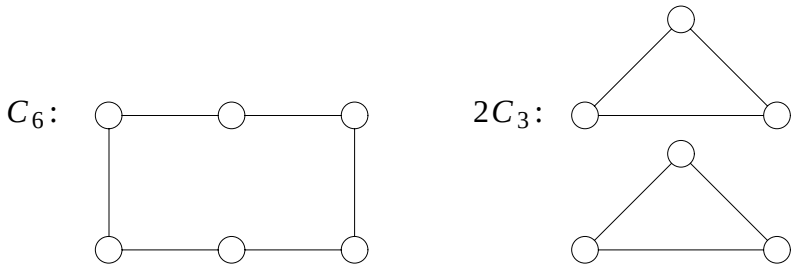


Figure 1.5.1. Two graphs with the same degree sequence.

The most important question raised earlier was: Can we determine when a sequence is graphical? The answer to our question was provided independently by Havel [8] and Hakimi [7].

Theorem 1.5.1 A nonincreasing sequence of nonnegative integers S: $d_1, d_2, \ldots, d_p$ $(p \geq 2,\ d_1 \geq 0)$ is graphical if, and only if, the sequence S_1 : $d_2 - 1, d_3 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_p$ is graphical.

Proof. Suppose that the sequence S_1 is graphical and let G_1 be a graph of order $p - 1$ with degree sequence S_1. The vertices of G_1 can be labeled as $x_2, x_3, \ldots, x_p$ in such a way that deg $x_i = d_i - 1$ if $2 \leq i \leq d_1 + 1$ and $deg\ x_i = d_i$ if $d_1 + 2 \leq i \leq p$. We can construct a new graph G with degree sequence S by inserting into G_1 a new vertex x_1 and the edges x_1x_j for $2 \leq j \leq d_1 + 1$. The degree of x_1 is d_1, and the degrees of the other vertices are now the remaining values of S. Thus, we have constructed a graph with degree sequence S, and so S is graphical.

Conversely, suppose that S is graphical and among all graphs with degree sequence S, let G be chosen with the following properties:

1. $V(G) = \{ x_1, x_2, \ldots, x_p \}$ and $\deg x_i = d_i, \quad i = 1, 2, \ldots, p.$
2. The sum of the degrees of the vertices adjacent to x_1 is a maximum.

Suppose that x_1 is not adjacent to vertices having degrees $d_2, d_3, \ldots, d_{d_1+1}$, that is, x_1 is not adjacent to the d_1 other vertices of largest degrees. Then there exist two vertices x_i and x_j with $d_j > d_i$ and such that x_i is adjacent to x_1 but x_j is not adjacent to x_1. Since $d_j > d_i$, there exists a vertex x_k such that x_k is adjacent to x_j but not to x_i. Now, removing the edges x_1x_i and x_jx_k and inserting x_1x_j and x_ix_k (see Figure 1.5.2) results in a new graph H with degree sequence S. However, in H the sum of the degrees of the vertices adjacent to x_1 is greater than in G, which contradicts property (2) in our choice of G. Thus, x_1 must be adjacent in G to the d_1 other vertices of largest degree. Now the graph $G - x_1$ has degree sequence S_1, and, hence, S_1 is graphical. □

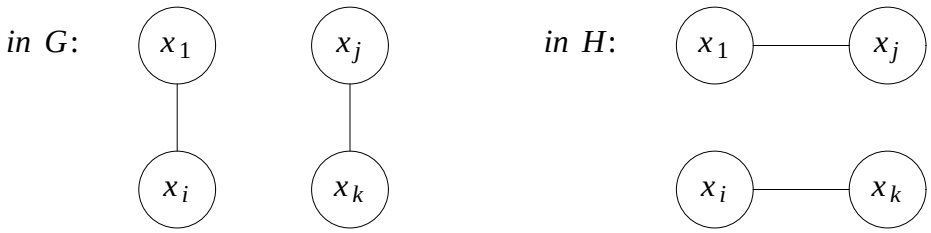


Figure 1.5.2. An edge interchange.

The fundamental step in the last proof was deleting the two edges x_1x_i and x_jx_k and inserting the edges x_1x_j and x_ix_k. This operation left the degrees of the vertices unchanged but varied the structure of the graph, and, it has come to be called an *edge interchange* (see Figure 1.5.2).

The proof of the last theorem essentially provides an algorithm for testing whether a sequence of nonnegative integers is graphical. We begin by applying our earlier tests for ruling out sequences, followed by the reduction from S to S_1. We continue to repeat this process until the sequence being tested fails or until a sequence of all zeros occurs. The sequence of t zeros is graphical because t vertices and no edges suffices. This is called the *empty graph.* We summarize these steps in the following algorithm.

Algorithm 1.5.1 Test for a Graphical Sequence.
Input: A sequence S of nonnegative integers of length p.
Output: Yes if the sequence is graphical, no otherwise.

1. If there exists an integer d in S such that $d > p - 1$, then halt and answer no.
2. If the sequence is all zeros, then halt and answer yes.
3. If the sequence contains a negative number, then halt and answer no.
4. Reorder the sequence (if necessary) so that it is nonincreasing.
5. Delete the first term d_1 from the sequence and subtract one from the next d_1 terms to form a new sequence. Go to step 2.

We can show that Algorithm 1.5.1 actually determines whether the sequence S is graphical. If the sequence halts before we apply step 5, then clearly a determination has been made in step 1, 2 or 3. Thus, we must show that after applying step 5, we will eventually produce a sequence of all zeros or a sequence that contains a negative term (since returning to step 2 means that only test 2 or 3 will halt the algorithm). By the time we reach step 5, we already know that S contains p terms and the largest is at most $p - 1$ (from step 1). In applying step 5, we produce a sequence of $p - 1$ terms with largest value at most $p - 2$. In general, if we apply step 5 a total of k times, we produce a sequence of $p - k$ terms with the largest entry at most $p - 1 - k$. If step 5 were actually applied $p - 1$ times, then the resulting sequence would be a single zero. Hence, we must eventually produce a sequence that has negative terms or we will surely produce a sequence of all zeros.

Once we have applied Algorithm 1.5.1 and determined that a particular sequence S is graphical, we can use the intermediate sequences we constructed to produce a graph with degree sequence S. Let's consider the following example.

Example 1.5.1. Construction of a graph from its intermediate degree sequences.

Suppose we begin with the sequence S_1: 5, 4, 4, 3, 2, 1, 1. Step 1 is satisfied, and we begin the loop of steps 2 – 5. The tests in steps 2 and 3 do not immediately halt us and repeating the loop of steps 2 – 5, we obtain the following collection of intermediate sequences:

$$S_1\text{: } 5,\ 4,\ 4,\ 3,\ 2,\ 1,\ 1$$
$$S_2\text{: } 3,\ 3,\ 2,\ 1,\ 1,\ 0$$
$$S_3\text{: } 2,\ 1,\ 1,\ 0,\ 0$$
$$S_4\text{: } 0,\ 0,\ 0,\ 0$$

Thus, we have determined by Algorithm 1.5.1 that S is graphical. To construct a graph with degree sequence S, we begin with the last sequence S_4. Clearly, S_4 is the

degree sequence of the empty graph on four vertices, say G_4. To proceed from S_4 to S_3, we must introduce a new vertex of degree 2 and produce two vertices of degree 1. That is, we simply undo step 5 of the algorithm. So insert a new vertex v_3 and make it adjacent to any two of the original vertices. Call the resulting graph G_3. Repeat this idea: Insert another vertex v_2 and join it to three vertices in G_3 to obtain the graph G_2. One final repetition of this process produces the graph G_1 with degree sequence S of Figure 1.5.3. □

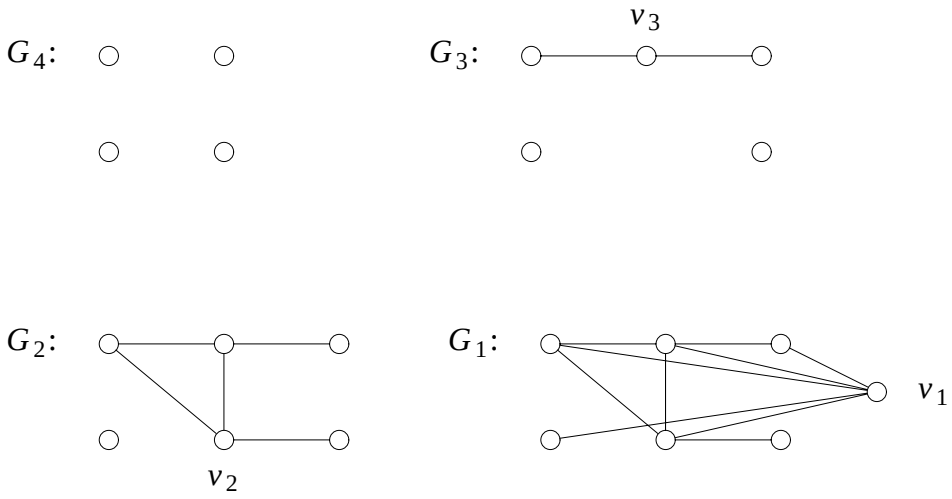


Figure 1.5.3. Reconstruction of a graph with degree sequence *S*.

An interesting result related to this construction and the proof of Theorem 1.5.1 was found independently by Eggleton [2] and by Fulkerson, Hoffman and McAndrew [5].

Theorem 1.5.2 Any realization of a graphical sequence can be obtained from any other realization by a finite sequence of edge interchanges.

An alternate result for testing graphical sequences is due to Erdös and Gallai [3].

Theorem 1.5.3 A nonincreasing sequence of nonnegative integers S: $d_1, d_2, \ldots, d_p$ $(p \geq 2)$ is graphical if, and only if, $\sum_{i=1}^{p} d_i$ is even and for each integer k, $1 \leq k \leq p - 1$, $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} \min\{k, d_i\}$.

Example 1.5.2. The Erdös and Gallai system of inequalities. Suppose we apply the last result to the sequence *S*: 5, 5, 5, 5, 2, 2, 2. The sum of the terms of *S* is even (26), and thus we must examine the system of inequalities.

1. For $k = 1$, $d_1 = 5 \leq 1(0) + \sum_{i=2}^{7} \min\{1, d_i\} = 6.$
2. For $k = 2$, $d_1 + d_2 = 10 \leq 2(1) + \sum_{i=3}^{7} \min\{2, d_i\} = 2 + 10 = 12.$
3. For $k = 3$, $\sum_{i=1}^{3} d_i = 15 \leq 3(2) + \sum_{i=4}^{7} \min\{3, d_i\} = 6 + 9 = 15.$
4. For $k = 4$, $\sum_{i=1}^{4} d_i = 20 > 4(3) + \sum_{i=5}^{7} \min\{4, d_i\} = 12 + 6 = 18.$

Thus, *S* cannot be graphical, because the required inequalities break down when $k = 4$.

We can see exactly why this happens if we think about the way edges must be distributed. When $k = 4$, the first four vertices (call them *U*) all have degree 5, and so we must determine if there is room to absorb all the edges leaving *U*. Even if <U> is complete, there must still be at least eight edges from <U> to the remaining three vertices. But these three vertices all are supposed to have degree 2; hence, they cannot handle the necessary edges. If we were to examine the earlier cases, we would see that numerically there was still a possibility of success. □

Let's consider a similar question for digraphs. First, we need to decide what information we want the degree sequence of a digraph to display. Should we show the outdegrees of the vertices or the indegrees? It actually seems reasonable to display both. Thus, we really want a sequence of ordered pairs, where the first entry of the pair is the indegree of the vertex and the second entry is the outdegree. A sequence S: $(i_1, o_1), (i_2, o_2), \ldots, (i_p, o_p)$ is called *digraphical* if it is the degree sequence of some digraph. Fulkerson [4] and Ryser [9] independently discovered a characterization of digraphical sequences that is reminiscent of our last result.

Theorem 1.5.4 A sequence $S : (i_1, o_1), (i_2, o_2), \ldots, (i_p, o_p)$ of ordered pairs of nonnegative integers with $i_1 \geq i_2 \geq \cdots \geq i_p$ is digraphical if, and only if, $i_k \leq p - 1$ and $o_k \leq p - 1$ for each k, and

$$\sum_{k=1}^{p} i_k = \sum_{k=1}^{p} o_k, \text{ and } \sum_{k=1}^{j} i_k \leq \sum_{k=1}^{j} \min\{ j - 1, o_k \} + \sum_{k=j+1}^{p} \min\{ j, o_k \}$$

for $1 \leq j < p$.

Section 1.6 Fundamental Counting

We have already seen several instances where it was necessary to determine how many possible items were involved in the problem under consideration. This is not unusual as we shall often find such counts to be very useful. The simplest and most useful law that aids our ability to count the number of objects in a complicated collection can be seen in the following idea: If we have "many" pigeons and "few" pigeon holes, then some pigeon hole will hold more than one pigeon. We can make this principle more precise with the aid of the following notation. By $\lfloor x \rfloor$ we mean the greatest integer less than or equal to x, and by $\lceil x \rceil$ we mean the least integer greater than or equal to x. Thus, for example, $\lfloor 3.71 \rfloor = 3$, and $\lceil 25.346 \rceil = 26$.

Theorem 1.6.1 (The Pigeon Hole Principle) If m pigeons are placed in k pigeon holes, then one hole will contain at least

$$\lceil m/k \rceil \text{ pigeons.}$$

Proof. Let x be the maximum number of pigeons in any one hole. Then $m \leq kx$, and so $x \geq \frac{m}{k}$. Since x is an integer, we see that $x \geq \lceil m/k \rceil$. □

We shall have occasion to use the Pigeon Hole Principle often. Although it seems obvious and its proof is very easy, it has many important consequences and generalizations. Certainly, its use is by no means restricted to graph theory.

As an illustration of the Pigeon Hole Principle, consider the following claim: In any group of six people, there are either three mutual acquaintances or three mutual nonacquaintances.

We can model this group of people using graphs. Represent each person as a vertex; two vertices are joined by an edge if, and only if, the corresponding people are acquaintances. What graph theoretic property will allow us to verify the claim? Three mutual acquaintances would induce a K_3 in the graph, while three mutual nonacquaintances would induce a $\overline{K}_3$. Thus, we can restate the claim as follows.

Theorem 1.6.2 Any graph on six vertices contains an induced K_3 or an induced $\overline{K}_3$ as a subgraph.

Proof. Let v be any vertex (that is, any person). By the Pigeon Hole Principle, of the remaining five vertices, either three are adjacent to v (acquaintances) or three are not adjacent to v (nonacquaintances). First, suppose that three are adjacent to v. If any two of these neighbors of v are themselves adjacent, then a K_3 is formed. If no pair of these three vertices are adjacent, then a $\overline{K}_3$ is formed by these vertices. A similar argument applies when we assume v has three vertices not adjacent to it, and so the result follows. □

Another example of a fundamental counting technique occurs when we are trying to count the number of available choices we have in performing some graph operation. For example, suppose we want to determine the number, $num(p, G)$, of labeled graphs on a set of p vertices. It is a straightforward matter to see that any one edge can be placed in this set of p vertices in $\binom{p}{2}$ different ways; we simply choose two of the p vertices as the end vertices of the edge. But in how many ways can we place k edges? Since there are $N = \binom{p}{2}$ possible ways to place an edge (the well-known binomial coefficient describing the number of ways of selecting two objects from p objects), there are $\binom{N}{k}$ ways of placing k edges in the p vertices. We can determine the total number of graphs on this set of vertices by summing these values over all possible sizes for our graph, that is:

$$num(p, G) = \sum_{q=0}^{N} \binom{N}{q}.$$

But it is a well-known property of the binomial coefficients that this sum equals 2^N. Hence, we have proved the following result.

Theorem 1.6.3 If $N = \binom{p}{2}$, then there are 2^N labeled graphs on p vertices.

We continue this approach with the following result.

Theorem 1.6.4 The number of subgraphs of K_n isomorphic to P_k is

$$\frac{n\,!}{2(n-k)\,!}.$$

Proof. Recall that P_k is a path on k vertices. Clearly, if $k > n$ there can be no such paths. Thus, suppose that $k \leq n$. If we begin at an arbitrary vertex, there are n choices for the first vertex. Clearly, there are $n - 1$ choices for the second vertex, $n - 2$ for the third vertex and so on. Thus, the number of choices is

$$n(n-1)(n-2)\cdots(n-(k-1)).$$

However, in counting the choices we have actually counted each path twice, since we could simply reverse the order of selection of the vertices and obtain the same path. Thus, the total number of subgraphs of K_n isomorphic to P_k is

$$\frac{n(n-1)\cdots(n-(k-1))}{2} = \frac{n\,!}{2(n-k)!}.\square$$

Exercises

1. Determine as many isomorphisms as you can between the graphs G_1 and G_2 of Figure 1.1.5.
2. Define the complement of G using set differences.
3. Represent $K_{p_1, p_2, \ldots, p_n}$ as the join of graphs.
4. Prove that $G_1 \times G_2$ is isomorphic to $G_2 \times G_1$.
5. Determine a result analogous to Theorem 1.3.1 for digraphs.
6. Give examples to show that there are walks that are not trails and trails that are not paths.
7. Given a (p_1, q_1) graph G_1 and a (p_2, q_2) graph G_2, determine formulas for the order and size of $\overline{G_1}$, $G_1 \cup G_2$, $G_1 \times G_2$ and $G_1\ [G_2]$.
8. Prove or disprove: The graph $G_1\ [G_2]$ is isomorphic to $G_2\ [G_1]$.
9. Prove that if two graphs are isomorphic, then they have the same order and size and degree sequence.
10. Find all nonisomorphic graphs of order 4.

11. Show that two graphs G and H are isomorphic if, and only if, there are two bijections (1-1 and onto functions) $f_1 : V(G) \to V(H)$ and $f_2 : E(G) \to E(H)$ such that $e = uv \in E(G)$ if, and only if, $f_2(e) = f_1(u)\ f_2(v)$.

12. Prove that a (p, q) graph G is a complete graph if, and only if, $q = \binom{p}{2}$.

13. Determine the order and size of $K_{p_1, p_2, \ldots, p_n}$, $n \geq 2$.

14. A (p, q) graph G is *self complementary* if G is isomorphic to $\overline{G}$. Show that if G is self complementary, then $p \equiv 0,1 (\bmod 4)$.

15. Suppose $\Delta(G) = k$. Prove that there exists a supergraph H of G (that is, a graph H that contains G as a subgraph) such that G is an induced subgraph of H and H is k-regular.

16. Prove that if G is a regular nonempty bipartite graph with partite sets V_1 and V_2, then $| V_1 | = | V_2 |$.

17. Determine all nonisomorphic digraphs of order 4.

18. Characterize the matrices that are adjacency matrices of digraphs, that is, those matrices $A(D) = [a_{ij}]$ where $a_{ij} = 1$ if $v_i \to v_j \in E(D)$ and $a_{ij} = 0$ otherwise.

19. Determine which of the following sequences is graphical, and for those that are graphical, find a realization of the sequence.
    a. 5, 5, 5, 3, 3, 2, 2, 2, 2, 2
    b. 7, 6, 5, 5, 4, 3, 2, 2, 2
    c. 4, 4, 3, 2, 1, 0

20. Show that the sequence $d_1, d_2, \ldots, d_p$ is graphical if, and only if, the sequence $p - d_1 - 1, p - d_2 - 1, \ldots, p - d_p - 1$ is graphical.

21. The *degree set* of a graph G is the set of degrees of the vertices of G.

    a. Show that every set $S = \{ a_1, a_2, \ldots, a_k \}$ $(k \geq 1)$ of positive integers with $a_1 < a_2 < \cdots < a_k$ is the degree set of some graph.

    b. Prove that if $u(S)$ is the minimum order of a graph with degree set S, then $u(S) = a_k + 1$.

    c. Find a graph of order 7 with degree set $S = \{ 3, 4, 5, 6 \}$.

22. Show that every graph of order n is isomorphic to a subgraph of K_n.

23. Show that every subgraph of a bipartite graph is bipartite.

24. Let G be a bipartite graph. If A_{21} is the transpose of A_{12}, show that the vertices of G can be partitioned so that the adjacency matrix of G has the following form:

$$\begin{bmatrix} 0 & A_{12} \\ A_{21} & 0 \end{bmatrix}$$

25. Let G be a (p, q) graph and let t be an integer, $1 < t < p - 1$. Prove that if $p \geq 4$ and all induced subgraphs of G on t vertices have the same size, then G is isomorphic to K_p or $\overline{K_p}$.

26. Let G be a (p, q) graph. Show that $\delta(G) \leq \dfrac{2q}{p} \leq \Delta(G)$.

27. Show that the entries on the diagonal of A^2 are the degrees of G.

28. Show that any degree sequence of a nontrivial graph has two equal terms.

29. Show that $d_1, d_2, \ldots, d_p$ is the degree sequence of a multigraph if and only if $\sum_{i=1}^{p} d_i$ is even and $d_1 \leq \sum_{i=2}^{p} d_i$.

30. The *line graph* $L(G)$ of a nonempty (p, q) graph G is that graph with $V(L(G)) = E(G)$ and such that two vertices in $L(G)$ are adjacent if, and only if, the corresponding edges in G are adjacent. If G has degree sequence $d_1, d_2, \ldots, d_p$, determine formulas for the order and size of $L(G)$.

31. Find $L(K_{2,3})$.

32. Assuming no human head has more than 2,000,000 hairs on it, show that there are at least two people in New York City with exactly the same number of hairs on their heads.

33. Show that if the digits $1, 2, \ldots, 10$ are used to randomly label the vertices of a C_{10} (no label is repeated), that the sum of the labels on some set of three consecutive vertices along the cycle will be at least 17.

34. Show that there exist graphs on five vertices that do not contain an induced K_3 or $\overline{K_3}$.

35. If we color the vertices of C_{11} either red, white, blue or green, what can be said about the order of the largest subgraph each of whose vertices has the same color?

36. Prove that in any group of $p \geq 2$ people, there are always two people that have the same number of acquaintances.
37. How many subgraphs isomorphic to $K_{1,3}$ are in K_n?
38. How many subgraphs isomorphic to C_t are in K_n?
39. How many subgraphs isomorphic to C_4 are in $K_{m,n}$?
40. How many subgraphs isomorphic to P_5 are in $K_{m,n}$?
41. If three men check their hats at a club, in how many ways can the hats be returned so that no man receives his own hat?
42. (*) Prove that any sequence of $n^2 + 1$ distinct integers contains either an increasing subsequence of $n + 1$ terms or a decreasing subsequence of $n + 1$ terms.
43. Find a sequence of n^2 ($n \geq 3$) distinct integers that does not contain an increasing subsequence of $n + 1$ terms or a decreasing subsequence of $n + 1$ terms.
44. (*) Prove that if $n + 1$ numbers are selected from the set $\{ 1, 2, \ldots, 2n \}$, then one of these numbers will divide a second one of these numbers.
45. (*) If every vertex of G has degree k, then
    a. k is an eigenvalue of G
    b. if G is connected, then the multiplicity of k is one,
    c. for any eigenvalue λ, $|\lambda| \leq k$.

References

1. Biggs, N. L., *Algebraic Graph Theory, 2nd Edition.* Cambridge University Press, London (1993).
2. Eggleton, R. B., Graphic Sequences and Graphic Polynomials. A report in: *Infinite and Finite Sets,* Vol. 1, ed. by A. Hajnal. Colloq. Math. Soc. J. Bolyai 10 (North Holland, Amsterdam, 1975), 385 – 392.
3. Erdös, P., and Gallai, T., Graphs with Prescribed Degrees of Vertices (Hungarian). *Mat. Lapok* 11(1960) 264 – 274.

4. Fulkerson, D. R., Upsets in Round Robin Tournaments. *Canad. J. Math.*, 17(1965), 957 – 969.

5. Fulkerson, D. R., Hoffman, A. J., and McAndrew, M. H., Some Properties of Graphs with Multiple Edges. *Canad. J. Math.*, 17(1965), 166 – 177.

6. Garey, M. R., and Johnson, D. S., *Computers and Intractability, A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

7. Hakimi, S. L., On the Realization of a Set of Integers as Degrees of the Vertices of a Graph. *J. SIAM Appl. Math.*, 10(1962), 496 – 506.

8. Havel, V., A Remark on the Existence of Finite Graphs (Czech.), *Casopis Pest. Mat.*, 80(1955), 477 – 480.

9. Ryser, H. J., Combinatorial Properties of Matrices of Zeros and Ones. *Canad. J. Math.*, 9(1957), 371 – 377.

10. Sabidussi, G., Graph Multiplication. *Math. Z.*, 72(1960), 446 – 457.

Chapter 2
Paths and Searching

Section 2.1 Distance

Almost every day you face a problem: You must leave your home and go to school. If you are like me, you are usually a little late, so you want to take the shortest route. How do you find such a route? Figure 2.1.1 models one such situation. Suppose vertex *h* represents your home, vertex *s* represents school and the other vertices represent intersections of the roads between home and school. Each edge represents a road and is labeled with its length (distance).

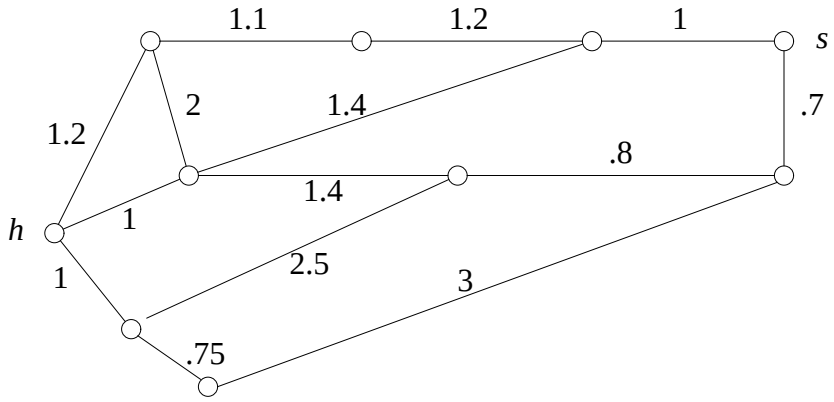


Figure 2.1.1. The model of routes to school.

Earlier, we discussed the idea of paths and of walking from one vertex to another. We also have seen that there may be many different paths between two vertices. I'm sure that ordinarily you feel it is best to take the shortest route between two places when you travel, and we are no different when it comes to graphs. But how do we measure the distance between two vertices in a graph? Certainly, following paths seems more efficient than following walks or trails since with paths we do not repeat any edges or vertices. Recall that we defined the length of a path to be the number of edges in the path. However, this definition treats each edge equally, as if each edge had length 1. This is inaccurate when the graph is modeling a road system, as in the problem just discussed. In order to adjust to more situations, we shall simply think of each edge as having a nonnegative label representing its length, and unless otherwise stated, that length will be 1. The *length of a path*, then, becomes the sum of the lengths of the edges in that path. The idea of distance stays the same no matter what the label; we merely let the *distance from x to y*, denoted $d(x, y)$, be the minimum length of an $x - y$ path in the

graph. We now have a very general and flexible idea of distance that applies in many settings and is useful in many applications.

But how do we go about finding the distance between two vertices? What do we do to find the distance between all pairs of vertices? What properties does distance have? Does graph distance behave as one usually expects distance to behave? For that matter, how does one expect distance functions to behave? Let's try to answer these questions.

We restrict the length of any edge to a positive number, since this corresponds to our intuition of what length should be. In doing this, we can show that the following properties hold for the distance function d on a graph G (see exercises):

1. $d(x, y) \geq 0$ and $d(x, y) = 0$ if, and only if, $x = y$.
2. $d(x, y) = d(y, x)$.
3. $d(x, y) + d(y, z) \geq d(x, z)$.

These three properties define what is normally called a *metric function* (or simply a metric) on the vertex set of a graph. Metrics are well-behaved functions that reflect the three properties usually felt to be fundamental to distance (namely, properties $1-3$). Does each of these properties also hold for digraphs?

The *diameter*, denoted $diam(G)$, of a connected graph G equals $\max_{u \in V} \max_{v \in V} d(u, v)$, while the *radius* of G, denoted $rad(G)$, equals $\min_{u \in V} \max_{v \in V} d(u, v)$. Theorem 2.1.1 shows that these terms are related in a manner that is also consistent with our intuition about distance.

Example 2.1.1. We find the value of $d(x, y)$ for each pair of vertices x, y in the graph of Figure 2.1.2. These distances are shown in Table 2.1.1.

$d(x, y)$	a	b	c	d	e
a	0	1	2	1	3
b	1	0	1	2	2
c	2	1	0	1	1
d	1	2	1	0	2
e	3	2	1	2	0

Table 2.1.1 Table of distances in the graph of Figure 2.1.2.

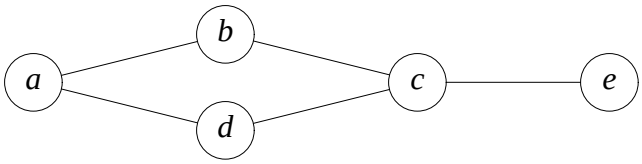


Figure 2.1.2. A graph of diameter 3 and radius 2.

Theorem 2.1.1 For any connected graph G,

$$rad(G) \leq diam(G) \leq 2rad(G).$$

Proof. The first inequality comes directly from our definitions of radius and diameter. To prove the second inequality, let $x, y \in V(G)$ such that $d(x, y) = diam(G)$. Further, let z be chosen so that the longest distance path from z has length $rad(G)$. Since distance is a metric, by property (3) we have that

$$diam(G) = d(x, y) \leq d(x, z) + d(z, y) \leq 2\ rad(G).\square$$

Another interesting application of distance occurs when you try to preserve distance under a mapping from one graph to another. A connected graph H is *isometric from* a connected graph G if for each vertex x in G, there is a 1-1 and onto function $F_x : V(G) \rightarrow V(H)$ that preserves distances from x, that is, such that $d_G(x, y) = d_H(F_x(x), F_x(y))$.

Example 2.1.2. The graph G_2 is isometric from G_1 (see Figure 2.1.3). The following mappings show that G_2 is isometric from G_1:

F_u: $u \rightarrow d$, $w \rightarrow c$, $x \rightarrow a$, $v \rightarrow b$
F_w: $u \rightarrow a$, $w \rightarrow c$, $x \rightarrow d$, $v \rightarrow b$
$F_x = F_w$
F_v: $u \rightarrow a$, $w \rightarrow b$, $x \rightarrow d$, $v \rightarrow c$. □

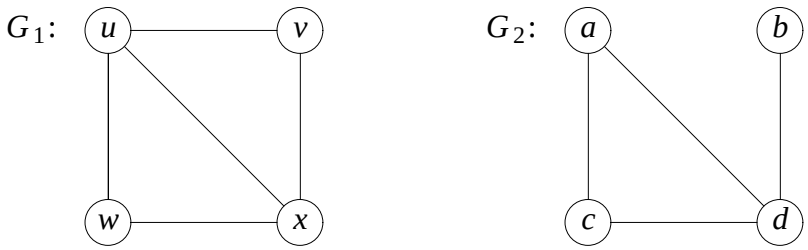


Figure 2.1.3. The graph G_2 is isometric from G_1 but not conversely.

Theorem 2.1.2 The relation isometric from is not symmetric, that is, if G_2 is isometric from G_1, then G_1 need not be isometric from G_2.

Proof. Graphs G_1 and G_2 of Figure 2.1.3 show a graph G_2 that is isometric from G_1.

To see that G_1 is not isometric from G_2, we note that each mapping F_i in the definition must preserve distances from the vertex *i* and there can be no mapping F_b with this property, because *b* has only one vertex at distance 1 from itself. □

We wish to develop efficient algorithms for computing distances in graphs. In order to do this, we begin by considering the simplest case, graphs with all edges of length 1. The first algorithm we present is from Moore [9] and is commonly called the *breadth-first search algorithm* or *BFS.* The idea behind a breadth-first search is fairly simple and straightforward. Beginning at some vertex *x*, we visit each vertex dominated by (adjacent from) *x*. Then, in turn, from each of these vertices, we visit any vertex that has not yet been visited and is dominated by the vertex we are presently visiting. We continue in this fashion until we have reached all vertices possible.

Algorithm 2.1.1 Breadth-First Search.
Input: An unlabeled graph $G = (V, E)$ with distinguished vertex *x*.
Output: The distances from *x* to all vertices reachable from *x*.
Method: Use a variable *i* to measure the distance from *x*, and label vertices with *i* as their distance is found.

1. $i \leftarrow 0$.

2. Label x with "i."
3. Find all unlabeled vertices adjacent to at least one vertex with label i. If none is found, stop because we have reached all possible vertices.
4. Label all vertices found in step 3 with $i+1$.
5. Let $i \leftarrow i + 1$, and go to step 3.

Example 2.1.3. As an example of the BFS algorithm, consider the graph of Figure 2.1.2. If we begin our search at the vertex d, then the BFS algorithm will proceed as follows:

1. Set $i = 0$.
2. Label d with 0.
3. Find all unlabeled vertices adjacent to d, namely a and c.
4. Label a and c with 1.
5. Set $i = 1$ and go to step 3.
3. Find all unlabeled vertices adjacent to one labeled 1, namely b and e.
4. Label b and e with 2.
5. Set $i = 2$ and go to step 3.
3. There are no unlabeled vertices adjacent to one labeled 2; hence, we stop. □

In essence, we can view the search as producing a *search tree*, using some edge to reach each new vertex along a path from x. In Figure 2.1.4 we picture two possible search trees for the previous example.

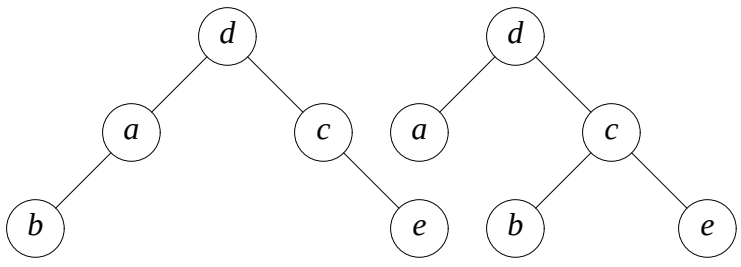


Figure 2.1.4. Two BFS search trees in the graph of Figure 2.1.2.

Theorem 2.1.3 When the BFS algorithm halts, each vertex reachable from x is labeled with its distance from x.

Proof. Suppose vertex $v = v_k$ has been labeled k. Then, by BFS (steps 3 and 4), there must exist a vertex v_{k-1}, which is labeled $k - 1$ and is adjacent to v_k, and similarly a vertex v_{k-2}, which is labeled $k - 2$ and is adjacent to v_{k-1}. By repeating this argument, we eventually reach v_0, and we see that $v_0 = x$, because x is the only vertex labeled zero. Then

$$x = v_0, v_1, \ldots, v_{k-1}, v_k = v$$

is a path of length k from x to v. Hence, $d(x, v) \leq k$.

In order to prove that the label on v is the distance from x to v, we apply induction on k, the distance from x to v. In step 2 of BFS, we label x with 0, and clearly $d(x, x) = 0$. Now, assume the result holds for vertices with labels less than k and let P: $x = v_0, v_1, \ldots, v_k = v$ be a shortest path from x to v in G. By the inductive hypothesis, $v_0, v_1, \ldots, v_{k-1}$ is a shortest path from x to v_{k-1}. By the inductive hypothesis, v_{k-1} is labeled $k - 1$. By the algorithm, when $i = k - 1$, v receives the label k. To see that v could not have been labeled earlier, suppose that indeed it had been labeled with $h < k$. Then there would be an $x - v$ path shorter than P, contradicting our choice of P. Hence, the result follows by induction. □

When the BFS algorithm is performed, any edge in the graph is examined at most two times, once from each of its end vertices. Thus, in step 3 of the BFS algorithm, the vertices labeled i are examined for unvisited neighbors. Using incidence lists for the data, the BFS algorithm has time complexity $O(|E|)$. In order to obtain the distances between any two vertices in the graph, we can perform the BFS algorithm, starting at each vertex. Thus, to find all distances, the algorithm has time complexity $O(|V|\ |E|)$. How does the BFS algorithm change for digraphs? Can you determine the time complexity for the directed version of BFS? Can you modify the BFS labeling process to make it easier to find the $x - v$ distance path?

Next, we want to consider arbitrarily labeled (sometimes these labels are called *weights*) digraphs, that is, digraphs with arcs labeled $l(e) \geq 0$. These labels could represent the length of the arc e, as in our home to school example, or the cost of traveling that route, or the cost of transmitting information between those locations, or transmission times, or any of many other possibilities.

When we wish to determine the shortest path from vertex u to vertex v, it is clear that we must first gain information about distances to intermediate vertices. This information

is often recorded as a label assigned to the intermediate vertex. The label at intermediate vertex w usually takes one of two forms, the distance $d(u, w)$ between u and w, or sometimes the pair $d(u, w)$ and the predecessor of w on this path, $pred(w)$. The predecessor aids in backtracking to find the actual path.

Many distance algorithms have been proposed and most can be classified as one of two types, based upon how many times the vertex labels are updated (see [6]). In *label-setting* methods, during each pass through the vertices, one vertex label is assigned a value which remains unchanged thereafter. In *label-correcting* methods, any label may be changed during processing. These methods have different limitations. Label-setting methods cannot deal with graphs having negative arc labels. Label-correcting methods can handle negative arc labels, provided no negative cycles exist, that is, a cycle with edge weight sum a negative value.

Most label-setting or correcting algorithms can be recast into the same basic form which allows for finding the shortest paths from one vertex to all other vertices. Often what distinguishes these algorithms is how they select the next vertex from the candidate list C of vertices to examine. We now present a generic distance algorithm.

Algorithm 2.1.2a Generic Distance Algorithm.
Input: A labeled digraph $D = (V, E)$ with initial vertex v_1.
Output: The distance from v_1 to all other vertices.
Method: Generic labeling of vertices with label $(L(v), pred(v))$.

1. For all $v \in V(D)$ set $L(v) \leftarrow \infty$.
2. Initialize $C =$ the set of vertices to be checked.
3. While $C \neq \emptyset$;
   Select $v \in C$ and set $C = C - v$.
   For all u adjacent from v;
   If $L(u) > L(v) + l(vu)$;
   $L(u) = L(v) + l(vu)$;
   $pred(u) = v$;
   Add u to C if it is not there.

One of the earliest label-setting algorithms was given by Dijkstra [1]. In Dijkstra's algorithm, a number of paths from vertex v_1 are tried and each time the shortest among them is chosen. Since paths can lead to new vertices with potentially many outgoing arcs, the number of paths can increase as we go. Each vertex is tried once, all paths leading from it are added to the list and the vertex itself is labeled and no longer used (label-setting). After all vertices are visited the algorithm is finished. At any time during

execution of the algorithm, the value of $L(v)$ attached to the vertex v is the length of the shortest $v_1 - v$ path presently known.

Algorithm 2.1.2 Dijkstra's Distance Algorithm.
Input: A labeled digraph $D = (V, E)$ with initial vertex v_1.
Output: The distance from v_1 to all vertices.
Method: Label each vertex v with $(L(v), pred(v))$, which is the length of a shortest path from v_1 to v that has been found at that instant and the predecessor of v along that path.

1. $L(v_1) \leftarrow 0$ and for all $v \neq v_1$ set $L(v) \leftarrow \infty$ and $C \leftarrow V$.
2. While $C \neq \emptyset$;
   Find $v \in C$ with minimum label $L(v)$.
   $C \leftarrow C - \{ v \}$
   For every $e = v \rightarrow w$,
   if $w \in C$ and $L(w) > L(v) + l(e)$ then
   $L(w) \leftarrow L(v) + l(e)$ and $pred(w) = v$.

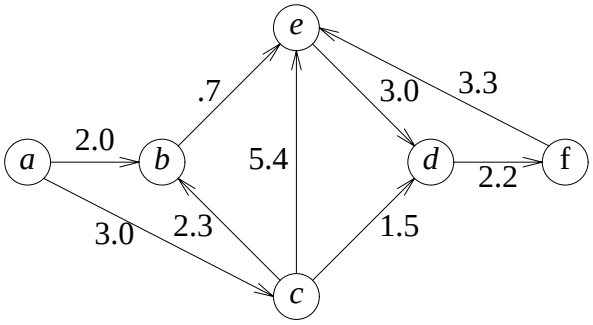


Figure 2.1.5. A digraph to test Dijkstra's algorithm.

Example 2.1.4. We now apply Dijkstra's algorithm to the digraph of Figure 2.1.5, finding the distances from vertex a. The steps performed and the actions taken are shown in the following table.

iteration	vertex	distances currently to vertices					
		a	b	c	d	e	f
initial		0	∞	∞	∞	∞	∞
1	a		2	3	∞	∞	∞
2	b			3	∞	2.7	∞
3	e			3	5.7		∞
4	c				4.5		∞
5	d						6.7
6	f						

Table 2.1.2 Distances from vertex *a* via Dijkstra's Algorithm.

We next verify that upon completion, Dijkstra's algorithm has labeled the vertices with their proper distances from *x*.

Theorem 2.1.4 If $L(v)$ is finite when Algorithm 2.1.2 halts, then $d(x, v) = L(v)$.

Proof. First we show that if $L(v)$ is finite, then there is a $v_1 - v$ path of length $L(v)$ in D. Since $L(v)$ is finite, its value must have changed during step 2 of the algorithm. Let u be the vertex used to label v, that is, $e = u \to v$ and $L(v) = L(u) + l(e)$. During this step, we delete u from C, and, hence, its label can never be changed again.

Now find the vertex w used to label u (this is just $pred(u)$) and again repeat this backtracking search. Eventually, we will backtrack to v_1, and we will have found a $v_1 - v$ path whose length is exactly $L(v)$. The backtracking search finds each time a vertex was deleted from C, and, thus, no vertex on the path can be repeated. Therefore, the path must backtrack to v_1, which has label 0.

Next, to see that $L(v) = d(v_1, v)$, we proceed by induction on the order in which we delete vertices from C. Clearly, v_1 is the first vertex deleted, and $L(v_1) = 0 = d(v_1, v_1)$. Next, assume that $L(u) = d(v_1, u)$ for all vertices u deleted from C before v.

If $L(v) = \infty$, let w be the first vertex chosen with an infinite label. Clearly, for every vertex v remaining in C, $L(v) = \infty$ and for all $u \in V - C$ (that is, those already deleted from C), $L(u)$ must be finite. Therefore, there are no arcs from $V - C$ to C, and since $v_1 \in V - C$ and $w \in C$, there are no $v_1 - w$ paths.

Next, suppose $L(v)$ is finite. We already know there exists a $v_1 - v$ path P of length $L(v)$; hence, it must be the case that $L(v) \geq d(v_1, v)$. In order to see that P is a shortest

path, suppose that

$$P_1 : v_1, v_2, \ldots, v_k = v$$

is a shortest $v_1 - v$ path and let $e_i = v_i \rightarrow v_{i+1}$. Then

$$d(v_1, v_i) = \sum_{j=1}^{i-1} l(e_j).$$

Suppose v_i is the vertex of highest subscript on P_1 deleted from C before v. By the inductive hypothesis

$$L(v_i) = d(v_1, v_i) = \sum_{j=1}^{i-1} l(e_j).$$

If $v_{i+1} \neq v$, then $L(v_{i+1}) \leq L(v_i) + l(e_i)$ after v_i is deleted from C. Since step 2 can only decrease labels, when v is chosen, $L(v_{i+1})$ still satisfies the inequality. Thus,

$$\begin{aligned} L(v_{i+1}) &\leq L(v_i) + l(e_i) \\ &= d(v_1, v_i) + l(e_i) \\ &= d(v_1, v_{i+1}) \leq d(v_1, v). \end{aligned}$$

If $d(v_1, v) < L(v)$, then v should not have been chosen. If $v_{i+1} = v$, the same argument shows that $L(v) \leq d(v_1, v)$, which completes the proof. □

We now determine the time complexity of Dijkstra's algorithm. Note that in step 2, the minimum label of C must be found. This can certainly be done in $|C| - 1$ comparisons. Initially, $C = V$, and step 4 reduces C one vertex at a time. Thus, the process is repeated $|V|$ times. The time required in step 2 is then on the order of $\sum_{i=1}^{|V|} i$ and therefore is $O(|V|^2)$.

Step 2 uses each arc once at most, so it requires at most $O(|E|) = O(|V|^2)$ time. The entire algorithm thus has time complexity $O(|V|^2)$. If we want to obtain the distance between any two vertices, we can use this algorithm once with each vertex playing the role of the initial vertex x. This process requires $O(|V|^3)$ time.

Can you modify Dijkstra's algorithm to work on undirected graphs?

What can we do to further generalize the problems we have been considering? One possibility is to relax our idea of what the edge labels represent. If we consider these labels as representing some general value and not merely distance, then we can permit these labels to be negative. We call these generalized labels *weights,* and we denote them as $w(e)$. The "distance" between two vertices x and y will now correspond to the minimum sum of the edge weights along any $x - y$ path.

We noted earlier that label-setting methods failed when negative arc labels were allowed. To see why this happens in Dijkstra's Algorithm, consider the example below.

Example 2.1.5

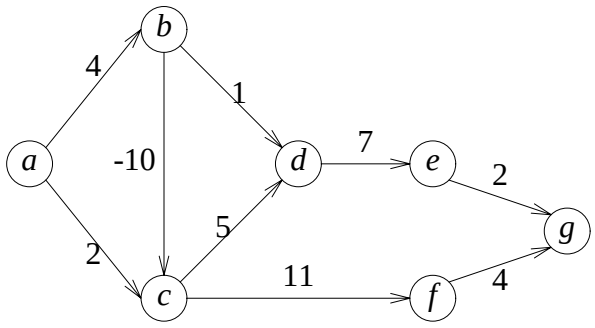


Figure 2.1.6. A digraph with negative arc weights allowed.

The chart below shows the changes made to the labels during the execution of Dijkstra's algorithm on the digraph of Figure 2.1.6. As you can see, the label set at 2 for vertex *c* is never corrected to it's proper value. The fact it can actually decrease because of negative weights is the problem.

iteration	vertex	current distances of vertices						
		a	b	c	d	e	f	g
init		0	∞	∞	∞	∞	∞	∞
1	a		4	2	∞	∞	∞	∞
2	c		4		7	∞	13	∞
3	b				5	∞	13	∞
4	d					12	13	∞
5	e						13	14
6	f							14
7	g							

However, the path a, b, c, d, e, g is never explored and has distance 8. □

Unfortunately, Dijkstra's algorithm can fail if we allow the edge weights to be negative. However, Ford [4, 5] gave an algorithm for finding the distance between a

distinguished vertex x and all other vertices when we allow negative weights. This algorithm features a label correcting method to record the distances found.

Initially, x is labeled 0 and every other vertex is labeled ∞. Ford's algorithm, then, successively refines the labels assigned to the vertices, as long as improvements can be made. Arcs are used to decrease the labels of the vertices they reach. There is a problem, however, when the digraph contains a cycle whose total length is negative (called a *negative cycle*). In this case, the algorithm continually traverses the negative cycle, decreasing the vertex labels and never halting. Thus, in order to properly use Ford's algorithm, we must restrict its application to digraphs without negative cycles.

Algorithm 2.1.3 Ford's Distance Algorithm.
Input: A digraph with (possibly negative) arc weights $w(e)$, but no negative cycles.
Output: The distance from x to all vertices reachable from x.
Method: Label correcting.

1. $L(x) \leftarrow 0$ and for every $v \neq x$ set $L(v) \leftarrow \infty$.
2. While there is an arc $e = u \rightarrow v$ such that $L(v) > L(u) + w(e)$ set $L(v) \leftarrow L(u) + w(e)$ and $pred(v) = u$.

Example 2.1.5. Now we apply Ford's algorithm to the digraph of Figure 2.1.7, which has some negative weights, but no negative cycles. We will find the distances from vertex a.

iteration	arc	current label					
		a	b	c	d	e	f
init		0	∞	∞	∞	∞	∞
1	$a \rightarrow b$	0	2.0	∞	∞	∞	∞
2	$b \rightarrow c$	0	2.0	4.3	∞	∞	∞
3	$c \rightarrow e$	0	2.0	4.3	∞	4.0	∞
4	$c \rightarrow d$	0	2.0	4.3	5.8	4.0	∞
5	$d \rightarrow e$	0	2.0	4.3	5.8	2.8	∞
6	$d \rightarrow f$	0	2.0	4.3	5.8	2.8	8.0

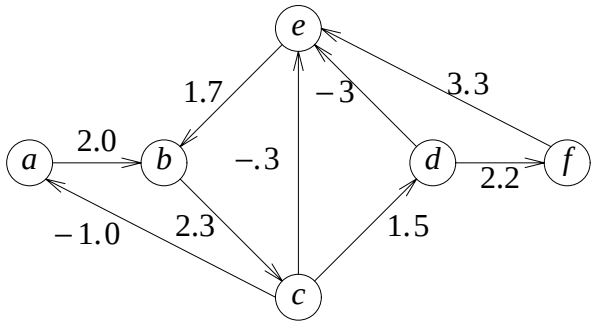


Figure 2.1.7. A digraph to test Ford's algorithm.

The next result assures us that Ford's algorithm actually produces the desired distances.

Theorem 2.1.5 For a digraph D with no negative cycles, when Algorithm 2.1.3 halts, $L(v) = d(x, v)$ for every vertex v.

In order to compute the time complexity of Ford's algorithm, we begin by ordering the arcs of D, say $e_1, e_2, \ldots, e_{|E|}$. We then perform step 2 of the algorithm by examining the arcs in this order. After examining all the arcs, we continue repeating this process until one entire pass through the arc list occurs without changing any label. If D contains no negative cycles, then it can be shown that if a shortest $x - v$ path contains k arcs, then v will have its final label by the end of the kth pass through the arc list (see exercises). Since $k \leq |V|$ Ford's algorithm then has time complexity bounded by $O(|E|\,|V|)$. Further, we can detect a negative cycle by seeing if there is any improvement in the labels on pass $|V|$

Can you modify Ford's algorithm to make it easier to find the path?

Unfortunately, Ford's algorithm can only be used on digraphs. The problem with graphs is that any edge $e = xy$ with negative weight causes the algorithm to continually use this edge while decreasing the labels of x and y.

We conclude this section with a $O(|V|^3)$ algorithm from Floyd [3] for finding all distances in a digraph. Floyd's algorithm also allows negative weights. To accomplish Floyd's algorithm, we define for $i \neq j$:

$$d^0(v_i, v_j) = \begin{cases} l(e) & \text{if } v_i \to v_j \\ \infty & \text{otherwise} \end{cases}$$

and let $d^k(v_i, v_j)$ be the length of a shortest path from v_i to v_j among all paths from v_i to v_j that use only vertices from the set $\{ v_1, v_2, \ldots, v_k \}$. The distances are then updated as we allow the set of vertices used to build paths to expand. Thus, the d^0 distances represent the arcs of the digraph, the d^1 distances represent paths of length at most two that include v_1, etc. Note that because there are no negative cycles, the $d^k(v_i, v_i)$ values will remain at 0.

Algorithm 2.1.4 Floyd's Distance Algorithm.
Input: A digraph $D = (V, E)$ without negative cycles.
Output: The distances from v_i to v_j.
Method: Constant refinement of the distances as the set of excluded vertices is decreased.

1. $k \leftarrow 1$.
2. For every $1 \le i, j \le n$,
   $d^k(v_i, v_j) \leftarrow \min \{ d^{k-1}(v_i, v_j), \; d^{k-1}(v_i, v_k) + d^{k-1}(v_k, v_j) \}$.
3. If $k = |V|$, then stop;
   else $k \leftarrow k + 1$ and go to 2.

Example 2.1.6. We show one pass of Floyd's algorithm on the digraph of Figure 2.1.8. The d^0 and d^1 distances are shown in Tables 2.1.3 and 2.1.4:

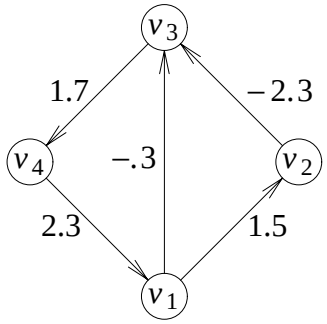


Figure 2.1.8. A digraph for Floyd's Algorithm.

d^0	v_1	v_2	v_3	v_4
v_1	0	1.5	-.3	∞
v_2	∞	0	-2.3	∞
v_3	∞	∞	0	1.7
v_4	2.3	∞	∞	0

Table 2.1.3 The d^0 distances.

We now find the d^1 distances as:

$$d^1(v_i, v_j) \leftarrow \min \{ d^0(v_i, v_j),\ d^0(v_i, v_1) + d^0(v_1, v_j) \}.$$

d^1	v_1	v_2	v_3	v_4
v_1	0	1.5	-.3	∞
v_2	∞	0	-2.3	∞
v_3	∞	∞	0	1.7
v_4	2.3	3.8	2.0	0

Table 2.1.4 The d^1 distances.

Many other algorithms exist to find distances in graphs. However, the techniques used in these algorithms often depend upon higher data structures or sorting techniques and are beyond the scope of this text. The interested reader should see the fine article by Gallo and Pallottino [6].

Section 2.2 Connectivity

In Chapter 1 we defined the idea of a connected graph, that is, a graph in which there is a path between any two vertices. In the last section, while constructing algorithms for finding distances, we actually developed algorithms for testing if a graph is connected. For example, if we apply BFS to a graph $G = (V, E)$, and we are able to visit every vertex of V, then G is connected. This follows by observing that for an arbitrary pair of vertices u and w, there is a path from x to w and from x to u; so these two paths can be used together to form a path from u to w.

There are other techniques for determining if a graph is connected. Perhaps the best known of these is the approach of Tremaux [11] (see also [7]) known as the *depth-first search.* The idea is to begin at a vertex v_0 and visit any vertex adjacent to v_0, say v_1. Now visit any vertex that is adjacent to v_1 that has not yet been visited. Continue to perform this process as long as possible. If we reach a vertex v_k with the property that all its neighbors have been visited, we backtrack to the last vertex visited prior to going to v_k, say v_{k-1}. We then try to visit new vertices neighboring v_{k-1}. If we can find an unvisited neighbor of v_{k-1}, we visit it. If we cannot find such a vertex, we again backtrack to the vertex visited immediately before v_{k-1}, say v_{k-2}. We continue visiting new vertices where possible and backtracking where necessary until we backtrack to v_0 and there are no unvisited neighbors at v_0. At this stage we have visited all possible vertices reachable from v_0, and we stop.

The set of edges used in the depth-first search from v_0 are the edges of a tree. Upon returning to v_0, if we cannot continue the search and the graph still contains unvisited vertices, we may choose such a vertex and begin the algorithm again. When all vertices have been visited, the edges used in performing these visits are the edges of a forest (simply a tree when the graph is connected).

The depth-first search algorithm actually partitions the edge set into two sets T and B: those edges T contained in the forest (and, hence, used in the search), which are usually called *tree edges,* and the remaining edges $B = E - T$, called *back edges.* The set B of back edges can be further partitioned as follows: Let B_1 be the set of back edges that join two vertices x and y, along some path from v_0 to y in the depth-first search tree that begins at v_0. Let C be the set of edges in B that join two vertices joined by a unique tree path that contains v_0. The edges of C are called *cross edges,* since they are edges between vertices that are not descendents of one another in the depth-first search tree.

Although it is not necessary in the search, we shall number the vertices v with an integer $n(v)$. The values of $n(v)$ represent the order in which the vertices are first encountered during the search. This numbering will be very useful in several problems that we will study later.

Algorithm 2.2.1 Depth-First Search.
Input: A graph $G = (V, E)$ and a distinguished vertex x.
Output: A set T of tree edges and an ordering $n(v)$ of the vertices.
Method: Use a label $m(e)$ to determine if an edge has been examined.
Use $p(v)$ to record the vertex previous to v in the search.

1. For each $e \in E$, do the following: Set $m(e) \leftarrow$ "unused."
   Set $T \leftarrow \varnothing$, $i \leftarrow 0$.

For every $v \in V$, do the following: set $n(v) \leftarrow 0$.

2. Let $v \leftarrow x$.
3. Let $i \leftarrow i + 1$ and let $n(v) \leftarrow i$.
4. If v has no unused incident edges, then go to 6.
5. Find an unused edge $e = uv$ and set $m(e) \leftarrow$ "used." Set $T \leftarrow T \cup \{ e \}$.
   If $n(u) \neq 0$, then go to 4;
   else $p(u) \leftarrow v$, $v \leftarrow u$ and go to 3.
6. If $n(v) = 1$, then halt; else $v \leftarrow p(v)$ and go to 4.

There is an alternate way of expressing recursive algorithms that often simplifies their description. This method involves the use of procedures. The idea of a procedure is that it is a "process" that can be repeatedly used in an algorithm. It is started with certain values, and the variables determined within the procedure are local to it. The procedure may be halted at some stage by reference to another procedure or by reference to itself. Should this happen, another version of the procedure is invoked, with new parameters passed to it, and all values of the old procedure are saved, until this procedure once again begins its work. We now demonstrate such a procedural description of the depth-first search algorithm.

Algorithm 2.2.2. Recursive Version of the Depth-First Search.
Input: A graph $G = (V, E)$ and a starting vertex v.
Output: A set T of tree edges and an ordering of the vertices traversed.

1. Let $i \leftarrow 1$ and let $F \leftarrow \varnothing$. For all $v \in V$, do the following: Set $n(v) \leftarrow 0$.
2. While for some u, $n(u) = 0$, do the following: DFS(u).
3. Output T.

(The recursive procedure DFS is now given.)

Procedure DFS(v)

1. Let $n(v) \leftarrow i$ and $i \leftarrow i + 1$.
2. For all $u \in N(v)$, do the following:
   if $n(u) = 0$, then $T \leftarrow T \cup \{ e = uv \}$
   DFS(u)

end DFS

The depth-first search is an extremely important tool, with applications to many other algorithms. Often, an algorithm for determining a graph property or parameter is dependent on the structure of the particular graph under consideration, and we must search that graph to discover this structural dependence. We shall have occasion to make use of the depth-first search as we continue to build other more complicated algorithms.

We now have a variety of ways of determining if a graph is connected. However, when dealing with graphs, you realize quickly that some graphs are "more connected" than others. For example, the path P_n seems much less connected than the complete graph K_n, in the sense that it is much easier to disconnect P_n (remove one internal vertex or edge) than it is to disconnect K_n (where removing a vertex merely reduces K_n to K_{n-1}).

Denote by $k(G)$ the *connectivity of G*, which is defined to be the minimum number of vertices whose removal disconnects G or reduces it to a single vertex K_1. Analogously, the edge connectivity, denoted $k_1(G)$, is the minimum number of edges whose removal disconnects G. If G is disconnected, then $k(G) = 0 = k_1(G)$. If $G = K_n$, then $k(G) = n - 1 = k_1(G)$, as we must remove $n - 1$ vertices to reduce K_n to K_1, and we can disconnect any vertex by removing the $n - 1$ edges incident with it.

We say G is *n-connected* if $k(G) \geq n$ and *n-edge connected* if $k_1(G) \geq n$. A set of vertices whose removal increases the number of components in a graph is called a *vertex separating set* (or *vertex cut set*) and a set of edges whose removal increases the number of components in a graph is called an *edge separating set* (or *edge cut set*). If the context is clear, we will simply use the term *separating set* (or *cut set*). If a cut set consists of a single vertex, it is called a *cut vertex* (some call it an *articulation point*), while if the cut set consists of a single edge, this edge is called a *cut-edge* or *bridge*.

Paths can be used to describe both cut vertices and bridges.

Theorem 2.2.1 In a connected graph G:

1. A vertex v is a cut vertex if, and only if, there exist vertices u and w $(u, w \neq v)$ such that v is on every $u - w$ path of G.
2. An edge e is a bridge if, and only if, there exist vertices u and w such that e is on every $u - w$ path of G.

Proof. To prove (1), let v be a cut vertex of G. If u and w are vertices in different components of $G - v$, then there are no $u - w$ paths in $G - v$. However, since G is

connected, there are $u - w$ paths in G. Thus, v must lie on every $u - w$ path in G.

Conversely, suppose that there exist vertices u and w in G such that v lies on every $u - w$ in G. Then in $G - v$, there are no $u - w$ paths, and so $G - v$ is disconnected. Thus, v is a cut vertex of G.

To prove (2), let e be a bridge of G. Then $G - e$ is disconnected. If u and w are vertices in different components of $G - e$, then there are no $u - w$ paths in $G - e$. But, since G is connected, there are $u - w$ paths in G. Thus, e must be on every $u - w$ path of G.

Conversely, if there exist vertices u and w such that e is on every $u - w$ path in G, then clearly in $G - e$ there are no $u - w$ paths. Hence, $G - e$ is disconnected, and, hence, e is a bridge. □

The depth-first search algorithm can be modified to detect the *blocks* of a graph, that is, the maximal 2-connected subgraphs. The strategy of the algorithm is based on the following observations which are stated as a sequence of lemmas.

Lemma 2.2.1 Let G be a connected graph with DFS tree T. If vw is not a tree edge, then it is a back edge.

Lemma 2.2.2 For $1 \leq i \leq k$, let $G_i = (V_i, E_i)$ be the blocks of a connected graph G. Then

1. For all $i \neq j$, $V_i \cap V_j$ contains at most one vertex.
2. Vertex x is a cut vertex if, and only if, $x \in V_i \cap V_j$ for some $i \neq j$.

Lemma 2.2.3 Let G be a connected graph and $T = (V, E_1)$ be a DFS search tree for G. Vertex x is a cut vertex of G if, and only if,

1. x is the root and x has more than one child in T, or
2. x is not the root and for some child s of x, there is no back edge between a descendant of s (including s itself) and a proper ancestor of x.

Suppose we perform a DFS on G and number the vertices $n(v)$ as usual. Further, suppose we define a lowpoint function LP(v) as the minimum number of a vertex reachable from v by a path in T followed by at most one back edge. Then we can use the lowpoint function to help us determine cut vertices.

Lemma 2.2.4 If v is not a root of T (a DFS tree for G) then v is a cut vertex if, and only if, v has a child s in T with $LP(s) \geq n(v)$.

These observations are illustrated in Figure 2.2.1, where tree edges are shown as dashed lines.

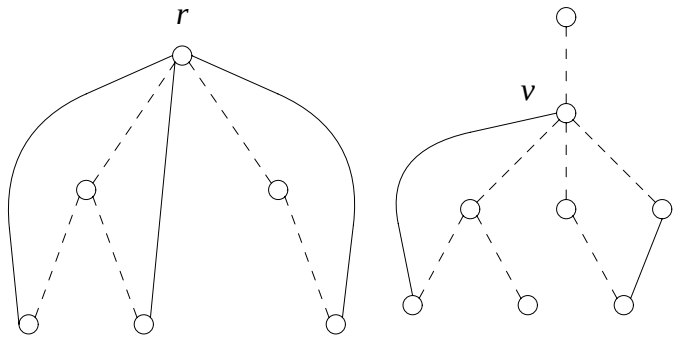


Figure 2.2.1. Cut vertex r as root and cut vertex v, not the root.

In order to identify the blocks of a graph, we really must identify the cut vertices. In order to make use of our observations, we must employ a device called a *stack* to help us. We record values on a stack and retrieve these values in the reverse order of their placement on the stack. The name stack comes from the idea that this structure behaves much like a stack of trays in a cafeteria. The first trays placed on the stack are the last ones taken (from the top) for use. Thus, stacks are often described as first in-last out devices.

Algorithm 2.2.3 Finding Blocks of a Graph.
Input: A connected graph $G = (V, E)$.
Output: The edges in each block of G.
Method: A modified DFS.

1. Set $T \leftarrow \varnothing$, $i \leftarrow 1$, and for all $v \in V$ set $n(v) \leftarrow 0$.
2. Select $v_0 \in V$ and call $UDFS(v_0)$
3. When vertex w is encountered in UDFS, push the edge vw on the stack (if it is not already there).

4. After discovering a pair vw such that w is a child of v and $LP(w) \geq n(v)$, pop from the stack all edges up to and including vw. These are the edges from a block of G

Upgraded DFS - Procedure $UDFS(v)$

1. $i \leftarrow 1$
2. $n(v) \leftarrow i, LP(v) \leftarrow n(v), i \leftarrow i + 1.$
3. For all $w \in N(v)$, do the following:
   If $n(w) = 0$, then
   add uv to T
   $p(w) \leftarrow v$
   $UDFS(w)$
   If $LP(w) \geq n(v)$, then a block has been found.
   $LP(v) \leftarrow \min (LP(v), LP(w))$
   Else If $w \neq p(v)$, then
   $LP(v) \leftarrow \min (LP(v), n(w))$

A simple inequality from Whitney [12] relates connectivity, edge connectivity and the minimum degree of a graph.

Theorem 2.2.2 For any graph G, $k(G) \leq k_1(G) \leq \delta(G)$.

Proof. If G is disconnected, then clearly $k_1(G) = 0$. If G is connected, we can certainly disconnect it by removing all edges incident with a vertex of minimum degree. Thus, in either case, $k_1(G) \leq \delta(G)$.

To verify that the first inequality holds, note that if G is disconnected or trivial, then clearly $k(G) = k_1(G) = 0$. If G is connected and has a bridge, then $k_1(G) = 1 = k(G)$, as either $G = K_2$ or G is connected and contains cut vertices. Finally, if $k_1(G) \geq 2$, then the removal of $k_1(G) - 1$ of the edges in an edge-separating set leaves a graph that contains a bridge. Let this bridge be $e = uv$. For each of the other edges, select an end vertex other than u or v and remove it from G. If the resulting graph is disconnected, then $k(G) < k_1(G)$. If the graph is connected, then it contains the bridge e and the removal of either u or v disconnects it. In either case, $k(G) \leq k_1(G)$, and the result is verified. □

Can you find a graph G for which $k(G) < k_1(G) < \delta(G)$?

Bridges played an important role in the last proof. We can characterize bridges, again taking a structural view using cycles.

Theorem 2.2.3 In a graph G, the edge e is a bridge if, and only if, e lies on no cycle of G.

Proof. Assume G is connected and let $e = uv$ be an edge of G. Suppose e lies on a cycle C of G. Also let w_1 and w_2 be distinct arbitrary vertices of G. If e does not lie on a $w_1 - w_2$ path P, then P is also a $w_1 - w_2$ path in $G - e$. If e does lie on a $w_1 - w_2$ path P', then by replacing e by the $u - v$ (or $v - u$) path of C not containing e produces a $w_1 - w_2$ walk in $G - e$. Thus, there is a $w_1 - w_2$ path in $G - e$ and hence, e is not a bridge.

Conversely, suppose $e = uv$ is an edge of G that is on no cycle of G. Assume e is not a bridge. Then, $G - e$ is connected and hence there exists a $u - v$ path P in $G - e$. Then P together with the edge e produces a cycle in G containing e, a contradiction. □

With the aid of Theorem 2.2.3, we can now characterize 2-connected graphs. Once again cycles play a fundamental role in the characterization. Before presenting the result, we need another definition. Two $u - v$ paths P_1 and P_2 are said to be *internally disjoint* if

$$V(P_1) \cap V(P_2) = \{ u, v \}.$$

Theorem 2.2.4 (Whitney [12]). A graph G of order $p \geq 3$ is 2-connected if, and only if, any two vertices of G lie on a common cycle.

Proof. If any two vertices of G lie on a common cycle, then clearly there are at least two internally disjoint paths between these vertices. Thus, the removal of one vertex cannot disconnect G, that is, G is 2-connected.

Conversely, let G be a 2-connected graph. We use induction on $d(u, v)$ to prove that any two vertices u and v must lie on a common cycle.

If $d(u, v) = 1$, then since G is 2-connected, the edge uv is not a bridge. Hence, by Theorem 2.2.3, the edge uv lies on a cycle. Now, assume the result holds for any two vertices at a distance less than d in G and consider vertices u and v such that $d(u, v) = d \geq 2$. Let P be a $u - v$ path of length d in G and suppose w precedes v on P. Since $d(u, w) = d - 1$, the induction hypothesis implies that u and w lie on a

common cycle, say C.

Since G is 2-connected, $G - w$ is connected and, hence, contains a $u - v$ path P_1. Let z (possibly $z = u$) be the last vertex of P_1 on C. Since $u \in V(C)$, such a vertex must exist. Then G has two internally disjoint paths: one composed of the section of C from u to z not containing w together with the section of P_1 from z to v, and the other composed of the other section of C from u to w together with the edge wv. These two paths thus form a cycle containing u and v. □

A very powerful generalization of Whitney's theorem was proved by Menger [8]. Menger's theorem turns out to be related to many other results in several branches of discrete mathematics. We shall see some of these relationships later. Although a proof of Menger's theorem could be presented now, we postpone it until Chapter 4 in order to better point out some of these relationships to other results.

Theorem 2.2.5 (Menger's theorem). For nonadjacent vertices u and v in a graph G, the maximum number of internally disjoint $u - v$ paths equals the minimum number of vertices that separate u and v.

Theorem 2.2.4 has a generalization (Whitney [12]) to the k-connected case. This result should be viewed as the global version of Menger's theorem.

Theorem 2.2.6 A graph G is k-connected if, and only if, all distinct pairs of vertices are joined by at least k internally disjoint paths.

It is natural to ask if there is an edge analog to Menger's theorem. This result was independently discovered much later by Ford and Fulkerson [5] and Elias, Feinstein and Shannon [2]. We postpone the proof of this result until Chapter 4.

Theorem 2.2.7 For any two vertices u and v of a graph G, the maximum number of edge disjoint paths joining u and v equals the minimum number of edges whose removal separates u and v.

Section 2.3 Digraph Connectivity

The introduction of direction to the edges of a graph complicates the question of connectivity. In fact, we already know there are several levels of connectivity possible for digraphs. To help clarify the situation even further, we define a $u - v$ *semiwalk* to be a sequence $u = v_1, v_2, \ldots, v_k = v$, where for each $i = 1, 2, \ldots, k$, either $v_i \rightarrow v_{i+1}$ or $v_i \leftarrow v_{i+1}$ is an arc of the digraph. Can you define a $u - v$ semipath? A semipath may be a directed path in the digraph, or it may not. However, a semipath would be a $u - v$ path in the underlying graph. We say a digraph D is

1. *weakly connected* if every two vertices of D are joined by a semipath
2. *unilaterally connected (or unilateral*) if for every two vertices u and v, there is a directed $u - v$ path or a directed $v - u$ path in D
3. *strongly connected* (or *strong*) if all pairs of vertices u and v are joined by both a $u - v$ and a $v - u$ directed path.

If D satisfies none of these conditions, we say D is *disconnected.* As you might expect, each type of digraph connectivity can be characterized in terms of spanning semiwalks or paths. The proof of Theorem 2.3.1 is similar in nature to that of connectivity in graphs, and so it is omitted.

Theorem 2.3.1 Let D be a digraph. Then

1. D is weakly connected if, and only if, D contains a spanning semiwalk
2. D is unilateral if, and only if, D contains a spanning walk
3. D is strong if, and only if, D contains a closed spanning walk.

We say a vertex u is *reachable from* v if there exists a directed $v - u$ path in the digraph. The set of all vertices that are reachable from v is denoted as $R(v)$. The relation "mutually reachable" is an equivalence relation (see exercises); hence, this relation partitions the vertex set into classes $V_1, V_2, \ldots, V_k$ $(k \geq 1)$. Since the vertices u and v are in the same equivalence class if, and only if, D contains both a $u - v$ and $v - u$ directed path, the subgraphs $S_i = < V_i >$ have come to be called the *strong components* of D. Despite the fact that the strong components of D partition the vertex set of D, they do not necessarily partition the arc set. This fact can be seen in the example in Figure 2.3.1.

The term *strong component* is still appropriate, even when D is weakly connected or unilateral, since if S_1 and S_2 are two strong components, all arcs between these strong components are directed in one way, from S_1 to S_2 or from S_2 to S_1. Thus, there will always be vertices in one of these components that cannot reach any vertex of the other component.

Tarjan [10] developed an algorithm for finding the strongly connected components of a digraph. This algorithm makes use of the digraph $S_D = (V_S, E_S)$, called the *superstructure* of $D = (V, E)$, where

$$V_S = \{S_1, S_2, \ldots, S_k\} \text{ and}$$

$$E_S = \{ e = S_i \rightarrow S_j \mid i \neq j \text{ and } x \rightarrow y \in E \text{ where } x \in S_i \text{ and } y \in S_j \}.$$

Note that the digraph S_D must be acyclic, for if it were not, then all strongly connected components on some cycle of S_D would form one strongly connected component of D, contradicting the way the strong components were chosen. Now, we see that since S_D is acyclic, some vertex, say S_j, must have outdegree zero.

Suppose that we perform a depth-first search on D. Let v be the first vertex of S_j to be visited during this search. Since all the vertices of S_j are reachable from v, the depth-first search will never backtrack from v until all of the vertices in S_j have been visited. Thus, the number $n(u)$ assigned to each vertex u as it is first reached during the DSF ensures that each vertex of S_j has a number at least as large as $n(v)$. Since there are no arcs out of S_j, no vertex outside of S_j is visited from the time we first encounter v until we finally backtrack from v. Our only remaining problem is to determine when we actually perform a backtrack on the first vertex encountered in that strong component.

In order to solve this problem, we again turn to the bookkeeping lowpoint function. Here, the *lowpoint of v*, denoted $LP(v)$, is the least number $n(u)$ of a vertex u reachable from v using a (possibly empty) directed path consisting of tree arcs followed by at most one back arc or cross arc, provided u is in the same strong component as v. This definition seems circular. To find strong components, we need the lowpoint, and to find the lowpoint, we need to know the strong components. Tarjan [10] eliminated this problem by using a stack. The vertices visited are stored on a stack in the order in which they are reached during the search. For each vertex we also record whether it is on the stack, using the function onstack (with values of true or false). Again we traverse the digraph using a depth-first search.

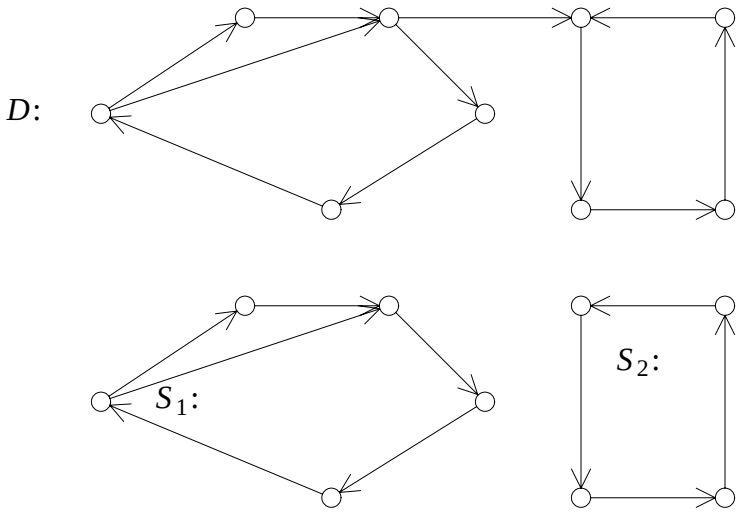


Figure 2.3.1. A digraph D and its strong components S_1 and S_2.

Algorithm 2.3.1 Finding Strongly Connected Components.
Input: A digraph D.
Output: The vertices of the strong components of D.

1. Set $i \leftarrow 1$ and empty the stack.
2. For all $v \in V$, do the following:
   $n(v) \leftarrow 0$
   onstack(v) $\leftarrow$ false.
3. While $n(v) = 0$ for some v, do the following: strongcomp(v).

Procedure strongcomp(v)

1. Set $n(v) \leftarrow i$, $LP(v) \leftarrow n(v)$ and $i \leftarrow i + 1$.
2. Place v on the stack and set onstack(v) $\leftarrow$ true.
3. For all $u \in N(v)$, do the following:
4. If $n(u) = 0$, then do the following:
5. strongcomp(u)

6. $LP(v) \leftarrow \min \{ LP(v), LP(u) \}$

7. Else if $n(u) < n(v)$ and onstack(u) = true,
   then $LP(v) \leftarrow \min \{ LP(v), n(u) \}$.

8. If $LP(v) = n(v)$ then delete and output the stack from the top down through v, and for each such vertex w, set onstack(w) $\leftarrow$ false.

Can you determine the complexity of the strong component algorithm?

Section 2.4 Problem Solving and Heuristics

Suppose you are confronted with the following problem to solve (what else is new?). You have two water jugs, a 4-gallon jug and a 3-gallon jug. Neither jug has any measure markings on it. There is also a water pump that you can use to fill the jugs. Your problem, however, is that you want exactly 2 gallons of water in the larger jug so that you can make your secret recipe. How can you solve this problem?

The solution of this particular problem can actually help us see what some of the general techniques for problem solving are like. These techniques revolve around a search (often on a computer) for the solution among all possible situations the problem can produce. We may not even be sure that a solution exists when we begin this search or even know what the structure of the graph model happens to be.

There are many approaches one might take to problem solving, but any organized approach certainly encompasses the following points:

- Define the problem precisely. This includes precise specifications of what the initial situation will be as well as what constitutes an acceptable solution to the problem.
- Analyze the problem. Some feature or features can have a tremendous impact on the techniques we should use in solving the problem. Understand the "legal moves" you can make to try to find the solution.
- Choose the best technique and apply it.

We have just defined the water jug problem. We know that in the initial situation, both jugs are empty. We also know that a solution is found when there are exactly 2 gallons of water in the 4-gallon jug. Next, we must analyze the problem to try to determine what techniques to apply.

We can perform several "legal" operations with the water jugs:

- We can fill either jug completely from the pump.
- We can pour all the water from one jug into the other jug.
- We can fill one jug from the other jug.
- We can dump all the water from either jug.

Why have we restricted our operations in these ways? The answer is so that we can maintain control over the situation. By restricting operations in this way, we will always know exactly how much water is in either jug at any given time. Note that these operations also ensure that our search must deal with only a finite number of situations, since each jug can be filled to only a finite number of levels.

We have just examined an important concept, the idea of "moving" from one situation to another by performing one of a set of *legal moves.* What we want to do is to move from our *present state* (the starting vertex in the state space model) to some other state via these legal moves. Just what constitutes a legal move is dependent on the problem at hand and should be the result of our problem analysis.

The digraph model we have been building should now be apparent. Represent each state that we can reach by a vertex. There is an arc from vertex *a* to vertex *b* if we can move from state *a* to state *b* via some legal move. Our digraph and the collection of legal moves that define its arcs, constitute *the state space* (or the *state graph*) of the problem. Let's determine the state space of the water jug problem.

Label the state we are in by the ordered pair (j_1, j_2), which shows the amount of water in the 4-gallon and 3-gallon jugs, respectively. The initial state is then labeled $(0, 0)$. From this state we can move to either $(4, 0)$ or $(0, 3)$. Since we can move from these states back to $(0, 0)$, for simplicity we will join these vertices by one undirected edge representing the symmetric pair of arcs. We picture this in Figure 2.4.1.

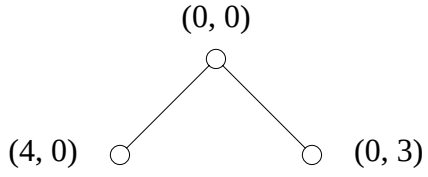


Figure 2.4.1. Early states we can reach.

From $(4, 0)$ we can move to $(0, 0)$, $(4, 3)$ or $(1, 3)$; while from $(0, 3)$ we can move to $(0, 0)$, $(4, 3)$ or $(3, 0)$. Thus, we have the situation in Figure 2.4.2.

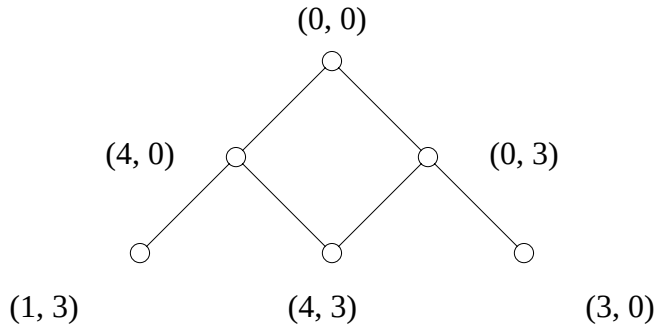


Figure 2.4.2. Two levels of the search.

It makes no sense for us to follow arcs that would return us to states we have already determined are not solutions; therefore, we will not bother to include arcs like the one from $(3, 0)$ to $(0, 0)$. As we continue to generate new states, we will eventually reach the graph in Figure 2.4.3.

We can see that this graph contains several possible solutions as well as several different paths from the initial vertex to these solutions. Thus, there are many ways to solve this particular problem. In fact, since any path from $(0, 0)$ to any vertex $(2, j_2)$ demonstrates a set of legal moves necessary to produce an acceptable solution, we will content ourselves with finding any one of these paths.

Our diagram representing the development of the state space also serves to point out another important fact. Rarely would we be presented with the state space and asked to find a path from the initial vertex to the solution. Instead, we would begin at a start vertex *s* and generate those vertices reachable from *s* in one move. If no solution is present, we begin generating neighbors of these vertices as we search for a solution. Thus, our search amounts to a *blind search,* that is, we cannot see the entire graph, only the local neighbors of the vertex we are presently examining. Luckily, we have already learned two search techniques that are essentially blind searches. Both the breadth-first search and the depth-first search are designed to operate in exactly this fashion.

There are some problems with using these search techniques on state spaces. The fundamental difficulty is that we have no idea how large the state space may actually be, and, therefore, we have no idea how many vertices we might have to examine. Especially in the case of the breadth-first search, where entire new levels of neighbors are being produced at one time, the amount of data we need to handle may increase exponentially. In the case of either search, there may simply be too many intermediate

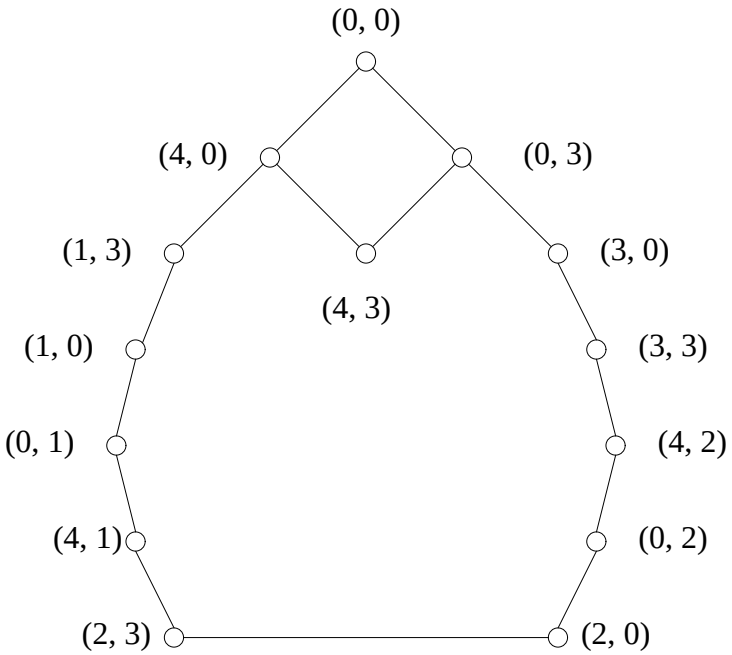


Figure 2.4.3. Many levels of the search (not all edges shown).

vertices to examine to be able to find any solution.

Often, we must sacrifice the completeness of a systematic search and construct a different control mechanism, one that hopefully will produce an acceptable solution in a reasonable amount of time. There are two approaches that are often tried:

- Perhaps it is the case that an exact answer is not necessary, that is, something close will be good enough for our purpose. Thus, we search for the best possible "approximate" answer we can find within a reasonable amount of time.
- Suppose, instead, that we sacrifice the guarantee (provided we have ample space and time) of finding a solution, because we make decisions during the search that mean we will never actually examine all vertices. If these decisions are good ones, we may work our way to a solution faster simply because we avoided taking a "wrong turn" in our search. This approach is clearly more dangerous because we may conclude our search with nothing more than what we started with. But, when it is clear that we may not be able to complete a thorough search anyway, the risk is often worth taking.

A *heuristic* is a technique that improves the efficiency of a search, while possibly sacrificing claims of completeness. Heuristics are very much like guided tours; they are

intended to point out the highlights and eliminate time wasted on unnecessary events. Some heuristics are of more help than others, and usually this is a problem-dependent characteristic. However, several general heuristic techniques have become popular.

One general technique is called the *nearest neighbor algorithm.* The idea is that we examine all unvisited neighbors of some vertex and next visit the neighbor that most satisfies some test criterion. Our hope is that the test criterion will point us more rapidly toward a solution.

For example, suppose we apply the following heuristic to the water jug problem: During a DFS search we shall next visit the neighboring vertex whose label (j_1, j_2) has j_1 closest to the desired value of two. Under these conditions our search would proceed as follows:

- From $(0, 0)$ we generate $(4, 0)$ and $(0, 3)$. Since either has first coordinate within two of the goal, we randomly select the first as our next state.
- From $(4, 0)$ we generate the neighbors $(1, 3)$ and $(4, 3)$, and since $(1, 3)$ is best, we move there.
- From $(1, 3)$ we generate in turn (as there is only one neighbor each time) the sequence $(1, 0)$, $(0, 1)$, $(4, 1)$ and $(2, 3)$ and, thus, solve the problem.

In performing the above process, we examined less than half of the vertices in the state space and, thus, speeded our finding of a solution by a considerable amount. The general process we applied has been given the name *best-first search* because we used the "best" neighbor as our next choice.

Other modifications in these techniques are possible. Suppose that we decide to move only to the unvisited neighbor that produces the greatest improvement in our position, relative to some heuristic test. This technique is called *hill climbing.* We test all unvisited neighbors of the present vertex and using this information move to the vertex of greatest improvement. As you might already suspect, there are some obvious potential problems with this approach.

The major issue is what we do if the search reaches a vertex that is not a solution, but from which there are no neighbors that improve our position. There are several ways that this could happen. A *local maximum* is achieved if the present vertex is not a solution and all neighbors fail to improve our position. If, in fact, all the neighbors are essentially equivalent to our present vertex we say that a *plateau* has been reached. A far worse situation would be that the state space was not a connected graph and we were in a component that contained no solutions. Then, no simple move would ever achieve our

goal.

Typical strategies for dealing with these problems are:

- Backtrack to a previous vertex and start the search again.
- Make a "big jump" to a new vertex, possibly by making two or more moves without regard to the heuristic test.
- Apply two or more moves all the time, using several levels of vertices to try to determine the next state. This process is called *lookahead.*

We have discussed a variety of options for trying to search for solutions to difficult problems. The central theme in each is that the set of possible solutions can be viewed as a (possibly infinite) graph or digraph. If we are able to search this graph exhaustively, we will find a solution, provided one exists.

However, if we cannot perform an exhaustive search, we are not necessarily doomed to failure. Creative heuristic tests can be (and often are) a great deal of help. The descriptions here are by no means a complete list of "standard heuristics" (if any such thing exists), but merely an indication that we should not immediately abandon a search when the graph model seems too large to handle.

Exercises

1. Show that graph distance is a metric function. Is distance still a metric function on labeled or weighted graphs?
2. Modify the BFS labeling process to make it easier to find the $x - v$ distance path.
3. Modify the BFS algorithm to find the distance from x to one specified vertex y.
4. Develop a recursive version of the BFS algorithm.
5. What modifications are necessary to make Dijkstra's algorithm work for undirected graphs?
6. Prove that the relation "is connected to" is an equivalence relation on the vertex set of a graph.
7. Show that if G is a connected graph of order p, then the size of G is at least $p - 1$.
8. Characterize those graphs having the property that every one of their induced subgraphs is connected.

9. Continue Example 2.1.6 by finding the tables for d^2, d^3 and d^4.

10. Prove that every circuit in a graph contains a cycle.

11. Prove that if G is a graph of order p and $\delta(G) \geq \frac{p}{2}$, then $k_1(G) = \delta(G)$.

12. Suppose that G is a (p, q) graph with $k(G) = n$ and $k_1(G) = m$, where both n and m are at least 1. Determine what values are possible for the following:
$$k(G - v),\ k_1(G - v),\ k(G - e),\ k_1(G - e).$$

13. Let G be an n-connected graph and let $v_1, v_2, \ldots, v_n$ be distinct vertices of G. Suppose we insert a new vertex x and join x to each of $v_1, v_2, \ldots, v_n$. Show that this new graph is also n-connected.

14. Prove that if G is an n-connected graph and $v_1, \ldots, v_n$ and v are $n + 1$ vertices of G, then there exist internally disjoint $v - v_i$ paths for $i = 1, \ldots, n$.

15. Show that if G contains no vertices of odd degree, then G contains no bridges.

16. Prove Theorem 2.1.5.

17. Prove Lemma 2.2.1.

18. Prove Lemma 2.2.2.

19. Prove Lemma 2.2.3.

20. Prove Lemma 2.2.4.

21. Prove Theorem 2.3.1.

22. Apply Algorithm 2.3.1 to the digraph D of Figure 2.3.1.

23. In applying Ford's algorithm to a weighted digraph D that contains no negative cycles, show that if a shortest $x - v$ path contains k arcs, then v will have its final label by the end of the kth pass through the arc list.

24. Modify the labeling in Ford's algorithm to make backtracking to find the distance path easier.

25. Show that G contains a path of length at least $\delta(G)$.

26. Show that G is connected if, and only if, for every partition of $V(G)$ into two nonempty sets V_1 and V_2, there is an edge from a vertex in V_1 to a vertex in V_2.

27. Show that if $\delta(G) \geq \frac{p - 1}{2}$, then G is connected.

28. Show that any nontrivial graph contains at least two vertices that are not cut vertices.

29. Show that if G is disconnected, then $\overline{G}$ is connected.

30. Show that if G is connected, then either G is complete or G contains three vertices x, y, z such that xy and yz are edges of G but $xz \notin E(G)$.

31. A graph G is a *critical block* if G is a block and for every vertex v, $G - v$ is not a block. Show that every critical block of order at least 4 contains a vertex of degree 2.

32. A graph G is a *minimal block* if G is a block and for every edge e, $G - e$ is not a block. Show that if G is a minimal block of order at least 4, then G contains a vertex of degree 2.

33. The *block index* $b(v)$ of a vertex v in a graph G is the number of blocks of G to which v belongs. If $b(G)$ denotes the number of blocks of G, show that
$$b(G) = k(G) + \sum_{v \in V(G)} (b(v) - 1).$$

34. Three cannibals and three missionaries are traveling together and they arrive at a river. They all wish to cross the river; however, the only transportation is a boat that can hold at most two people. There is another complication, however; at no time can the cannibals outnumber the missionaries (on either side of the river), for then the missionaries would be in danger. How do they manage to cross the river?

35. Prove that there can be no solution to the three cannibals and three missionaries problem that uses fewer than eleven river crossings.

36. Does a four-cannibal and four-missionary problem make sense? If so, explain this problem and try to solve it.

37. Three wives and their jealous husbands wish to go to town, but their only means of transportation is an RX7, which seats only two people. How might they do this so that no wife is ever left with one or both of the other husbands unless her own husband is present?

38. The 8-puzzle is a square tray in which are placed eight numbered tiles. The remaining ninth square is open. A tile that is adjacent to the open square can slide into that space. The object of this game is to obtain the following configuration from the starting configuration:

start		
2	8	3
1	6	4
7		5

goal		
1	2	3
8		4
7	6	5

How does this problem differ from those studied earlier? Can you build a mechanism into the rules that handles this difference?

39. A problem-solving search can proceed forward (as we have done) or backward from the goal state. What factors should influence your decision on how to proceed?

References

1. Dijkstra, E. W., A Note on Two Problems in Connection with Graphs. *Numerische Math.*, (1959), 269 – 271.
2. Elias, P., Feinstein, A. and Shannon, C. E., A Note on the Maximum Flow Through a Network. *IRE Trans. Inform. Theory*, IT-2(1956), 117 – 119.
3. Floyd, R. W., Algorithm 97: Shortest Path. *Comm. ACM*, 5(1962), 345.
4. Ford, L. R., *Network Flow Theory*. The Rand Corporation, P-923, August, 1956.
5. Ford, L. R. and Fulkerson, D. R., Maximal Flow Through a Network. *Canad. J. Math.*, 8(1956), 399 – 404.
6. Gallo, G. and Pallottino, S., Shortest Path Methods: A Unifying Approach. *Math. Programming Study* 26(1986), 38-64.
7. Hopcroft, J. and Tarjan, R., Algorithm 447: Efficient Algorithms for Graph Manipulation. *Comm. ACM*, 16(1973), 372 – 378.
8. Menger, K., Zur Allgemeinen Kurventheorie. *Fund. Math.*, 10(1927), 95 – 115.
9. Moore, E. F., The Shortest Path Through a Maze. *Proc. Iternat. Symp. Switching Th.*, 1957, Part II, Harvard Univ. Press, (1959), 285 – 292.

10. Tarjan, R., Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(1972), 146 – 160.

11. Tremaux: see Lucas, E., *Recreations Mathematiques.* Paris, 1982.

12. Whitney, H., Congruent Graphs and the Connectivity of Graphs. *Amer. J. Math.*, 54(1932), 150 – 168.

Chapter 3
Trees

Section 3.1 Fundamental Properties of Trees

Suppose your city is planning to construct a rapid rail system. They want to construct the most economical system possible that will meet the needs of the city. Certainly, a minimum requirement is that passengers must be able to ride from any station in the system to any other station. In addition, several alternate routes are under consideration between some of the stations. Some of these routes are more expensive to construct than others. How can the city select the most inexpensive design that still connects all the proposed stations?

The model for this problem associates vertices with the proposed stations and edges with all the proposed routes that could be constructed. The edges are labeled (weighted) with their proposed costs. To solve the rapid rail problem, we must find a connected graph (so that all stations can be reached from all other stations) with the minimum possible sum of the edge weights.

Note that to construct a graph with minimum edge weight sum, we must avoid cycles, since otherwise we could remove the most expensive edge (largest weight) on the cycle, obtaining a new connected graph with smaller weight sum. What we desire is a connected, acyclic graph (hence, a tree) with minimum possible edge weight sum. Such a tree is called a *minimum weight spanning tree.* Before we determine algorithms for finding minimum weight spanning trees, let's investigate more of the properties of trees.

Trees are perhaps the most useful of all special classes of graphs. They have applications in computer science and mathematics, as well as in chemistry, sociology and many other studies. Perhaps what helps make trees so useful is that they may be viewed in a variety of equivalent forms.

Theorem 3.1.1 A graph T is a tree if, and only if, every two distinct vertices of T are joined by a unique path.

Proof. If T is a tree, by definition it is connected. Hence, any two vertices are joined by at least one path. If the vertices u and v of T are joined by two or more different paths, then a cycle is produced in T, contradicting the definition of a tree.

Conversely, suppose that T is a graph in which any two distinct vertices are joined by a unique path. Clearly, T must be connected. If there is a cycle C containing the vertices u and v, then u and v are joined by at least two paths, contradicting the hypothesis.

Hence, T must be a tree. □

Theorem 3.1.2 A (p, q) graph T is a tree if, and only if, T is connected and $q = p - 1$.

Proof. Given a tree T of order p, we will prove that $q = p - 1$ by induction on p. If $p = 1$, then $T = K_1$ and $q = 0$. Now, suppose the result is true for all trees of order less than p and let T be a tree of order $p \geq 2$. Let $e = uv \in E(T)$. Then, by Theorem 3.1.1, $T - e$ contains no $u - v$ path. Thus, $T - e$ is disconnected and in fact, has exactly two components (see Chapter 3, exercise 1). Let T_1 and T_2 be these components. Then T_1 and T_2 are trees, and each has order less than p; hence, by the inductive hypothesis

$$| E(T_i) | = | V(T_i) | - 1, \quad \text{for } i = 1, 2.$$

Now we see that

$$\begin{aligned} | E(T) | &= | E(T_1) | + | E(T_2) | + 1 \\ &= | V(T_1) | + | V(T_2) | - 1 \\ &= p - 1. \end{aligned}$$

Conversely, suppose T is a connected (p, q) graph and $q = p - 1$. In order to prove that T is a tree, we must show that T is acyclic. Suppose T contains a cycle C and that e is an edge of C. Then $T - e$ is connected and has order p and size $p - 2$. But this contradicts exercise 7 in Chapter 2. Therefore, T is acyclic and hence, T is a tree. □

We summarize various characterizations of a tree in the following theorem.

Theorem 3.1.3 The following are equivalent on a (p, q) graph T:

1. The graph T is a tree.
2. The graph T is connected and $q = p - 1$.
3. Every pair of distinct vertices of T is joined by a unique path.
4. The graph T is acyclic and $q = p - 1$.

In any tree of order $p \geq 3$, any vertex of degree at least 2 is a cut vertex. However, every nontrivial tree contains at least two vertices of degree 1, since it contains at least two vertices that are not cut vertices (see Chapter 2, exercise 28). The vertices of degree 1 in a tree are called *end vertices* or *leaves.* The remaining vertices are called *internal vertices.* It is also easy to see that every edge in a tree is a bridge.

Every connected graph G contains a spanning subgraph that is a tree, called a *spanning tree.* If G is itself a tree, this is clear. If G is not a tree, simply remove edges lying on cycles in G, one at a time, until only bridges remain. Typically, there are many different spanning trees in a connected graph. However, if we are careful, we can construct a subtree in which the distance from a distinguished vertex v to all other vertices in the tree is identical to the distance from v to each of these vertices in the original graph. Such a spanning tree is said to be *distance preserving from* v or *v-distance preserving.* The following result is originally from Ore [7].

Theorem 3.1.4 For every vertex v of a connected graph G, there exists a v-distance preserving spanning tree T.

Proof. The graph constructed in the breadth-first search algorithm starting at v is a tree and is clearly distance preserving from v. □

The following result shows that there are usually many trees embedded as subgraphs in a graph.

Theorem 3.1.5 Let G be a graph with $\delta(G) \geq m$ and let T be any tree of order $m + 1$; then T is a subgraph of G.

Proof. We proceed by induction on m. If $m = 0$, the result is clear since $T = K_1$ is a subgraph of any graph. If $m = 1$, the result is also clear since $T = K_2$ is a subgraph of every nonempty graph. Now, assume the result holds for any tree T_1 of order m and any graph G_1 with $\delta(G_1) \geq m - 1$. Let T be a tree of order $m + 1$ and let G be a graph with $\delta(G) \geq m$.

To see that T is a subgraph of G, consider an end vertex v of T. Also, suppose that v is adjacent to w in T. Since $T - v$ is a tree of order m and the graph $G - v$ satisfies $\delta(G - v) \geq m - 1$, we see from the inductive hypothesis that $T - v \subseteq G - v \subseteq G$. Since $deg_G\ w \geq m$ and $T - v$ has order m, the vertex w has an adjacency in G outside of $V(T - v)$. But this implies that T is a subgraph of G. □

Section 3.2 Minimum Weight Spanning Trees

To solve the rapid rail problem, we now want to determine how to construct a minimum weight spanning tree. The first algorithm is from Kruskal [6]. The strategy of

the algorithm is very simple. We begin by choosing an edge of minimum weight in the graph. We then continue by selecting from the remaining edges an edge of minimum weight that does not form a cycle with any of the edges we have already chosen. We continue in this fashion until a spanning tree is formed.

Algorithm 3.2.1 Kruskal's Algorithm.
Input: A connected weighted graph $G = (V, E)$.
Output: A minimum weight spanning tree $T = (V, E(T))$.
Method: Find the next edge e of minimum weight $w(e)$ that does not form a cycle with those already chosen.

1. Let $i \leftarrow 1$ and $T \leftarrow \varnothing$.
2. Choose an edge e of minimum weight such that $e \notin E(T)$ and such that $T \cup \{ e \}$ is acyclic.
   If no such edge exists,
   then stop;
   else set $e_i \leftarrow e$ and $T \leftarrow T \cup \{ e_i \}$.
3. Let $i \leftarrow i + 1$, and go to step 2.

Theorem 3.2.1 When Kruskal's algorithm halts, T induces a minimum weight spanning tree.

Proof. Let G be a nontrivial connected weighted graph of order p. Clearly, the algorithm produces a spanning tree T; hence, T has $p - 1$ edges. Let

$$E(T) = \{e_1, e_2, \ldots, e_{p-1}\} \text{ and let } w(T) = \sum_{i=1}^{p-1} w(e_i).$$

Note that the order of the edges listed in $E(T)$ is also the order in which they were chosen, and so $w(e_i) \leq w(e_j)$ whenever $i \leq j$.

From the collection of all minimum weight spanning trees, let $T_{\min}$ be chosen with the property that it has the maximum number of edges in common with T. If T is not a minimum weight spanning tree, T and $T_{\min}$ are not identical. Let e_i be the first edge of T (following our listing of edges) that is not in $T_{\min}$. If we insert the edge e_i into $T_{\min}$, we get a graph H containing a cycle. Since T is acyclic, there exists an edge e on the cycle in H that is not in T. The graph $H - \{ e \}$ is also a spanning tree of G and

$$w(H - \{ e \}) = w(T_{\min}) + w(e_i) - w(e).$$

Since $w(T_{\min}) \leq w(H - \{ e \})$, it follows that $w(e) \leq w(e_i)$. However, by the algorithm, e_i is an edge of minimum weight such that the graph $< \{ e_1, e_2, \ldots, e_i \} >$ is acyclic. However, since all these edges come from $T_{\min}$,

$< \{ e_1, e_2, \ldots, e_{i-1}, e \} >$ is also acyclic. Thus, we have that $w(e_i) = w(e)$ and $w(H - \{ e \}) = w(T_{\min})$. That is, the spanning tree $H - \{ e \}$ is also of minimum weight, but it has more edges in common with T than $T_{\min}$, contradicting our choice of $T_{\min}$ and completing the proof. □

Example 3.2.1. We demonstrate Kruskal's algorithm on the graph of Figure 3.2.1.

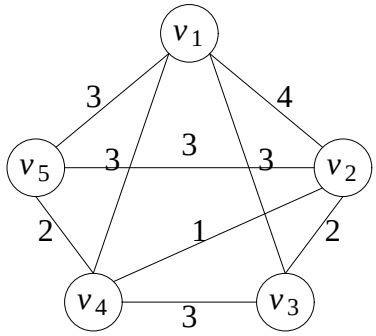


Figure 3.2.1. A weighted graph to test Kruskal's algorithm.

1. $i \leftarrow 1$ and $T \leftarrow \varnothing$. 2–3. $T \leftarrow e = v_2 v_4$ and $i \leftarrow 2$.

2–3. $T \leftarrow T \cup \{ v_2 v_3 \}$ and $i \leftarrow 3$.

2–3. $T \leftarrow T \cup \{ v_4 v_5 \}$ and $i \leftarrow 4$.

2–3. $T \leftarrow T \cup \{ v_1 v_4 \}$ and $i \leftarrow 5$.

2. Halt (with the minimum spanning tree shown in Figure 3.2.2).

In the performance of Kruskal's algorithm, it is best to sort the edges in order of nondecreasing weight prior to beginning the algorithm. On the average, this can be done in $O(q \log q)$ time using either a quicksort or heap sort (see [10]). With this in mind, can you determine the average time complexity of Kruskal's algorithm?

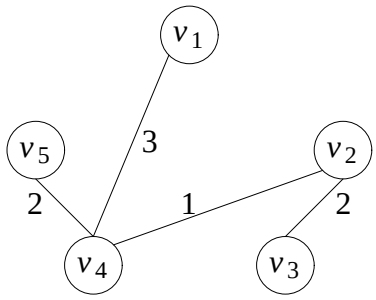


Figure 3.2.2. A minimum spanning tree for the graph of Figure 3.2.1.

Kruskal's algorithm is an example of a type of algorithm known as *greedy*. Simply stated, greedy algorithms are essentially algorithms that proceed by selecting the choice that looks the best at the moment. This local point of view can sometimes work very well, as it does in this case. However, the reader should not think that all processes can be handled so simply. In fact, we shall see examples later in which the greedy approach can be arbitrarily bad.

There are several other algorithms for finding minimum weight spanning trees. The next result is fundamental to these algorithms.

Theorem 3.2.2 Let $G = (V, E)$ be a weighted graph. Let $U \subseteq V$ and let e have minimum weight among all edges from U to $V - U$. Then there exists a minimum weight spanning tree that contains e.

Proof. Let T be a minimum weight spanning tree of G. If e is an edge of T, we are done. Thus, suppose e is not an edge of T and consider the graph $H = T \cup \{ e \}$, which must contain a cycle C. Note that C contains e and at least one other edge $f = uv$, where $u \in U$ and $v \in V - U$. Since e has minimum weight among the edges from U to $V - U$, we see that $w(e) \leq w(f)$. Since f is on the cycle C, if we delete f from H, the resulting graph is still connected and, hence, is a tree. Further, $w(H - f) \leq w(T)$ and hence $H - f$ is the desired minimum weight spanning tree containing e. □

This result directly inspired the following algorithm from Prim [8]. In this algorithm we continually expand the set of vertices U by finding an edge e from U to $V - U$ of minimum weight. The vertices of U induce a tree throughout this process. The end vertex of e in $V - U$ is then incorporated into U, and the process is repeated until $U = V$. For convenience, if $e = xy$, we denote $w(e)$ by $w(x, y)$. We also simply consider the tree T induced by the vertex set U.

Algorithm 3.2.2 Prim's Algorithm.
Input: A connected weighted graph $G = (V, E)$ with $V = \{ v_1, v_2, \ldots, v_n \}$.
Output: A minimum weight spanning tree T.
Method: Expand the tree T from $\{ v_1 \}$ using the minimum weight edge from T to $V - V(T)$.

1. Let $T \leftarrow v_1$.
2. Let $e = tu$ be an edge of minimum weight joining a vertex t of T and a vertex u of $V - V(T)$ and set $T \leftarrow T \cup \{ e \}$.
3. If $|E(T)| = p - 1$ then halt, else go to step 2.

Example 3.2.2. We now perform Prim's algorithm on the graph of Example 3.2.1.

1. $T \leftarrow \{ v_1 \}$.

2. $w(t, u) = 3$ and $T \leftarrow T \cup \{ v_1v_4 \}$. (Note that any of v_1v_3, v_1v_4, or v_1v_5 could have been chosen.)

3. Go to step 2.

2. $w(t, v_2) = 1$, $w(t, v_3) = 3$, $w(t, v_5) = 2$ so select $w(t, v_2) = 1$ and set $T \leftarrow T \cup \{ v_2v_4 \}$.

3. Go to step 2.

2. $w(t, v_3) = 2$, $w(t, v_5) = 2$ so select $w(t, u) = 2$ and set $T \leftarrow T \cup \{ v_4v_5 \}$.

3. Go to step 2.

2. $w(t, u) = 2$ and so set $T \leftarrow T \cup \{ v_2v_3 \}$.

3. Halt

We again obtain the minimum spanning tree T of Figure 3.2.2. □

To determine the time complexity of Prim's algorithm, note that step 2 requires at most $|V| - 1$ comparisons and is repeated $|V| - 1$ times (and, hence, requires $O(|V|^2)$

time). Hence, Prim's algorithm requires $O(|V|^2)$ time.

At this stage we must point out that the corresponding problem of finding minimum weight spanning trees in digraphs is much harder. In fact, there is no known polynomial algorithm for solving such a problem.

Section 3.3 Counting Trees

Let's turn our attention now to problems involving counting trees. Although there is no simple formula for determining the number of nonisomorphic spanning trees of a given order, if we place labels on the vertices, we are able to introduce a measure of control on the situation. We say two graphs G_1 and G_2 are *identical* if $V(G_1) = V(G_2)$ and $E(G_1) = E(G_2)$. Now we consider the question of determining the number of nonidentical spanning trees of a given graph (that is, on a given number of vertices). Say $G = (V, E)$ and for convenience we let $V = \{ 1, 2, \ldots, p \}$. For $p = 2$, there is only one tree, namely K_2. For $p = 3$, there are three such trees (see Figure 3.3.1).

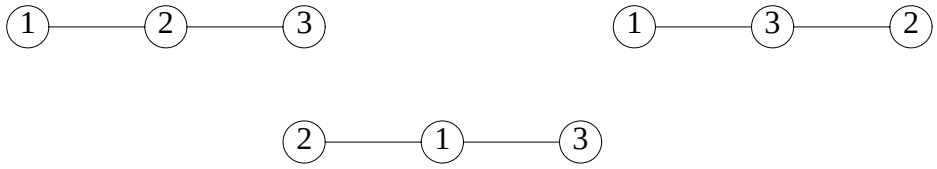


Figure 3.3.1. The spanning trees on $V = \{ 1, 2, 3 \}$.

Cayley [1] determined a simple formula for the number of nonidentical spanning trees on $V = \{ 1, 2, \ldots, p \}$. The proof presented here is from Prüfer [9]. This result is known as Cayley's tree formula.

Theorem 3.3.1 (Cayley's tree formula). The number of nonidentical spanning trees on p distinct vertices is p^{p-2}.

Proof. The result is trivial for $p = 1$ or $p = 2$ so assume $p \geq 3$. The strategy of this proof is to find a one-to-one correspondence between the set of spanning trees of G and the p^{p-2} sequences of length $p - 2$ with entries from the set $\{ 1, 2, \ldots, p \}$. We demonstrate this correspondence with two algorithms, one that finds a sequence corresponding to a tree and one that finds a tree corresponding to a sequence. In what follows, we will identify each vertex with its label. The algorithm for finding the

sequence that corresponds to a given tree is:

1. Let $i \leftarrow 1$.
2. Let $j \leftarrow$ the end vertex of the tree with smallest label. Remove j and its incident edge $e = jk$. The ith term of the sequence is k.
3. If $i = p - 2$ then halt; else $i \leftarrow i + 1$ and go to 2.

Since every tree of order at least 3 has two or more end vertices, step 2 can always be performed. Thus, we can produce a sequence of length $p - 2$. Now we must show that no sequence is produced by two or more different trees and that every possible sequence is produced from some tree. To accomplish these goals, we show that the mapping that assigns sequences to trees also has an inverse.

Let $w = n_1, n_2, \ldots, n_{p-2}$ be an arbitrary sequence of length $p - 2$ with entries from the set V. Each time (except the last) that an edge incident to vertex k is removed from the tree, k becomes the next term of the sequence. The last edge incident to vertex k may never actually be removed if k is one of the final two vertices remaining in the tree. Otherwise, the last time that an edge incident to vertex k is removed, it is because vertex k has degree 1, and, hence, the other end vertex of the edge was the one inserted into the sequence. Thus, $deg_T\ k = 1 +$ (the number of times k appears in w). With this observation in mind, the following algorithm produces a tree from the sequence w:

1. Let $i \leftarrow 1$.
2. Let j be the least vertex such that $deg_T\ j = 1$. Construct an edge from vertex j to vertex n_i and set $deg_T\ j \leftarrow 0$ and $deg_T\ n_i \leftarrow deg_T\ n_i - 1$.
3. If $i = p - 2$, then construct an edge between the two vertices of degree 1 and halt; else set $i \leftarrow i + 1$ and go to step 2.

It is easy to show that this algorithm selects the same vertex j as the algorithm for producing the sequence from the tree (Chapter 3, exercise 17). It is also easy to see that a tree is constructed. Note that at each step of the algorithm, the selection of the next vertex is forced and, hence, only one tree can be produced. Thus, the inverse mapping is produced and the result is proved. □

Example 3.3.1. Prüfer mappings. We demonstrate the two mappings determined in the proof of Cayley's theorem. Suppose we are given the tree T of Figure 3.3.2.

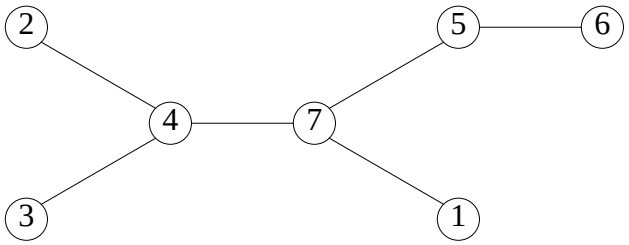


Figure 3.3.2. The tree *T*.

Among the leaves of *T*, vertex 1 has the minimum label, and it is adjacent to 7; thus, $n_1 = 7$. Our next selection is vertex 2, adjacent to vertex 4, so $n_2 = 4$. Our tree now appears as in Figure 3.3.3.

Figure 3.3.3. The tree after the first two deletions.

The third vertex selected is 3, so $n_3 = 4$. We then select vertex 4; thus, $n_4 = 7$. Finally, we select vertex 6, setting $n_5 = 5$. What remains is just the edge from 5 to 7; hence, we halt. The sequence corresponding to the tree *T* of Figure 3.3.2 is 74475.

To reverse this process, suppose we are given the sequence $s = 74475$. Then we note that:

$$\deg 1 = 1,\ \deg 2 = 1,\ \deg 3 = 1,\ \deg 4 = 3,$$
$$\deg 5 = 2,\ \deg 6 = 1,\ \deg 7 = 3.$$

According to the second algorithm, we select the vertex of minimum label with degree 1; hence, we select vertex 1. We then insert the edge from 1 to $n_1 = 7$. Now set $deg\ 1 = 0$ and $deg\ 7 = 2$ and repeat the process. Next, we select vertex 2 and insert the edge from 2 to $n_2 = 4$:

Figure 3.3.4. The reconstruction after two passes.

Again reducing the degrees, $deg\ 2 = 0$ and $deg\ 4 = 2$. Next, we select vertex 3 and insert the edge from 3 to $n_3 = 4$ (see Figure 3.3.5).

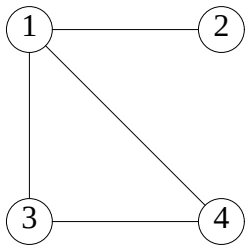
Figure 3.3.5. The reconstruction after three passes.

Now, select vertex 4 and insert the edge from 4 to $n_4 = 7$. This is followed by the selection of vertex 6 and the insertion of the edge from 6 to $n_5 = 5$. Finally, since $i = p - 2$, we end the construction by inserting the edge from 5 to 7, which completes the reconstruction of T. □

An alternate expression for the number of nonidentical spanning trees of a graph is from Kirchhoff [5]. This result uses the $p \times p$ *degree matrix* $C = [\ c_{ij}\ ]$ of G, where $c_{ii} = deg\ v_i$ and $c_{ij} = 0$ if $i \neq j$. This result is known as the matrix-tree theorem. For each pair (i, j), let the matrix B_{ij} be the $n - 1 \times n - 1$ matrix obtained from the $n \times n$ matrix B by deleting row i and column j. Then det B_{ij} is called the *minor of* B at position (i, j) and, $(-1)^{i+j}$ det B_{ij} is called the *cofactor of* B at position (i, j).

Theorem 3.3.2 (The matrix-tree theorem) Let G be a nontrivial graph with adjacency matrix A and degree matrix D. Then the number of nonidentical spanning trees of G is the value of any cofactor of $D - A$.

Example 3.3.2. Consider the following graph:



We can use the matrix-tree theorem to calculate the number of nonidentical spanning trees of this graph as follows. The matrices

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \qquad D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

are easily seen to be the adjacency matrix and degree matrix for this graph, while

$$(D - A) = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}.$$

Thus,

$$\det (D - A_{11}) = \det \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} = \det \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & -1 & 2 \end{bmatrix}$$

$$= -1 \det \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 2 \\ 0 & 0 & 3 \end{bmatrix} = 3.$$

These three spanning trees are easily found since the triangle has three spanning trees. □

Section 3.4 Directed Trees

As with connectivity, directed edges create some additional complications with trees. Initially, we need to decide exactly what we want a directed tree to be. For our purposes, the following definition is most useful: A *directed tree* $T = (V, E)$ has a distinguished vertex r, called the *root*, with the property that for every vertex $v \in V$, there is a directed $r - v$ path in T and the underlying undirected graph induced by V is also a tree. As with trees, directed trees have many possible characterizations. We consider some of them in the next theorem.

If there is an edge e in a digraph D with the property that for some pair of vertices u, v in D, e lies on every $u - v$ path, then we say that e is a *bridge* in D.

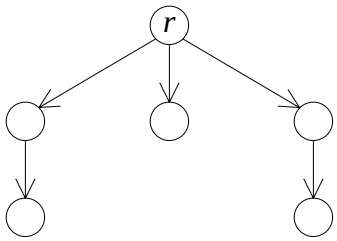


Figure 3.4.1. A directed tree with root *r*.

Theorem 3.4.1 The following conditions are equivalent for the digraph $T = (V, E)$:

1. The digraph T is a directed tree.
2. The digraph T has a root r, and for every vertex $v \in V$, there is a unique $r - v$ path in T.
3. The digraph T has a root r with $id\ r = 0$, and for every $v \neq r$, $id\ v = 1$, and there is a unique directed $(r - v)$–path in T.
4. The digraph T has a root r, and for every $v \in V$, there is an $r - v$ path and every arc of T is a bridge.
5. The graph underlying T is connected, and in T, there is a vertex r with $id\ r = 0$ and for every other vertex $v \in V$, $id\ v = 1$.

Proof. Our strategy is to show the following string of implications: 1 => 2 => 3 => 4 => 5 => 1.

To see that 1 => 2, assume that T has a root r, there are paths from r to v for every $v \in V$ and the underlying graph of T is a tree. Since there is an $r - v$ path in T and since the underlying graph is a tree, this $r - v$ path must be unique.

To see that 2 => 3, assume that T has a root r and a unique directed path from r to every vertex $v \in V$. Suppose that $e = u \rightarrow r$ is an arc of T. Since there is a directed path from r to u, the arc e completes a directed cycle containing r. But then there are at least two paths from r to r, namely the trivial path and the path obtained by following the arcs of this cycle. This contradicts the hypothesis; hence, $id\ r = 0$. Now, consider an arbitrary vertex $v \neq r$. Clearly, $id\ v > 0$ since there is a directed $r - v$ path in T. Suppose that $id\ v > 1$; in fact, suppose that $e_1 = v_1 \rightarrow v$ and $e_2 = v_2 \rightarrow v$ are two arcs into v. Note that T contains a directed $r - v_1$ path P_1 and a directed $r - v_2$ path P_2. By adding the arc e_1 to P_1 and adding e_2 to the path P_2, we obtain two different

$r - v$ paths, producing a contradiction. If $v \in P_1$ (or P_2) then the segment of P_1 from r to v and the segment of P_2 followed by the arc e_2 are two different $r - v$ paths in T, again producing a contradiction.

To see that 3 => 4, note that the deletion of any arc $e = u \rightarrow v$ means that v is unreachable from r; hence, each arc must be a bridge.

To see that 4 => 5, suppose that T has root r and that every arc is a bridge. Since any arc into r can be deleted without changing the fact that there are $r - v$ paths to all other vertices v, no such arc can exist. Hence, $id\ r = 0$. If $v \neq r$, $id\ v > 0$ since there is a directed $r - v$ path in T. Suppose e_1 and e_2 are two arcs into v. Then the path P from r to v cannot use both of these arcs. Thus, the unused arc can be deleted without destroying any $r - v$ path. But this contradicts the fact that every arc is a bridge. Hence, $id\ v = 1$ for every vertex $v \neq r$.

To see that 5 => 1, assume that the graph G underlying T is connected, $id\ r = 0$ and $id\ v = 1$ for all $v \neq r$. To see that there is an $r - v$ path for any vertex v, let P_G be an $r - v$ path in G. Then P_G corresponds to a directed path in T for otherwise, some arc along P_G is oriented incorrectly and, hence, either $id\ r > 0$ or $id\ w > 1$ for some $w \neq r$. Similarly, G must be acyclic or else a cycle in G would correspond to a directed cycle in T. □

A subgraph T of a digraph D is called a *directed spanning tree* if T is a directed tree and $V(T) = V(D)$. In order to be able to count the number of nonidentical directed spanning trees of a digraph D, we need a useful variation of the adjacency matrix. For a digraph D with m arcs from vertex j to vertex k we define the *indegree matrix*, as $A_i(D) = A_i = [\ d_{jk}\ ]$, where

$$d_{jk} = \begin{cases} id\ j & \text{if } j = k \\ -m & \text{if } j \neq k. \end{cases}$$

Using the indegree matrix, we can obtain another characterization of directed trees (Tutte [11]).

Theorem 3.4.2 A digraph $T = (V, E)$ is a directed tree with root r if, and only if, $A_i(T) = [\ d_{jk}\ ]$ has the following properties:

1. The entry $d_{jj} = 0$ if $j = r$ and the entry $d_{jj} = 1$ otherwise.
2. The minor at position (r, r) of $A_i(T)$ has value 1.

Proof. Let $T = (V, E)$ be a directed tree with root r. By Theorem 3.4.1, condition (1)

must be satisfied. We now assign an ordering to the vertices of T as follows:

1. The root r is numbered 1.
2. If the arc $u \rightarrow v$ is in T, then the number i assigned to u is less than the number j assigned to v.

This numbering is done by assigning the neighbors of r the numbers $2, 3, \ldots, (1 + od\ r)$, and we continue the numbering with the vertices a distance 2 from r, then number those a distance 3 from r, etc.

The indegree matrix $A_i^* = [\ d_{jk}^*\ ]$ (with row and column ordering according to our new vertex ordering) has the following properties:

1. $d_{11}^* = 0$.
2. $d_{jj}^* = 1$ for $j = 2, 3, \ldots, |\ V\ |$
3. $d_{jk}^* = 0$ if $j > k$.

Note that A_i^* can be obtained from the original indegree matrix A_i by permuting rows and performing the same permutations on the columns. Since such permutations do not change the determinant except for sign, and since each row permutation is matched by the corresponding column permutation, the two minors are equal. The value of the minor obtained from A_i^* by deleting the first row and column and computing the determinant is easily seen to be 1.

Conversely, suppose that A_i satisfies conditions (1) and (2). By (1) and Theorem 3.4.1, either T is a tree or its underlying graph contains a cycle C. The root r is not a member of C since $id\ r = 0$ and $id\ v = 1$ for all other vertices. Thus, C must be of the form:

$$C\colon x_1, x_2, \ldots, x_a, x_1, \text{where } x_i \neq r \text{ for each } i = 1, 2, \ldots, a.$$

Any of the vertices may have other arcs going out, but no other arcs may come into these vertices. Thus, each column of A_i corresponding to one of these vertices must contain exactly one +1 (on the main diagonal) and exactly one -1, and all other entries are 0. Each row of this submatrix either has all zeros as entries or contains exactly one +1 and one -1. But then the sum of the entries on these columns is zero, and, hence, the minor at position (r, r) is zero. This contradicts condition (2), and the result is proved. □

Corollary 3.4.1 If D is a digraph with indegree matrix A_i and the minor of A_i is zero, then D is not a directed tree.

Corollary 3.4.2 The number of directed spanning trees with root r of a digraph D equals the minor of $A_i(D)$ resulting from the deletion of the rth row and column.

Proof. Define the digraph D_G obtained from G in the natural manner; that is, for every edge $e = uv$ in G, replace e by the symmetric pair of arcs $u \to v$ and $v \to u$ to form D_G. Now, let $v \in V(D_G)$ (hence, of $V(G)$). There is a one-to-one correspondence between the set of spanning trees of G and the set of directed spanning trees of D_G rooted at r. To see that this is the case, suppose that T is a spanning tree of G and suppose that $e = uv$ is an edge of T. In the directed tree T^* rooted at r, we insert the arc $u \to v$ if $d_T(u, r) < d_T(v, r)$, and we insert the arc $v \to u$ otherwise. Hence, for every spanning tree T of G, we obtain a distinct directed spanning tree T^* of D_G.

Conversely, given a directed tree T^* rooted at r, we can simply ignore the directions on the arcs to obtain a spanning tree T of G. □

Example 3.4.1. Determine the number of spanning trees rooted at vertex 1 of the digraph D of Figure 3.4.2.

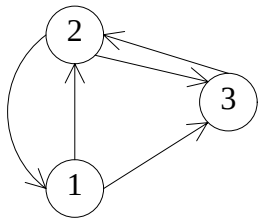


Figure 3.4.2. A digraph D with three spanning trees rooted at 1.

We begin by constructing the indegree matrix A_i of the digraph D.

$$A_i = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Then, the determinant resulting from the deletion of row 1 and column 1 can be found as:

$$\det \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} = 4 - 1 = 3.$$

Hence, D has three spanning trees rooted at vertex 1. Consulting Figure 3.4.3(a), we see these spanning trees are exactly those shown. For spanning trees rooted at vertex 2 we find that there are two such; while the number rooted at vertex 3 is one. These are shown in Figure 3.4.3(b)

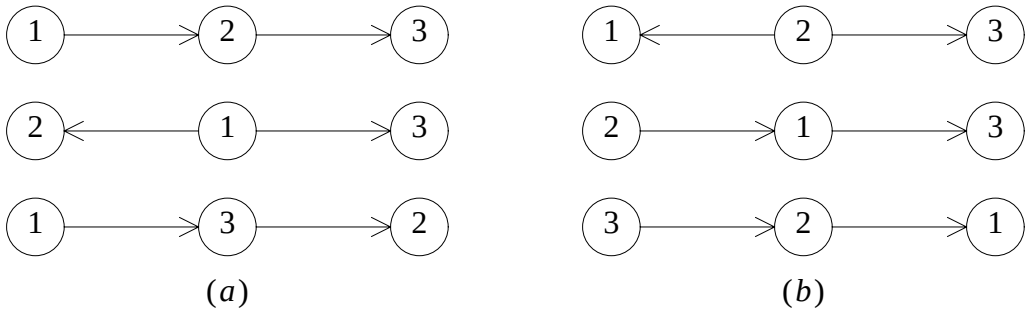


Figure 3.4.3. (a) Directed spanning trees of D rooted at 1 and (b) others.

Suppose we now consider the undirected case. Let $G = (V, E)$ be an undirected graph. We form a digraph D_G from G as follows: Let $V(D_G) = V(G)$, and for every edge $e = uv$ in G, we form two arcs, $e_1 = u \to v$ and $e_2 = v \to u$. If $r \in V(G)$, then there is a 1-1 correspondence between the set spanning trees of G and the set of directed spanning trees of D_G rooted at r. To see that this is the case, let T be a spanning tree of G. If the edge $e = uv$ is in T and if $d_T(u, r) < d_T(v, r)$, then select e_1 for the directed spanning tree T_D; otherwise, select e_2. Conversely, given a directed spanning tree T_D of D_G, it is easy to see that simply ignoring the directions of the arcs of T_D produces a spanning tree of G.

Thus, to compute the number of spanning trees of G, begin by forming D_G. If there are m arcs from vertex i to vertex j, then let

$$A_i(D_G) = \begin{cases} deg_G\ v_i & \text{if } i = j, \\ -m & \text{if } i \neq j. \end{cases}$$

Thus, applying Corollary 3.4.2 produces the desired result; the choice of r makes no difference.

As an example of this, consider the graph of Figure 3.3.2. We earlier determined it had three spanning trees. We now verify this again, using our result on digraphs. First we form D_G.

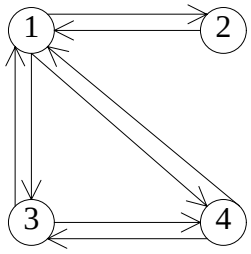


Figure 3.4.4. D_G for the graph of Example 3.3.2.

Now form

$$A_i(D_G) = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}.$$

But note $A_i(D_G)$ equals the matrix $C - A$ of Example 3.3.2 (this is no coincidence!). Hence, we must get the number of spanning trees with the choice of r making no difference, by applying the matrix-tree theorem.

Section 3.5 Optimal Directed Subgraphs

We now wish to consider a problem for digraphs similar to the minimal spanning tree problem for graphs; that is, given a weighted digraph D, we want to find a minimal weight acyclic subgraph of D. Notice that we are not restricting the subgraphs under consideration to directed trees or even to spanning subgraphs, but rather to a somewhat larger class of digraphs often called *branchings.* A subgraph $B = (V, E^*)$ of a digraph $D = (V, E)$ is a branching if B is acyclic and $id\ v \leq 1$ for every $v \in V$. If for exactly one vertex r, $id\ r = 0$ and for all other vertices v, $id\ v = 1$, then B is a directed tree with root r.

For finding optimum branchings, the following idea is useful. An arc $e = u \to v$ is called *critical* if it satisifies the following two conditions:

1. $w(e) < 0$.
2. $w(e) \leq w(e_1)$ for all other arcs $e_1 = z \to v$.

Form the arc set $E_c \subseteq E$ by selecting one critical arc entering each vertex of V. Using this arc set, we obtain the *critical subgraph* $C = (V, E_c)$. Karp [4] showed the relationship between critical subgraphs and minimum (as well as maximum) branchings. (For maximum branchings, merely reverse the inequalities in the above definition of critical arcs.)

Proposition 3.5.1 Let $C = (V, E_c)$ be a critical subgraph of a weighted digraph $D = (V, E)$. Then

1. Each vertex of C is on at most one cycle.
2. If C is acyclic, then C is a minimum weight branching.

Proof. (1) Suppose that v is on two directed cycles. Then there must be a vertex with indegree at least 2, which is a contradiction to the way we selected arcs for C.
(2) It is clear that if C is acyclic, then it is a branching. Suppose the vertex v has no negatively weighted arcs entering it in D. Then in a branching B of D, either B has no arc entering v or we can remove the arc entering v without increasing the weight of B. It is clear that C has no arc entering v. If the vertex v has negatively weighted arcs entering it in D, then the arc entering v contained in E_c is of minimum weight. Thus, no branching can have a smaller weighted arc at v and, hence, C is a minimum weight branching. □

It is possible to have many different branchings in a given digraph. In fact, some of these branchings may actually be very similar, that is, they may have a large number of arcs in common with one another. In fact, it is possible that simply deleting one arc and inserting another creates a new branching. If $B = (V, E_B)$ is a branching and if $e = u \rightarrow v$ is an arc of D that is not in B, then we say that e is *eligible relative to B* if there is an arc $e_1 \in E_B$ such that

$$B_1 = (V, E_B \cup \{ e \} - e_1)$$

is also a branching. We can characterize eligible arcs using directed paths.

Theorem 3.5.1 Let B be a branching of the digraph D and let $e = u \rightarrow v$ be an arc of D that is not in B. Then e is eligible relative to B if, and only if, there is no directed $v - u$ path in B.

Proof. Suppose there is a directed $v - u$ path in B. Then when e is inserted into B, a directed cycle is formed. The removal of the arc in B entering v (if any did exist) does not destroy this cycle. Thus, e is not eligible.

Conversely, if there is no directed $v - u$ path in B, then inserting e cannot form a directed cycle. The arc set that results from the insertion of e is not a branching if there already is an arc entering v. Removing any such arc ensures that B is a branching and, hence, e is eligible. □

There is a strong tie between the set of eligible arcs and the arc set of a branching. In fact, we can show that there is a time when they are nearly the same.

Theorem 3.5.2 Let $B = (V, E_B)$ be a branching and C a directed circuit of the digraph D. If no arc of $E(C) - E_B$ is eligible relative to B, then $|E(C) - E_B| = 1$.

Proof. Since B is acyclic, it contains no circuits. Thus, $|E(C) - E_B| \geq 1$. Let $e_1, e_2, \ldots, e_k$ be the arcs of $E(C) - E_B$ in the order in which they appear in C. Say

$$C\colon u_1, e_1, v_1, P_1, u_2, e_2, v_2, P_2, \ldots, v_k, P_k, u_k, e_k, v_k, u_1$$

is the circuit, where the P_i's are the directed paths in both B and C. Since e_1 is not eligible relative to B, by Theorem 3.5.1 there must be a directed path P^* in B from v_1 to u_1. This path leaves P_1 at some point and enters v_k and continues on to u_1, so P^* cannot enter the path p_k after v_k; if it could, B would have two arcs entering the same vertex. Similarly, e_j is not eligible relative to B and, thus, there must be a directed path from v_j to u_j in B. This path must leave P_j at some point and enter P_{j-1} at v_{j-1}. But we now see that B contains a directed circuit from v_1, along a section of P_1, to a path leading to v_k, via part of P_k, to a path leading to v_{k-1}, etc., until it finally returns to v_1. Since B is circuit-free, $k \leq 1$. □

Theorem 3.5.3 Let $C = (V, E_C)$ be a critical subgraph of the digraph $D = (V, E)$. For every directed circuit C^* in C, there exists a branching $B = (V, E_B)$ such that $|E(C^*) - E_B| = 1$.

Proof. Among all maximum branchings of D, let B be one that contains the maximum number of arcs of C. Let $e = u \to v \in E_C - E_B$. If e is eligible, then

$$E_B \cup \{ e \} - \{ e' \mid e' \text{ enters } v \text{ in } B \}$$

determines another maximum branching which contains more arcs of C than does B; thus, we have a contradiction to our choice of B. Thus, no arc of $E_C - E_B$ is eligible relative to B, and, by the last theorem, $| E(C^*) - E_B | = 1$, for every directed circuit C^* of C. □

Thus, we see that in trying to construct maximum branchings, we can restrict our attention to those branchings which have all arcs (except one per circuit) in common with a critical subgraph $C = (V, E_1)$. Edmonds [2] realized this and developed an algorithm to construct maximum branchings. His approach is as follows: Traverse D, examining vertices and arcs. The vertices are placed in the set B_v as they are examined, and the arcs are placed in the set B_e if they have been previously selected for a branching. The set B_e is always the arc set of a branching. Examining a vertex simply means selecting the critical arc e into that vertex, if one exists. We then check to see if e forms a circuit with the arcs already in B_e. If e does not form a circuit, then it is inserted in B_e and we begin examining another vertex. If e does form a circuit, then we "restructure" D by shrinking all the vertices of the circuit to a single new vertex and assigning new weights to the arcs that are incident to this new vertex. We then continue the vertex examination until all vertices of the final restructured graph have been examined. The final restructured graph contains these new vertices, while the vertices and arcs of the circuit corresponding to a new vertex have been removed. The set B_e contains the arcs of a maximum branching for this restructured digraph.

The reverse process of replacing the new vertices by the circuits that they represent then begins. The arcs placed in B_e during this process are chosen so that B_e remains a maximum branching for the digraph at hand. The critical phase of the algorithm is the rule for assigning weights to arcs when circuits are collapsed to single vertices. This process really forces our choice of arcs for B_e when the reconstruction is performed.

Let $C_1, C_2, \ldots, C_k$ be the circuits of the critical graph (V, H). Let e_i^m be an edge of minimum weight on C_i. Let $\overline{e}$ be the edge of C_i which enters v. For arcs $e = u \rightarrow w$ on C_i, define the new weight $\overline{w}$ as: $\overline{w}(e) = w(e) - w(\overline{e}) + w(e_i^m)$. Edmonds's algorithm is now presented.

Algorithm 3.5.1 Edmonds's Maximum Branching Algorithm.
Input: A directed graph $D = (V, E)$.
Output: The set B_e of arcs in a maximum branching.
Method: The digraph is shrunk around circuits.

1. $B_v \leftarrow B_e \leftarrow \varnothing$ and $i \leftarrow 0$.
2. If $B_v = V_i$, then go to step 13.
3. For some $v \notin B_v$ and $v \in V_i$, do steps 4 – 6:
4. $B_v \leftarrow B_v \cup \{ v \}$,

5. find an arc $e = x \rightarrow v$ in E_i with maximum weight.
6. If $w(e) \geq 0$, then go to step 2.
7. If $B_e \cup \{ e \}$ contains a circuit C_i, then do steps $8-10$:
8. $i \leftarrow i + 1$.
9. Construct G_i by shrinking C_i to u_i.
10. Update B_e and B_v and the necessary arc weights.
11. $B_e \leftarrow B_e \cup \{ e \}$.
12. Go to step 2.
13. While $i \neq 0$, do steps $14-17$:
14. Reconstruct G_{i-1} and rename some arcs in B_e.
15. If u_i was a root of an out-tree in B_e,
then $B_e \leftarrow B_e \cup \{ e \mid e \in C_i$ and $e \notin e_i^m \}$;
16. else $B_e \leftarrow b_e \cup \{ e \mid e \in C_i$ and $e \notin \overline{e}_i \}$.
17. $i \leftarrow i - 1$.
18. $w(B) \leftarrow \sum_{e \in B_e} w(e)$.

Section 3.6 Binary Trees

One of the principle uses of trees in computer science is in data representation. We use trees to depict the relationship between pieces of information, especially when the usual linear (list-oriented) representation seems inadequate. Tree representations are more useful when slightly more structure is given to the tree. In this section, all trees will be rooted. We will consider the trees to be *leveled*, that is, the root r will constitute level 0, the neighbors of r will constitute level 1, the neighbors of the vertices on level 1 that have not yet been placed in a level will constitute level 2, etc. With this structure, if v is on level k, the neighbors of v on level $k + 1$ are called the *children* (or *descendents*) of v, while the neighbor of v on level $k - 1$ (if it exists) is called the *parent* (or *father*, or *predecessor*) of v.

A *binary tree* is a rooted, leveled tree in which any vertex has at most two children. We refer to the descendents of v as the *left child* and *right child* of v. In any drawing of this tree, we always place the root at the top, vertices in level 1 below the root, etc. The left child of v is always placed to the left of v in the drawing and the right child of v is

always placed to the right of *v*. With this orientation of the children, these trees are said to be *ordered*. If every vertex of a binary tree has either two children or no children, then we say the tree is a *full* binary tree.

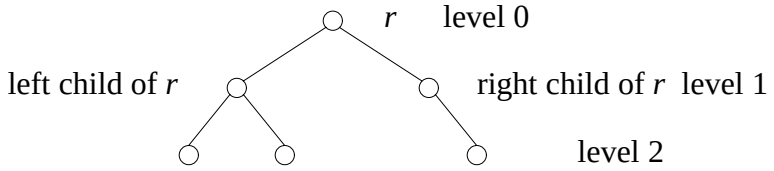


Figure 3.6.1. A binary tree.

The additional structure that we have imposed in designating a distinction between the left and right child means that we obtain distinct binary trees at times when ordinary graph isomorphism actually holds. The trees of Figure 3.6.2 are examples of this situation.

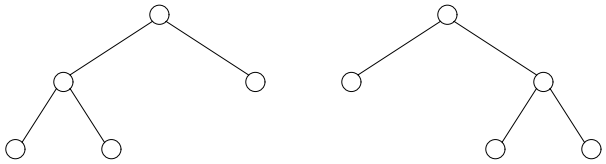


Figure 3.6.2. Two different binary trees of order 5.

The *height* of a leveled tree is the length of a longest path from the root to a leaf, or alternately, the largest level number of any vertex.

As an example of the use of binary trees in data representation, suppose that we wish to represent the arithmetic expression $a + 3b = a + 3 \times b$. Since this expression involves binary relations, it is natural to try to use binary trees to depict these relations. For example, the quantity $3b$ represents the number 3 multiplied by the value of b. We can represent this relationship in the binary tree as shown in Figure 3.6.3.

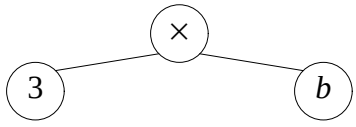


Figure 3.6.3. Representing the arithmetic expression $3b$.

Now the quantity represented by $3b$ is to be added to the quantity represented by a. We repeat the tree depiction of this relationship to obtain another binary tree (see Figure 3.6.4).

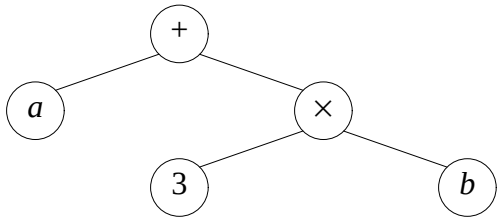


Figure 3.6.4. The expression $a + 3b$ represented using a binary tree.

Once we have a representation for data using a tree, it is also necessary to recover this information and its inherent relationships. This usually involves examining the data contained in (or represented within) the vertices of the tree. This means we must somehow visit the vertices of the tree in an order that allows us not only to retrieve the necessary information but also to understand how these data are related. This visiting process is called a *tree traversal*, and it is done with the aid of the tree structure and a particular set of rules for deciding which neighbor we visit next. One such traversal, the *inorder traversal* (or *symmetric order*) is now presented.

Algorithm 3.6.1 Inorder Traversal of a Binary Tree.
Input: A binary tree $T = (V, E)$ with root r.
Output: An ordering of the vertices of T (that is, the data contained within these vertices, received in the order of the vertices).
Method: Here "visit" the vertex simply means perform the operation of your choice on the data contained in the vertex.

procedure inorder(r)

1. If $T \neq \phi$, then
2. inorder (left child of v)
3. visit the present vertex
4. inorder (right child of v)
   end

This recursive algorithm amounts to performing the following steps at each vertex:

1. Go to the left child if possible.
2. Visit the vertex.
3. Go to the right child if possible.

Thus, on first visiting a vertex *v*, we immediately proceed to its left child if one exists. We only visit *v* after we have completed all three operations on all vertices of the subtree rooted at the left child of *v*.

Applying this algorithm to the tree of Figure 3.6.4 and assuming that visit the vertex simply means write down the data contained in the vertex, we obtain the following traversal.

First, we begin at the root vertex and immediately go to its left child. On reaching this vertex, we immediately attempt to go to its left child. However, since it has no left child, we then "visit" this vertex; hence, we write the data *a*. We now attempt to visit the right child of this vertex, again the attempt fails. We have now completed all operations on this vertex, so we backtrack (or recurse) to the parent of vertex *a*. Thus, we are back at the root vertex. Having already performed step 2 at this vertex, we now visit this vertex, writing the data +. Next, we visit the right child of the root. Following the three steps, we immediately go to the left child, namely vertex 3. Since it has no left child, we visit it, writing its data 3. Then, we attempt to go to the right child (which fails), and so we recurse to its parent. We now write the data of this vertex, namely $\times$. We proceed to the right child, attempt to go to its left child, write out its data *b*, attempt to go to the right child, recurse to $\times$ and recurse to the root. Having completed all three instructions at every vertex, the algorithm halts. Notice that the data were written as $a + 3 \times b$. We have recovered the expression.

Two other useful and closely related traversal algorithms are the *preorder* and *postorder* traversals. The only difference between these traversals is the order in which we apply the three steps. In the preorder traversal, we visit the vertex, go to the left child and then go to the right child. In the postorder traversal, we go to the left child, go to the right child and, finally, visit the vertex. Can you write the preorder and postorder algorithms in recursive form?

Another interesting application of binary trees concerns the transmission of coded data. If we are sending a message across some medium, such as an electronic cable, the characters in the message are sent one at a time, in some coded form. Usually, that form is a binary number (that is, a sequence of 1s and 0s). Since speed is sometimes important, it will be helpful if we can shorten the encoding scheme as much as possible,

while still maintaining the ability to distinguish between the characters. An algorithm for determining binary codes for characters, based on the frequency of use of these characters, was developed by Huffman [3]. Our aim is to assign very short code numbers to frequently used characters, thereby attempting to reduce the overall length of the binary string that represents the message.

As an example of Huffman's construction, suppose that our message is composed of characters from the set { a, b, c, d, e, f } and that the corresponding frequencies of these characters is (13, 6, 7, 12, 18, 10). Huffman's technique is to build a binary tree based on the frequency of use of the characters. More frequently used characters appear closer to the root of this tree, and less frequently used characters appear in lower levels. All characters (and their corresponding frequencies) are initially represented as the root vertex of separate trivial trees. Huffman's algorithm attempts to use these trees to build other binary trees, gradually merging the intermediate trees into one binary tree. The vertices representing the characters from our set will be leaves of the Huffman tree. The internal vertices of the tree will represent the sum of the frequencies of the leaves in the subtree rooted at that vertex.

For example, suppose we assign each of the characters of our message and its corresponding frequency to the root vertex of a trivial tree. From this collection of trees, select the two trees with roots having the smallest frequencies. In case of ties, randomly select the trees from those having the smallest frequencies. In this example the frequencies selected are 6 and 7. Make these two root vertices the left and right children of a new vertex, with this new vertex having frequency 13 (Figure 3.6.5). Return this new tree to our collection of trees.

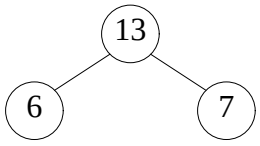


Figure 3.6.5. The first stage of Huffman's algorithm.

Again, we choose from the collection of trees the two trees with roots having the lowest frequencies, 10 and 12, and again form a larger tree by inserting a new vertex whose frequency is 22 and that has vertex 10 and vertex 12 as its left and right children, respectively. Again, return this tree to the collection. Repeating this process a third time, we select vertices 13 and 13. Following the construction, we obtain the tree of Figure 3.6.6.

We return this tree to the collection and again repeat this process. The next two roots selected have frequencies 18 and 22. We build the new tree and return it to the collection. Finally, only the roots 26 and 40 remain. We select them and build the tree shown in Figure 3.6.7. In addition to the tree we constructed, we also place a value of 0 on the edge from any vertex to its left child and a value of 1 on any edge from a vertex to its right child. Note that this choice is arbitrary and could easily be reversed.

We can read the code for each of the characters by following the unique path from the root 66 to the leaf representing the character of interest. The entire code is given in Table 3.6.1.

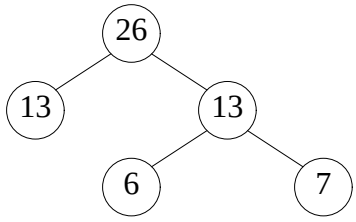


Figure 3.6.6. The new tree formed.

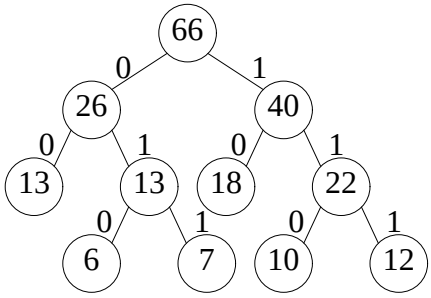


Figure 3.6.7. The final Huffman tree.

Next, suppose we are presented with an encoded message string, say, for example, a string like:

01011101100000101010110.

Assuming this encoded string was created from the Huffman tree of our example, we can decode this string by again using the Huffman tree. Beginning at the root, we use the next digit of the message to decide which edge of the tree we will follow. Initially, we follow the 0 edge from vertex 66 to vertex 26, then the 1 edge to vertex 13 and then the 0 edge to vertex 6. Since we are now at a leaf of the Huffman tree, the first character of the

message is the letter *b*, as it corresponds to this leaf.

character	code
a	00
b	010
c	011
d	111
e	10
f	110

Table 3.6.1 The Huffman codes for the example character set.

Return to the root and repeat this process on the remaining digits of the message. The next three digits are 111, the code that corresponds to d, followed in turn by 011 (c), 00 (a), 00 (a), 010 (b), 010 (b), 10 (e) and 110 (f). Thus, the encoded message was bdcaabef.

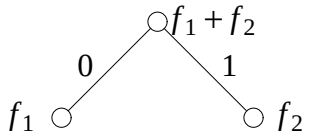
Algorithm 3.6.2 Construction of a Huffman Tree.

Input: Ordered frequencies $(f_1, f_2, \ldots, f_n)$ corresponding to the characters $(a_1, a_2, \ldots, a_n)$.

Output: A Huffman tree with leaves corresponding to the frequencies above.

Method: From a collection of trees, select the two root vertices corresponding to the smallest frequencies. Then insert a new vertex and make the two selected vertices the children of this new vertex. Return this tree to the collection of trees and repeat this process until only one tree remains.

1. If $n = 2$, then halt, thereby forming the tree:



2. If $n > 2$, then reorder the frequencies so that f_1 and f_2 are the two smallest frequencies. Let T_1 be the Huffman tree resulting from the algorithm being recursively applied to the frequencies $(f_1 + f_2, f_3, \ldots, f_n)$ and let T_2 be the Huffman tree that results from calling the algorithm recursively on the frequencies (f_1, f_2). Halt the algorithm with the tree that results from substituting T_2 for

some leaf of T_1 (which has value $f_1 + f_2$).

Note that the Algorithm does not produce a unique tree. If several frequencies are equal, their positions in the frequency list and, hence, their positions in the tree can vary. Can you find a different Huffman tree for the example data? What conditions would produce a unique Huffman tree?

Huffman trees are in a sense the optimal structure for binary encoding. That is, we would like to show that the Huffman code minimizes the length of encoded messages, with characters and frequencies matching those used in the construction of the Huffman tree. Our measure of the efficiency of the code is called the *weighted path length* of the coding tree and is defined to be: $\sum_{1 \le i \le n} f_i l_i$, where f_i is the frequency of the ith letter and l_i is the length of the path from the root in the Huffman tree to the vertex corresponding to the ith letter. The weighted path length is a reasonable measure to minimize since, when this value is divided by $\sum_{i=1}^{n} f_i$ (that is, the number of characters being encoded), we obtain the average length of the encoding per character.

Theorem 3.6.1 A Huffman tree for the frequencies $(f_1, f_2, \ldots, f_n)$ has minimum weighted path length among all full binary trees with leaves

$f_1, f_2, \ldots, f_n$.

Proof. We proceed by induction on n, the number of frequencies. If $n = 2$, the weighted path length of any full binary tree is $f_1 + f_2$, as is the value we obtain from the algorithm. Now, suppose that $n \ge 3$ and assume that the result follows for all Huffman trees with fewer than n leaves.

Reorder the frequencies so that $f_1 \le f_2 \le, \ldots, \le f_n$ (if necessary). Since there are only a finite number of full binary trees with n leaves, there must be one, call it T, with minimum weighted path length. Let x be an internal vertex of T whose distance from the root r is a maximum (for the internal vertices).

If f_1 and f_2 are not the frequencies of the leaves of T that are children of x, then we can exchange the frequencies of the children of x, say f_i and f_j, with f_1 and f_2 without increasing the weighted path length of T. This follows since $f_i \ge f_1$ and $f_j \ge f_2$ and this interchange moves f_i closer to the root and f_1 farther away from the root. But T has minimum weighted path length, and thus, its value cannot decrease. Hence, there must be a tree with minimum weighted path length that does have f_1 and f_2 as the frequencies of the children of an internal vertex that is a maximum distance from the root (again, this

maximum is taken over internal vertices only).

Finally, it remains for us to show that this tree is minimal if, and only if, the tree that results from deleting the leaves f_1 and f_2 is also minimal for the frequencies that remain, namely $f_1 + f_2, f_3, \ldots, f_n$.

Note that the value at any internal vertex equals the sum of the frequencies of the leaves of the subtree rooted at that internal vertex. Thus, the weighted path length of a Huffman tree is the sum of the values of the internal vertices of the tree. If W_T is the weighted path length of T and W^* is the weighted path length of the graph $T^* = T - \{ f_1, f_2 \}$, then $W_T = W^* + f_1 + f_2$. But now we see that T has minimum weighted path length if, and only if, T^* does, since their weights differ by the constant $f_1 + f_2$. Thus, if either tree failed to be minimal, both trees would fail to be minimal. □

We can also verify that Huffman's algorithm assigns the shortest codes to the most frequently used characters.

Theorem 3.6.2 If $c_1, c_2, \ldots, c_n$ are the binary codes assigned by Huffman's algorithm to the characters with frequencies $f_1, f_2, \ldots, f_n$, respectively, and if $f_i < f_j$, then $length(c_i) \geq length(c_j)$.

Proof. Suppose instead that $length(c_j) > length(c_i)$. If we assign the code word c_i to the character with frequency f_j and c_j to the character with frequency f_i and leave all other code assignments the same, then we obtain a new code with minimum weighted path length less than the Huffman code. To see that this is the case, note that if W_H is the minimum weighted path length for the original Huffman tree and W^* is the new minimum weighted path length for the modified code, then

$$\begin{aligned} W_H - W^* &= (f_i\ length(c_i) + f_j\ length(c_j)) \\ &\quad - (f_i\ length(c_j) - f_j\ length(c_i)) \\ &= (f_j - f_i)(length(c_j) - length(c_i)) > 0, \end{aligned}$$

contradicting the fact that W_H is minimum. □

Can you verify that Huffman's algorithm has time complexity $0(n^2)$?

Section 3.7 More About Counting Trees – Using Generating Functions

In this section, our fundamental goal is to introduce one of the principal tools used to count combinatorial objects, namely generating functions.

To begin with, what is a generating function? Suppose we have a sequence of numbers (c_i); then the power series $\sum_{i=0}^{\infty} c_i x^i$ is called the *generating function* for the sequence. Examples from calculus and algebra demonstrate that a power series can be used to approximate a function of x. Our goal is to use the tools already developed for series to help us count objects. Our purpose is not to present a complete development of generating functions, but rather to show the reader already familiar with series how they can be used for counting.

Suppose we consider the question of the number of binary trees T possible on a set of n vertices. Let's call this number b_n. Clearly, if $n = 1$, $b_1 = 1$. For $n > 1$, we can select one vertex to be the root, and the remaining $n - 1$ vertices can be partitioned into those in the left and right subtrees of T. If there are j vertices in the left subtree and $n - 1 - j$ vertices in the right subtree, then the number of binary trees we can form on n vertices depends on the number of ways we can build the left and right subtrees; that is b_n depends on b_j and b_{n-1-j}. To determine exactly how many binary trees b_n there are, we must sum the products $b_j b_{n-1-j}$ for $j = 0, 1, \ldots, n - 1$. This gives us the following *recurrence relation:*

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-1} b_0.$$

We now illustrate how to solve this recurrence relation using generating functions.

Suppose our generating function $g(x) = \sum_{i=0}^{\infty} b_i x^i$. When we square $g(x)$, we obtain the following:

$$(g(x))^2 = g(x) \times g(x) = \sum_{n \geq 0} \left(\sum_{0 \leq j \leq n} b_j \; b_{n-j} \right) x^n$$

$$= \sum_{n \geq 0} (b_0 b_n + b_1 b_{n-1} + b_2 b_{n-2} + \cdots + b_n b_0) \; x^n.$$

But on careful examination we see that the coefficient of x^n in $g^2(x)$ is nothing but b_{n+1}. Hence, we obtain another relationship, namely

$$1 + x g^2(x) = g(x).$$

But this equation is quadratic in $g(x)$, and so it yields the solution

$$g(x) = \frac{1 - \sqrt{1 - 4x}}{2x}.$$

Now, to obtain a series expansion for $g(x)$, we use the binomial generating function

$$(1 + z)^r = 1 + rz + r\left(\frac{r-1}{2}\right) z^2 + r\left(\frac{(r-1)(r-2)}{6}\right) z^3 + \cdots$$

We write the coefficients of this power series using the definition of generalized binomial coefficients, where for any real number r and integer k,

$$\binom{r}{k} = \frac{r(r-1)(r-2)\cdots(r-k+1)}{k!}, \text{for } k > 0,$$

while it has value zero if $k < 0$ and 1 when $k = 0$. Thus, we can rewrite the binomial generating function as

$$(1 + z)^r = \sum_{k \geq 0} \binom{r}{k} z^k.$$

Substituting this function in our expression for $g(x)$, we obtain

$$g(x) = \frac{1}{2x}\left(1 - \sum_{k \geq 0} \binom{1/2}{k} (-4x)^k\right).$$

Changing the dummy variable k to $n + 1$ and simplifying yields

$$\begin{aligned} g(x) &= \frac{1}{2x}\left(1 - \sum_{n+1 \geq 0} \binom{1/2}{n+1} (-4x)^{n+1}\right) \\ &= \frac{1}{2x} + \sum_{n+1 \geq 0} \binom{1/2}{n+1} (-1)^n 2^{2n+1} x^n \\ &= \sum_{n \geq 0} \binom{1/2}{n+1} (-1)^n 2^{2n+1} x^n. \end{aligned}$$

But this process yields coefficients of x^n which match b_n in our original definition of $g(x)$. Thus, we have that,

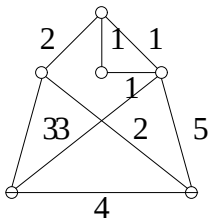
$$b_n = \binom{1/2}{n+1} (-1)^n 2^{2n+1}.$$

Using exercise 26, we can simplify this expression for b_n to obtain:

$$b_n = \frac{1}{n+1}\binom{2n}{n}.$$

Exercises

1. Show that if $T = (V, E)$ is a tree, then for any $e \in E$, $T - e$ has exactly two components.
2. Show that any connected graph on p vertices contains at least $p - 1$ edges.
3. Show that if T is a tree with $\Delta(T) \geq k$, then T has at least k leaves.
4. In a connected graph G, a vertex v is called *central* if $\max_{u \in V(G)} d(u, v) = rad(G)$. Show that for a tree T, the set of central vertices consists of either one vertex or two adjacent vertices.
5. Show that the sequence $d_1, d_2, \ldots, d_p$ of positive integers is the degree sequence of a tree if, and only if, the graph is connected and $\sum_{i=1}^{p} d_i = 2(p - 1)$.
6. Show that the number of end vertices in a nontrivial tree of order n equals $2 + \sum_{deg\ v_i \geq 3} (deg\ v_i - 2)$.
7. Determine the time complexity of Kruskal's algorithm.
8. What happens to the time complexity of Kruskal's Algorithm if we do not presort the edges in nondecreasing order of weight?
9. Apply Kruskal's algorithm to the graph:



10. Apply Prim's algorithm to the graph of the previous problem.

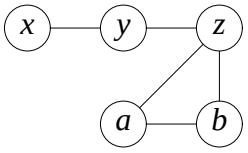
11. Prove that a graph G is acyclic if, and only if, every induced subgraph of G contains a vertex of degree one at most.

12. Characterize those graphs with the property that every connected subgraph is also an induced subgraph.

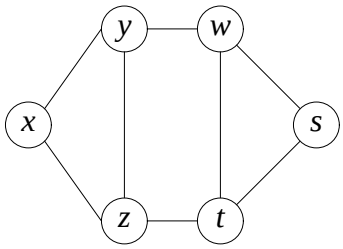
13. Find the binary tree representations for the expressions $4x - 2y$, $(3x + z)(xy - 7z)$, and $\sqrt{b^2 - 4ac}$.

14. Perform a preorder, postorder and inorder traversal on the trees constructed in the previous problem.

15. Determine the number of nonidentical spanning trees of the graph below. Before you begin your computation, make an observation about this graph that will simplify the calculations.



16. (a) Find the number of nonidentical spanning trees of the graph



(b) Determine the number of spanning trees in the above graph using D_G, the directed graph obtained from G.

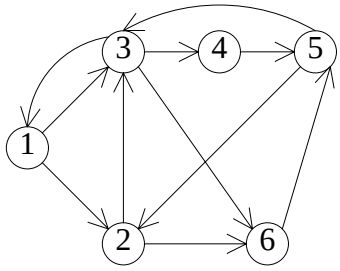
17. Find the spanning trees on the set { 1, 2, 3, 4 }.

18. Using the proof of Cayley's theorem, determine the sequences of length $p - 2$ on { 1, 2, 3, 4 } that correspond to any two of the trees found in the previous problem.

19. Show that the Prüfer algorithm for creating a tree from a sequence selects the same vertex as the algorithm for producing the sequence.

20. Use the matrix-tree theorem to prove Cayley's tree formula.

21. Find the number of directed spanning trees with root 3 in the following digraph:



How many are rooted at vertex 5? How many are rooted at vertex 2?

22. Given a graph G with adjacency matrix A and degree matrix C, show that the matrices $C - A$ and $A_i(D_G)$ are equal.

23. Show that the number of trees with m labeled edges and no labels on the vertices is $(m + 1)^{m-2}$.

24. Determine the number of trees that can be built on p labeled vertices such that one specified vertex is of degree k.

25. By *contracting an edge* $e = uv$, we mean removing e and identifying the vertices u and v as a single new vertex. Let $num_T(G)$ denote the number of spanning trees of the graph G. Show that the following recursive formula holds:

$$num_T(G) = num_T(G - e) + num_T(G \; o \; e)$$

where $G \; o \; e$ means the graph obtained from G by contracting the edge e. Hint: Interpret what $num_T(G - e)$ and $num_T(G \; o \; e)$ really count.

26. Show that the algorithm for determining the number of spanning trees of G implied by the previous problem takes exponential time.

27. Determine the Huffman tree and code for the alphabet $\{ x, y, z, a, b, c \}$ with corresponding frequencies $(3, 8, 1, 5, 4, 4)$.

28. What is the minimum weighted path length for the Huffman tree you constructed in the previous problem?

29. Using the definition of $\binom{1/2}{n+1}$, show that

$$\binom{1/2}{n+1} = \frac{(-1)^n}{2^{n+1}} \times \frac{(1)(3)(5)\cdots(2n-1)}{(1)(2)(3)\cdots(n+1)}.$$

Also show that $(1)(3)(5)\cdots(2n-1) = \frac{(2n)!}{2^n n!}$ and use these two facts to obtain the formula for b_n.

References

1. Cayley, A., A Theorem on Trees. *Quart. J. Math.,* 23(1889), 376 – 378.
2. Edmonds, J., Optimum Branchings. *J. of Res. of the Nat. Bureau of Standards,* 71B(1967), 233 – 240.
3. Huffman, D. A., A Method for the Construction of Minimum Redundancy Codes. *Proc. IRE,* No. 10, 40(1952), 1098 – 1101.
4. Karp, R. M., A Simple Derivation of Edmonds Algorithm for Optimum Branchings. *Networks,* 1(1972), 265 – 272.
5. Kirchhoff, G., Über die Aufflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird. *Ann. Phy. Chem.,* 72(1847), 497 – 508.
6. Kruskal, J. B. Jr., On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proc. Amer. Math. Soc.,* 7(1956), 48 – 50.
7. Ore, O., *Theory of Graphs.* Amer. Math. Soc. Publ., Providence (1962).
8. Prim, R. C., Shortest Connection Networks and Some Generalizations. *Bell System Tech. J.,* 36(1957), 1389 – 1401.
9. Prüfer, H., Neuer Beweis eines satzes über Permutationen. *Arch. Math. Phys.,* 27(1918), 742 – 744.
10. Standish, T. A., *Data Structure Techniques.* Addison-Wesley Pub. Co., Reading, Mass. (1980).
11. Tutte, W. T., The Dissection of Equilateral Triangles into Equilateral Triangles. *Proc. Cambridge Phil. Soc.,* 44(1948), 463 – 482.

Chapter 4
Networks

Section 4.1 Flows

You are designing an oil pipeline. The pipeline is to have a pumping station p and a receiving station r. Further, to minimize the possibility that pipe repairs will completely halt the flow of oil, there will be several routes. There will be a northern route and a southern route, and each will have two intermediate pumping stations, n_1 and n_2 on the northern route and s_1 and s_2 on the southern route. Finally, there will also be a pipeline from s_2 to n_1, from s_2 to n_2 and from n_2 to s_1. Our pipeline will also cross several county borders, and each county has a different restriction on the size of underground pipes. Thus, the amount of oil we can pump on any section of the pipeline differs. For example, from p to n_1 the maximum capacity is 4000 barrels of oil per hour (b/h). Figure 4.1.1 shows the restrictions on pipe capacity for each section of pipe in our system. Under all these conditions, what is the maximum rate at which oil can reach the receiving station r?

capacity (b/h) -->	n_1	n_2	s_1	s_2	r
p	4000		3000		
n_1		3000			
n_2			1700		4500
s_1				5400	
s_2	1000	2000			3200

Figure 4.1.1. Pumping capacity restrictions along pipeline sections.

Once again we wish to form a graph model for our problem. Clearly, the pipes and pumping stations can be modeled by a digraph. Our problem is to devise a method for determining how the pipe capacity restrictions and the pipeline interconnections (the graph structure) restrict the rate at which oil reaches r.

Before we attempt to solve this problem, we should note several fairly natural restrictions. Clearly, no pipe can carry more oil then its capacity allows. Also, all intermediate stations can only pump out as much oil as they receive. We now formulate our problem in more mathematical terms.

A *network* N is a 5-tuple $N = (V, E, s, t, c)$ where (V, E) is a digraph with distinguished vertices s and t called the *source* and *sink*, respectively. Each arc e of N is

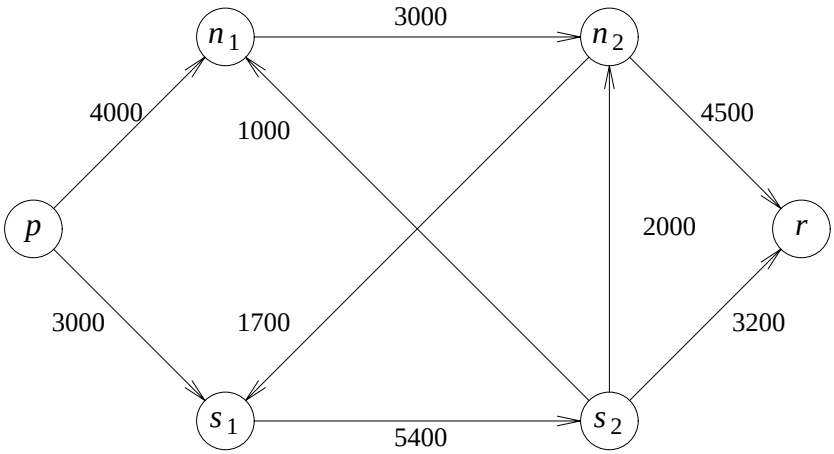


Figure 4.1.2. The pipeline network model with capacities shown.

assigned a nonnegative real number $c(e)$, called the *capacity* of the arc; that is, c is a function which assigns nonnegative real numbers to the arcs of N. Here, $in(v)$ $(out(v))$ denotes the set of all arcs entering (leaving) the vertex v.

A (legal) *flow* f in N is a mapping from the arc set E to the real numbers such that:

1. **(Capacity constraint)** $f(e) \leq c(e)$ for every $e \in E$

2. **(Flow conservation)** for each vertex v other than s and t,

$$0 = \sum_{e \in in(v)} f(e) - \sum_{e \in out(v)} f(e). \tag{4.1}$$

Clearly, a flow is a mapping that describes the movement (or flow) of material along the arcs of the network, while the capacity is a mapping that describes the maximum amount that can move along the arcs. Further, legal flows conform to the natural restrictions we determined earlier. That is, the flow on an arc never exceeds the capacity and flow conservation states that all that flows into a vertex (other than s or t) also flows out of that vertex.

Note that we allow s and t to be arbitrarily chosen within the digraph, and, thus, $in(s)$ and $out(t)$ need not be empty. We further note that loops never add anything to a network, because what flows out along the loop immediately flows back into the same vertex. Also, parallel arcs really add nothing to our study, since we can view the flows (and capacities) on these arcs as actually occurring on only one arc whose flow (and capacity) is the sum of the flows (and capacities) on all the parallel arcs. These restrictions do, however, allow us to be sure that $|E| \leq |V|$ $(|V| - 1)$.

Given a network N with flow f, the *total flow* F (sometimes called the value of the flow) is defined as:

$$F = \sum_{e \in in(t)} f(e) - \sum_{e \in out(t)} f(e). \tag{4.2}$$

That is, F is the net flow into the sink t. Our problem is to maximize F.

Ford and Fulkerson [6] were the first to study the maximum flow problem computationally. They established a number of results and developed an algorithm for finding the maximum flow. Strictly speaking, their original method was not an algorithm in that it was not always guaranteed to terminate. Edmonds and Karp [5] modified this method to ensure that the process would terminate. In fact, they showed that the modified algorithm had complexity $O(|V|\ |E|^2)$.

Many others have studied the maximum flow problem. Dinic [4] developed a $O(|V|^2|E|)$ algorithm and later Karzanov [7] developed a $O(|V|^3)$ improvement on Dinic's technique. In 1978, Malhorta, Kumar and Maheshwari [8] simplified this approach while maintaining the time complexity of $O(|V|^3)$. The fastest known algorithm is that of Sleator [9]. This algorithm has complexity $O(|V||E|\log(|V|))$; however, it requires some sophisticated data structure techniques and is beyond the scope of this text. In the remaining sections of this chapter, we will investigate flows in networks and the algorithms of Ford and Fulkerson, Dinic, and Malhorta, Kumar and Maheshwari.

Section 4.2 The Ford and Fulkerson Approach

The Ford and Fulkerson approach to finding the maximum total flow in a network N is based on the idea of improving the present flow along some "path" between the source s and sink t. Once this is done, we repeat the process on N with its modified flow. We continue to repeat the process until we cannot find a path whose flow can be improved. This technique has come to be known as the *augmenting path* technique.

Before we develop an algorithm for the augmenting path technique, it will be helpful to observe an interesting relationship between the maximum flow and the capacity of a set of arcs. Let S be a subset of V such that $s \in S$ and $t \in \overline{S} = V - S$. Recall that we denote by $out(S)$ the set of all arcs from vertices of S to vertices in $\overline{S}$ and, similarly, by $in(S)$ the set of all arcs from vertices in $\overline{S}$ to vertices in S. Simply put, $out(S)$ is the set of arcs that start anywhere in the set S, but end outside S, while $in(S)$ is the set of arcs that originate outside of S but end in S. Together these two sets constitute all arcs

between the vertex sets S and $\overline{S}$, and $out(S) \cup in(S)$ is called the *cut* determined by S. An interesting and important fact is that the total flow F can actually be measured at any cut in the network N.

Lemma 4.2.1 Given a network $N = (V, E, s, t, c)$ with flow f, then for every $S \subseteq V$ such that $s \in S$ and $t \in \overline{S}$,

$$F = \sum_{e \in out(S)} f(e) - \sum_{e \in in(S)} f(e). \tag{4.3}$$

Proof. In order to verify this equation, consider the sum of equation (4.2) and all equations on flow conservation (4.1) for vertices $v \in \overline{S} - \{ t \}$. The result of this sum has F on the left-hand side. To see what happens on the right-hand side, we must consider an arc from x to y. If x and y are both in S, then $f(e)$ never appears on the right-hand side of any equation in the sum. If x and y are both in $\overline{S}$, then $f(e)$ appears positively once and negatively once, and, hence, has no effect on the right-hand side. If $x \in S$ and $y \in \overline{S}$, then $f(e)$ appears positively on the right-hand side of the equation for y, and it appears in no other equation. If $x \in \overline{S}$ and $y \in S$, then $f(e)$ appears negatively on the right-hand side of the equation for x and in no other equation. Each of these cases agrees with equation (4.3) and, thus, the result is verified. □

For a set S of vertices, we call $c(S) = \sum_{e \in out(S)} c(e)$ the *capacity* of the cut determined by S. From what we have already seen about cuts and the total flow, one should expect that the capacity $c(S)$ is related to the total flow F. That this is indeed the case is our next result.

Proposition 4.2.1 Given a network N, for every flow f with total flow F and for every $S \subseteq V$,

$$F \leq c(S).$$

Proof. By Lemma 4.2.1 we know that

$$F = \sum_{e \in out(S)} f(e) - \sum_{e \in in(S)} f(e).$$

But by the capacity constraint, we know that $0 \leq f(e) \leq c(e)$ for each $e \in E$. Thus, by maximizing the positive term and minimizing the negative term we get that

$$F \leq \sum_{e \in out(S)} c(e) - 0 = c(S). \quad \square$$

A very important corollary of Proposition 4.2.1 is now easy to obtain.

Corollary 4.2.1 Given a network N with flow f and $S \subseteq V$ such that S contains s but not t, if $F = c(S)$, then F is a maximum and the cut determined by S is of minimum capacity.

Proof. Let F^* be a maximum flow and let K determine a cut of minimum capacity. Then, applying Proposition 4.2.1 and the natural relationships of these quantities, we see that

$$F \leq F^* \leq c(K) \leq c(S).$$

But since $F = c(S)$, we see that equality must hold throughout the above expression, and so F must be a maximum and S must have minimum capacity. □

We now wish to examine the Ford and Fulkerson process for finding the maximum flow. Their idea, again, was to use paths (not necessarily directed paths) from the source to the sink that were not being "optimally" used. The direction of the arcs on such a path would be used to help determine how to modify the present flow along the path in order to "push" more flow to the sink.

If the arc $e = x \rightarrow y$ is on such an $s - t$ path (we follow the direction on e from x to y) and we wish to use e to push more flow to t, then e presently must not be up to capacity. That is, $f(e) < c(e)$ must hold so that there is room for improvement on e. The amount of improvement is, of course, limited to $\Delta(e) = c(e) - f(e)$. The value $\Delta(e)$ is called the *slack* of e. If $f(e) = c(e)$, we say the arc e is *saturated.* If, instead, $e = y \rightarrow x$, then in order to increase the flow from s to t, we must cancel some of the flow into x on this arc (as the flow on e is away from t). Thus, there must already be some flow ($f(e) > 0$) on e if we are to accomplish our goal.

An *augmenting path* is a (not necessarily directed) path from s to t that can be used to increase the flow from s to t. As we attempt to find an augmenting path, we will label the vertices of N as we examine them. Initially, we label s, and then we label every vertex v for which we can find an augmenting $s - v$ path. When we finally label t, we have found an augmenting $s - t$ path. This path is then used to increase the total flow in N, and the process is begun again. If at some point, we cannot find any vertex to label, then no augmenting path exists and the present value of the flow is maximum.

In performing the labeling process, a *forward labeling* of vertex v using arc e (where e is directed from u to v) is done when u is labeled and v is not labeled and $c(e) > f(e)$. Here the label assigned to v is e^+. If the arc e is used for forward labeling, let

$\Delta(e) = c(e) - f(e)$. A *backward labeling* of vertex v using arc e (where $e = v \rightarrow u$) is done when u is labeled and v is not labeled and $f(e) > 0$. The label assigned to v is e^-, and in this case $\Delta(e) = f(e)$.

We now state the fundamental result about augmenting paths. The proof of Theorem 4.2.1 is left as an exercise.

Theorem 4.2.1 In a network N with flow f, the total flow F is maximum if, and only if, no augmenting path from s to t exists.

We now state the Ford and Fulkerson algorithm [6].

Algorithm 4.2.1 The Ford and Fulkerson Algorithm.
Input: A network $N = (V, E, s, t, c)$ and a flow f. (Initially, we usually choose $f(e) = 0$ for every arc e.)
Output: A modified flow f^* or the answer that the present total flow is maximum.
Method: Augmenting paths.

1. Label s with * and leave all other vertices unlabeled.
2. Find an augmenting path P from s to t.
3. If none exists,
   then halt, noting that the present flow is maximum;
   else compute and record the slack of each arc of P and compute the minimum such slack λ. Now, refine the flow f by adding λ to f for all forward arcs of P and subtracting λ from f for all backward arcs of P.
4. Set $f^* = f$ and repeat this process for N and the new flow f^*.

We are now capable of answering the question raised at the beginning of the chapter. We will use the Ford and Fulkerson algorithm for finding augmenting paths to determine a maximum flow in the oil pipeline model. We find the first augmenting path, say p, s_1, s_2, n_2, r, has minimum slack 2000. All arcs are forward, so we push 2000 units of additional flow along each arc. We picture this in Figure 4.2.1.

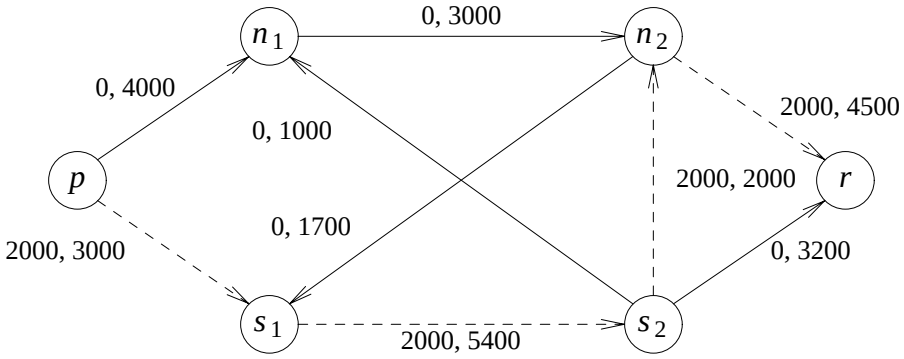


Figure 4.2.1. After the first augmenting path is found.

A second augmenting path is p, n_1, n_2, s_2, r, which includes a backward labeling of s_2. The minimum slack is again 2000. The resulting network and flow are shown in Figure 4.2.2.

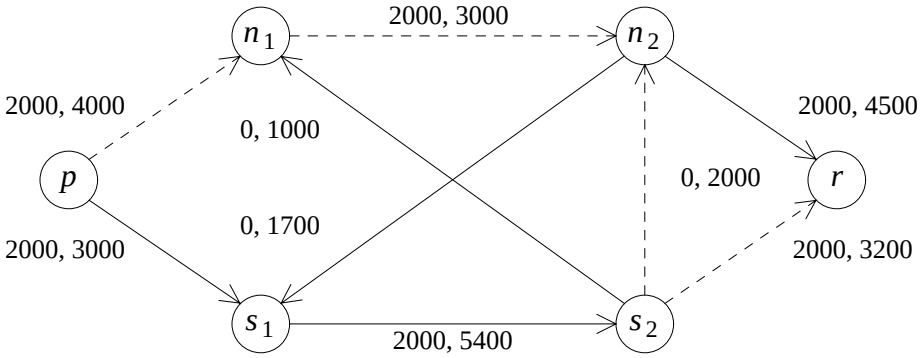


Figure 4.2.2. After the second augmenting path is found.

We continue with a third augmenting path p, n_1, n_2, r. This path has a minimum slack of 1000. We update the flows accordingly to obtain the network in Figure 4.2.3.

Finally, the augmenting path p, s_1, s_2, r, with minimum slack 1000, is found (Figure 4.2.4). At this stage the arc $p \to s_1$ is up to capacity, while $p \to n_1$ is not. However, the remaining arcs at n_1 are either up to capacity or directed into n_1 and without positive flow. Thus, no further augmenting paths can exist.

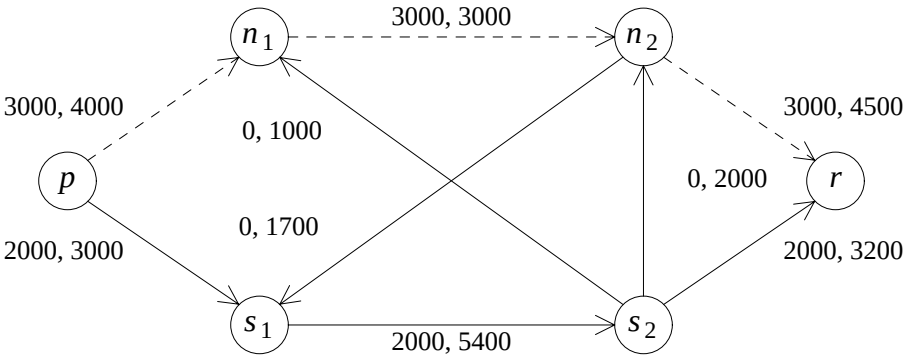


Figure 4.2.3. After the third augmentation.

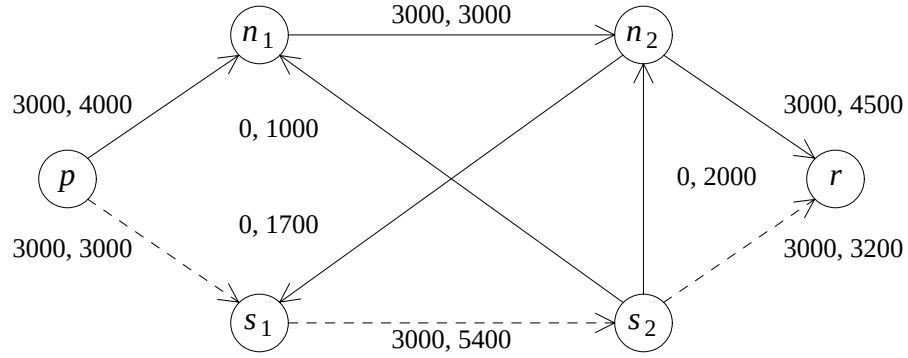


Figure 4.2.4. After the final augmentation, $F = 6000$.

We should make several points here. We found the maximum flow $F = 6000$, and the Ford and Fulkerson algorithm terminated. If both $f(e)$ and $c(e)$ are integers for each e, then since the algorithm only adds or subtracts the slack, it continues to produce only integer flows. Thus, once t is labeled and an augmenting path has been found, we will increase the value of the flow by at least 1. Since F is bounded above by $c(S)$ for any S, the process must terminate.

At first glance, you might argue that since all computer storage is limited, capacities and flows are effectively rational numbers anyway. Thus, the algorithm should terminate by an argument similar to that for integer flows and capacities. However, there is a fundamental problem with this kind of thinking. This problem is most easily seen in Figure 4.2.5, where the arc $x \to y$ has capacity 1 and all other arcs have capacity C. By alternating the augmenting paths from s, x, y, t to s, y, x, t, we need $2C$ augmentations

to find the maximum flow. Since we can make C as large as we like, the algorithm can still fail in practice.

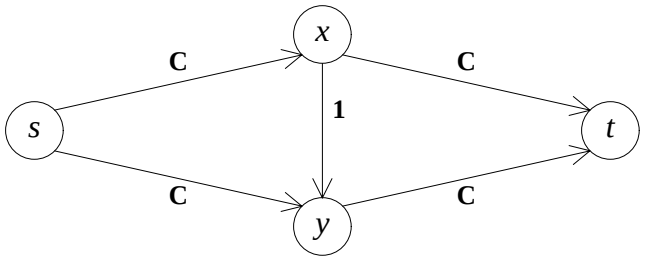


Figure 4.2.5. A network taking $2C$ augmentations.

Recall that we said that the Ford and Fulkerson algorithm might not terminate. Ford and Fulkerson showed that their procedure might take an infinite number of steps if all the capacities were irrational. Their example is somewhat complex, and we shall not examine it here. The important point in their example, as well as in the oil pipeline example and the last example, was that we did not do a very effective job of finding the augmenting paths. In fact, in each example we did more work than we should have. This is essentially what Ford and Fulkerson did in their example. They delayed as long as possible actually saturating arcs. Edmonds and Karp [5] were able to show that if one uses a breadth-first search in the labeling algorithm and always selects a shortest augmenting path, then the algorithm will terminate in at most $O(|V|^2|E|)$ steps (even if irrational capacities are allowed).

We now present an algorithm for finding the augmenting path. In this algorithm, consider the term *scan* to imply that a breadth-first search is being done.

Algorithm 4.2.2 Finding an Augmenting Path.
Input: A network N and a flow f.
Output: An augmenting path P or a message that none exists.
Method: A modified breadth-first search.

1. Label s with an *.
2. If all labeled vertices have been scanned,
   then halt, noting that no augmenting path exists; hence, the present flow is maximum,
   else find a labeled but unscanned vertex v and scan as follows: For each

$e = vu \in out(v)$, if $c(e) - f(e) > 0$ and u is unlabeled, label u with e^+. For each $e = uv \in in(v)$, if $f(e) > 0$ and u is unlabeled, then label u with e^-.

3. If t has been labeled,
   then starting with t, use the labels to backtrack to s along an augmenting path. The label at each vertex indicates its predecessor in the path. When you reach s output the path and halt; else repeat step 2.

Note that the labeling process also indicates the type of labeling that was done. Thus, the appropriate computation for the slack can be done as we trace back from t while finding the augmenting path.

We conclude this section with the most famous result concerning networks (Ford and Fulkerson [6]). As we shall see, it is closely related to many other results in graph theory, as well as results in such other areas as linear programming and combinatorics.

Theorem 4.2.2 (The Max Flow–Min Cut Theorem) In a network N, the maximum value of a flow equals the minimum capacity of a cut.

Proof. Let F be a maximum total flow and S be a cut of minimum capacity. It follows from Proposition 4.2.1 that $F \leq c(S)$. To show that the reverse inequality also holds, let flow f have maximum flow F. Then by Theorem 4.2.1, there can be no augmenting $s - t$ paths in N. Let C be the set of all vertices v such that there is an augmenting path from s to v. Clearly, t is not in C. By the definition of an augmenting path and the definition of C, we have that for all vertices $v \in C$ and for all vertices $w \in V - C$, the flow from v to w equals the capacity of the cut determined by C that separates v and w. Further, the flow from w to v is zero. If either the flow from v to w is less than the capacity of a cut that separates them or the flow from w to v is positive, we could find an augmenting path from s to w. Thus, we have found a cut, namely C, with the same capacity as the maximum flow, and by Corollary 4.2.1, this cut must be of minimum capacity. □

Section 4.3 The Dinic Algorithm and Layered Networks

Dinic's approach [4] to finding a maximum flow starts with some legal flow (possibly zero everywhere) and repeatedly attempts to improve it using augmenting paths. When no further improvement can be found, the total flow is maximum. The difference in Dinic's approach is the way in which the network is viewed and the restrictions this view

places on the way augmenting paths are formed.

The main idea behind this approach is the construction of a *layered network.* Layers are special subsets of the vertices. What is special about the layers is their structure and that they are determined by the present flow. Recall that in the Ford and Fulkerson approach, we concentrated on forward arcs $e = u \rightarrow v$ such that $f(e) < c(e)$ or on backward arcs $e = v \rightarrow u$ such that $f(e) > 0$. We again want to address such arcs, which we now designate as *useful arcs.* Denote the useful arcs between layers L_i and L_{i+1} by U_{i+1}. We build layers based on a breadth-first search using only useful arcs. Thus, as arcs become saturated, we have fewer and fewer useful arcs to use in relayering the network. Dinic's algorithm for construction of a layered network is as follows:

Algorithm 4.3.1 The Layering Algorithm.
Input: A network N and flow f.
Output: A sequence of layers of vertices $L_0, L_1, \ldots, L_d$ or the message that the present flow is maximum.
Method: A modified BFS using only useful arcs.

1. $L_0 \leftarrow \{s\}$ and $i \leftarrow 0$.
2. Set $T \leftarrow \{ v \mid v \notin \bigcup_{j=0}^{i} L_j$ and there is $e = u \rightarrow v$ or $v \rightarrow u$ such that e is useful $\}$.
3. If $T = \varnothing$, say that the present flow is maximum and halt.
4. If $t \in T$,
   then $k \leftarrow i + 1$ and $L_k \leftarrow \{t\}$ and halt with the layers $L_0, \ldots, L_k$;
   else $L_{i+1} \leftarrow T$, set $i \leftarrow i + 1$ and go to step 2.

In constructing the layered network, we examine each arc two times at most (once in each of two consecutive layers), hence, we note that the time complexity of the layering algorithm is $O(|E|)$.

In the sequence of layers $L_0, \ldots, L_d$, consecutive layers are joined only by useful arcs. We seek a *maximal flow* f^* in this layered network. By this we mean a flow f^* such that for every path $s = v_0, e_1, v_1, \ldots, e_d, v_d = t$, where $v_i \in L_i$ and $e_i \in U_{i+1}$, there is at least one saturated arc e. That is, every feasible augmenting path with vertices in consecutive layers has an arc whose flow is at capacity.

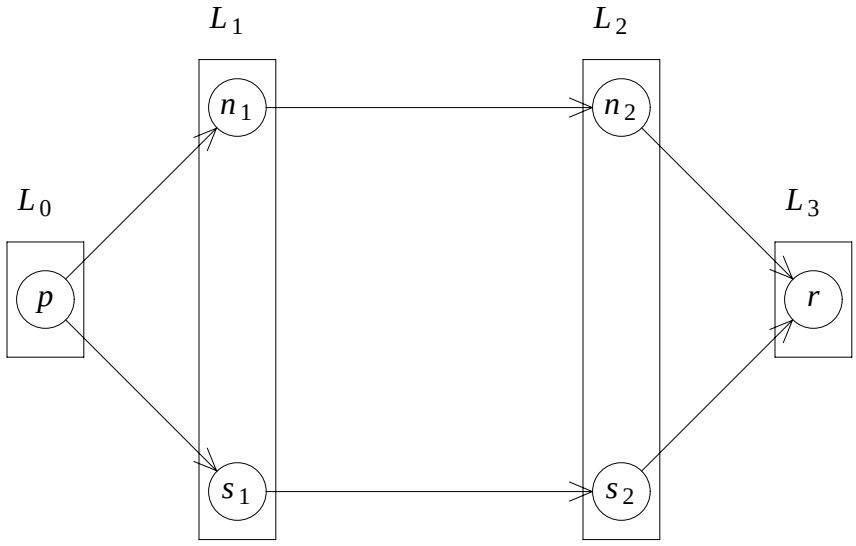


Figure 4.3.1. First layering of the network of Figure 4.1.2.

Typically, the process of finding a layered network, then finding a maximal flow on the layered network and improving the original flow, is called a *phase*. The important point about phases is that we can determine a bound on the number of phases we need to perform in order to find a maximum flow. This bound will allow us to compute a bound on the time complexity of Dinic's algorithm.

To do this, let the *length* of a layered network be the index of the final layer. Clearly, this is also a measure of the length of an augmenting path in the layered network. Denote the length of the layered network obtained in the jth phase by $len(j)$. Also, denote by $L_a(b)$, the ath layer in the bth phase.

Theorem 4.3.1 If phase $m + 1$ is not the final phase, then $len(m + 1) > len(m)$, and, hence, the number of phases is at most $|V| - 1$.

Proof. Suppose phase $m + 1$ is not the last phase of the algorithm. Then, in the $(m + 1)$-st layered network, there must be a path

$$P\colon\ s = v_0,\ e_1,\ v_1,\ e_2\ ,\ \ldots,\ e_{len(m+1)},\ v_{len(m+1)} = t.$$

We first assume that all the vertices of P also appear in the m-th layered network. We claim that if $v_j \in L_k(m)$, then $j \geq k$. That is, a vertex in layer k of phase m must come from layer j of phase $m + 1$ where $j \geq k$.

To prove this claim, we use induction on j. For $j = 0$ (recall $v_0 = s$), the claim is clearly true. Thus, assume the result is true for all vertices through j and assume $v_{j+1} \in L_i(m)$. If $i \leq j + 1$, the induction is trivial. But if $i > j + 1$, then the arc e_{j+1} has not been used in the *mth* phase since it is not even in the $m-th$ layered network, in which only arcs between adjacent layers appear. If e_{j+1} has not been used but is useful from v_j to v_{j+1} at the start of phase $m + 1$, then it was useful from v_j to v_{j+1} in the beginning of phase m. Thus, v_{j+1} cannot belong to $L_i\ (m)$ (by the algorithm). Now, in particular, $t = v_{len(m+1)}$ and $t \in L_{len(m)}(m)$. Hence, from our induction we conclude that $len(m + 1) \geq len(m)$. However, equality cannot hold, because in this case the entire path is in the *mth* layered network, and if all its arcs are still useful at the start of phase $m + 1$, then the flow in phase m was not maximal, and we have a contradiction.

We now assume that not all the vertices of P appear in the mth layered network. Then let $e_{j+1} = v_j v_{j+1}$ be the first arc such that for some b, $v_j \in L_b(m)$; but v_{j+1} is not in the *mth* layered network. Thus, e_{j+1} was not used in phase m. Since it is useful in the beginning of phase $m + 1$, it was also useful in the beginning of phase m. The only possible reason for v_{j+1} not to belong to $L_{b+1}(m)$ is that $b + 1 = len(m)$. By the previous paragraph, $j \geq b$. Thus, $j + 1 \geq len(m)$, and therefore,

$$len(m + 1) > len(m).$$

It is now clear that the number of phases is at most $|V| - 1$. □

We now present Dinic's algorithm. To do this, we make use of a simple data structure known as a stack. Recall that a stack is a last in-first out device. That is, whatever item is last placed on the stack is the first one to be removed from the stack (just like a stack of trays in a cafeteria). The act of placing X on the stack ST will be denoted ST <== X, while removing X from the top of the stack will be denoted X <== ST. These are the only two operations we allow on the stack.

Algorithm 4.3.2 Dinic's Maximal Flow Algorithm.

Input: A layered network N with $f(e) = 0$ and e marked unblocked for each arc e.

Output: A maximal flow on N.

1. Let $v \leftarrow s$ and empty the stack ST.
2. If all arcs to the next layer are blocked, then
   if $s = v$, then halt and note that the present flow is maximal;

else e <== ST (say $e = uv$), mark e as "blocked," $v \leftarrow u$, and repeat step 2.

3. Choose an unblocked arc $e = vu$ with u in the next layer, ST <== e and let $v \leftarrow u$. If v does not equal t, then go to step 2.

4. The edges on ST form an augmenting path P. Find the minimum slack Δ of an arc on P. For every arc e on P, set $f(e) \leftarrow f(e) + \Delta$ and if $f(e) = c(e)$, mark e as "blocked." Go to step 1.

It is clear that in Algorithm 4.3.2, an arc is marked as blocked only if no later augmenting path can use it. Thus, when this algorithm halts, the flow is maximal. The number of arcs examined between two declarations of arc blocking is at most $|V| - 1$. Since no arc is ever unblocked, the number of arcs examined is bounded by $|V|\,|E|$ Thus, the algorithm for finding a maximal flow in the layered network is bounded by $O(|V|\,|E|)$. Further, since the number of layerings is bounded (Theorem 4.3.1), the entire process is bounded by $O(|V|^2\,|E|)$.

Section 4.4 Layered Networks and Potential

In the last section, we developed an algorithm to block a layered network of length d of all augmenting paths of length d. We accomplished this by finding a flow which saturated at least one arc on every $s - t$ path. We call such a flow a *blocking flow* for the layered network. (Since Dinic developed the idea of a layered network, all new flow algorithms have concentrated on improved methods for finding blocking flows.)

Malhorta, Kumar, and Maheshwari [8] developed a simple algorithm to accomplish this in $O(|V|^3)$ time. We can use their algorithm in place of Dinic's Maximal Flow Algorithm to create a modified phase.

The Malhorta, Kumar and Maheshwari improvement centers on the idea of saturating arcs more rapidly. Given a layered network N with layers $L_0, L_1, \ldots, L_d$, define the *potential* of a vertex to be

$$p(v) = \min \{ \sum_{e \in out(v)} c(e), \sum_{e \in in(v)} c(e) \}.$$

Intuitively, the potential is the amount of flow that we can force through a vertex. We are then interested in a vertex of minimum potential so that we can saturate an arc (or hopefully more) at this vertex. Thus, if $p(w) = \min_{v \in V} p(v)$, we call w a *minimum vertex.*

Theorem 4.4.1 If a layered network N has minimal potential p^*, then N admits a flow with value p^*.

Proof. Let w be a vertex with minimum potential p^*. We begin by distributing a flow of p^* across the arcs from s to L_1, saturating them one by one. Since the flow into any vertex $v \in L_1$, is no greater than p^*, the flow into v can be distributed in the same manner among arcs from v to vertices in L_2. Repeating this process, we can push the flow all the way to $L_d = \{ t \}$. Clearly, at each layer, p^* enters and p^* leaves; thus the flow into t is p^*. □

The proof of Theorem 4.4.1 describes an algorithm for pushing a flow of $p(w) = p^*$ from s to t. Call this algorithm PUSH($p(w)$). Using this algorithm, we can now present the Malhorta, Kumar and Maheshwari algorithm for creating a blocking flow in the layered network.

Algorithm 4.4.1 The Potential Method Blocking Flow Algorithm.
Input: A layered network N with layers $L_0, L_1, \ldots, L_d$.
Output: A blocking flow or the message that the present flow is maximum.

1. Set all potentials to zero.
   While $\{s, t\} \cup V \neq \varnothing$, do the following:
2. Find a vertex w of minimum potential.
3. PUSH($p(w)$).
4. Update the arc capacities and potentials.
5. Delete saturated arcs and vertices without paths to both s and t.
6. endwhile

Section 4.5 Variations on Networks

In this section we consider several variations on the standard flow maximization problem. In the first variation, we wish to restrict the capacities that are possible. One goal is to see if this restriction does anything to the complexities of the algorithms we have studied. Our restriction will be fairly strong; for every arc e, let $c(e) = 1$. We call such networks $0 - 1$ *networks.* (Note that in applications of networks to other types of

graph problems or to other areas of discrete mathematics, this restriction is a useful and natural one.)

Suppose that N is a $0-1$ network with flow f. Recall from our earlier discussion of integer flows on networks with integer capacities that the flows remain integer in value. Thus, when we apply any algorithm to modify the present flow, arcs with flow zero will have their flows increase to 1 and, hence, become saturated. For example, in Dinic's algorithm, each time we find an augmenting path, all arcs on this path become blocked. In case the last arc leads to a dead end, we backtrack on this arc and it becomes blocked (that is, if the last arc does not lead to t, backtrack and consider this arc blocked). Thus, the total number of arc traversals is bounded by $|E|$, and, hence, an entire phase has time complexity $O(|E|)$. Since we know that the number of phases is at most $|V|-1$, the time complexity of Dinic's algorithm for $0-1$ networks can be reduced to $O(|V|\,|E|)$. Our goal now is to show that an even stronger improvement is possible.

Let N be a $0-1$ network with an integer flow function f. We define a new network NZ from N and f as follows: Let $V(NZ)=V(N)$ and if $e=uv\in E(N)$ and $f(e)=0$, then $e\in E(NZ)$, while if $e=vu\in E(N)$ and $f(e)=1$, then $e'=uv\in E(NZ)$. Thus, it is clear that $|E(N)|\geq|E(NZ)|$ and the useful arcs of N (in their direction of usefulness) are all arcs of NZ. Let M_N denote the maximal flow in the network N, F_N denote the present flow in N and $c_N(S)$ denote the capacity of S in the network N.

Lemma 4.5.1 Given a $0-1$ network N with flow f and the corresponding network NZ, then $M_{NZ}=M_N-F_N$.

Proof. Let $S\subseteq V$ with $s\in S$ and $t\in\bar{S}$. From the definition of NZ we have that

$$c_{NZ}(S)=|out_{NZ}(S)|=\sum_{e\in out_N(S)}(1-f(e))+\sum_{e\in out_N(\bar{S})}f(e).$$

However,

$$F_N=\sum_{e\in out_N(S)}f(e)-\sum_{e\in out_N(\bar{S})}f(e).$$

Thus,

$$c_{NZ}(S)=|out_N(S)|-F_N=c_N(S)-F_N.$$

This implies that a minimum cut of N corresponds to a minimum cut of NZ, that is, it is defined by the same vertex set S. By the max flow–min cut theorem, the capacity of a minimum cut of N is M_N and the capacity of a minimum cut of NZ is M_{NZ}. Thus, the lemma is proved. □

Lemma 4.5.2 The length of the layered network for the $0-1$ network defined by N and zero flow everywhere is at most $\dfrac{|E|}{M_N}$.

Proof. Since $f(e) = 0$ for every $e \in E$, the useful arcs are all forward. Thus, every U_i is equal to $out_N(S)$ where $S = \bigcup_{j=0}^{i-1} L_j$. Thus, by Lemma 4.5.1, we see that $M_N \leq |U_i|$ Summing this last inequality for every $i = 1, 2, \ldots, len(N)$ we get that $(len(N))\ M_N \leq |E|$ and the result follows. □

Theorem 4.5.1 For a $0 - 1$ network N, Dinic's algorithm is of time complexity $O(|E|^{3/2})$.

Proof. If $M_N \leq |E|^{1/2}$, then the number of phases is bounded by $|E|^{1/2}$, and the result follows easily. Otherwise, consider the phase during which the total flow reaches $M_N - |E|^{1/2}$. The total flow F in N, when the layered network for this phase is constructed, satisfies $F < M - |E|^{1/2}$. This layered network is identical with one constructed for NZ with zero flow everywhere. Thus, by Lemma 4.5.1, $M_{NZ} = M_N - F > |E|^{1/2}$. By Lemma 4.5.2, the length (*len*) of this layered network satisfies

$$len \leq \frac{|E|}{M_{NZ}} < \frac{|E|}{|E|^{1/2}} = |E|^{1/2}.$$

Thus, the number of phases up to this point is less than $|E|^{1/2}$, and the number of additional phases is at most $|E|^{1/2}$; thus the total number of phases is at most $2|E|^{1/2}$, and the result follows. □

Another variation of our original question is a simple generalization of the idea of a network. In all our previous problems, the flow on any arc e was at least zero. Now we wish to vary this lower bound and assume another function b assigns lower bounds on the flow that is legal on any arc. That is, suppose that $b(e) \leq f(e) \leq c(e)$ is our restriction on legal flows, where the function b assigns a real number to any arc e. We define the *generalized network GN* to be the 6-tuple $GN = (V, E, s, t, b, c)$.

In altering networks in this manner, we are faced with a new concern. Previously, it was easy to find an initial legal flow by simply taking the zero flow on each arc. Now, it is not even clear that a legal flow exists. In fact, it is easy to produce networks that have no legal flows. For example, the generalized network below with arcs labeled with the bounding pairs b, c has no legal flows.

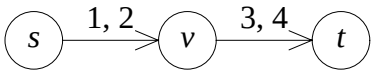


Figure 4.5.1. A generalized network with no legal flow.

Thus, the maximum flow problem for generalized networks calls for a solution in two phases. First, we must determine if the given network admits any legal flows, and if so, we must find one. Secondly, given a legal flow, we want to increase this flow until a maximum total flow is found.

Ford and Fulkerson developed a method for determining if a generalized network has a legal flow and for finding a legal flow. This method begins with a very natural idea: try to modify the network in some way that allows us to apply the techniques we have already developed. This modified network is called an *auxiliary network.* We now present Ford and Fulkerson's algorithm for constructing an auxiliary network.

Algorithm 4.5.1 Auxiliary Network Construction.
Input: A network $N = (V, E, s, t, b, c)$.
Output: An auxiliary network $\overline{N} = (\overline{V}, \overline{E}, \overline{s}, \overline{t}, \overline{b}, \overline{c})$.

1. Modify N as follows: Let $\overline{V} = \{ \overline{s}, \overline{t} \} \cup V$. The vertices $\overline{s}$ and $\overline{t}$ are called the auxiliary source and sink, respectively.
2. For every $v \in V$, construct an arc $\overline{e} = v\overline{t}$ with capacity $\overline{c}(\overline{e}) = \sum_{e \in out(v)} b(e)$. Let $\overline{b}(\overline{e}) = 0$.
3. For every $v \in V$, construct the arc $\overline{e} = \overline{s}v$ with $\overline{c}(\overline{e}) = \sum_{e \in in(v)} b(e)$. Now let $\overline{b}(\overline{e}) = 0$.
4. Modify the bounds on all arcs $e \in E$ by letting $\overline{b}(e) = 0$ and $\overline{c}(e) = c(e) - b(e)$.
5. Construct new arcs $e_1 = st$ and $e_2 = ts$ and let $\overline{c}(e_1) = \overline{c}(e_2) = \infty$ and $\overline{b}(e_1) = \overline{b}(e_2) = 0$.

Note that in the resulting auxiliary network, we regard s and t as normal vertices; hence, they must also satisfy the flow conservation constraint.

We have modified N to obtain the auxiliary network $\overline{N}$, but what have we gained?

Since each arc e in the original generalized network contributes $b(e)$ to an arc of $out(\overline{s})$ and an arc of $in(\overline{t})$, if all of $out(\overline{s})$ is saturated, then all of $in(\overline{t})$ must also be saturated. The answer to what we have gained is now provided by the following result from Ford and Fulkerson [6].

Theorem 4.5.2 The generalized network GN has a legal flow if, and only if, the maximum flow of $\overline{N}$ saturates all arcs of $out(\overline{s})$.

Proof. Suppose that a maximum flow function $\overline{f}$ of $\overline{N}$ saturates all arcs of $out(\overline{s})$. Define the following flow function f for N: For every $e \in E$, let $f(e) = \overline{f}(e) + b(e)$. Since

$$0 \le \overline{f}(e) \le \overline{c}(e) = c(e) - b(e),$$

it is clear that $b(e) \le f(e) \le c(e)$.

Now, consider $v \in V - \{ s, t \}$. Let $e_1 = \overline{s} \to v$ and $e_2 = v \to \overline{t}$ be arcs of N. Then

$$\sum_{e \in in(v)} \overline{f}(e) + \overline{f}(e_1) = \sum_{e \in out(v)} \overline{f}(e) + \overline{f}(e_2).$$

By hypothesis,

$$\overline{f}(e_1) = \overline{c}(e_1) = \sum_{e \in out(v)} b(e) \text{ and } \overline{f}(e_2) = \overline{c}(e_2) = \sum_{e \in in(v)} b(e).$$

Thus,

$$\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e);$$

hence, f is a legal flow on N.

To prove the reverse implication, we assume N has a legal flow f and we define the flow $\overline{f}$ as $\overline{f}(e) = f(e) - b(e)$ for each $e \in E$. Since $b(e) \le f(e) \le c(e)$, it is clear that

$$0 \le \overline{f}(e) \le c(e) - b(e) = \overline{c}(e).$$

Since f satisfies the conservation constraint, if we let $e_1 = \overline{s} \to v$ and $e_2 = v \to \overline{t}$ and let $\overline{f}(e_1) = \overline{c}(e_1)$ and $\overline{f}(e_2) = \overline{c}(e_2)$, then

$$\sum \overline{f}(e) + \overline{f}(e_1) = \sum \overline{f}(e) + \overline{f}(e_2),$$

and so $\overline{f}$ also satisfies flow conservation when all arcs of $out(\overline{s})$ are saturated.

Finally, the net flow out of *s* equals the net flow into *t*; hence, they will satisfy the conservation constraint if we push this flow on the arcs *st* and *ts* constructed in step 5 of the algorithm. □

We now have an approach to solving our two problems. Given the network $N = (V, E, s, t, b, c)$, we construct the auxiliary network $\overline{N}$. Then we apply one of our algorithms for finding a maximum flow in $\overline{N}$. If $out(\overline{s})$ is saturated, we use the techniques of the proof of Theorem 4.5.2 to construct an initial legal flow in *N*.

Next, we must maximize the flow in *N*. With a few minor modifications, the Ford and Fulkerson algorithm can be adjusted to work on *N*. The necessary adjustments center on the backward labelings. A backward labeling is done for the arc $e = v \rightarrow u$ if *u* is labeled and *v* is not labeled and $f(e) > b(e)$. Otherwise, the algorithm remains intact.

All the results about the Ford and Fulkerson algorithm can be shown to hold under these conditions, provided we also modify the definition of cut capacity to be: $c(S) = \sum c(e) - \sum b(e)$.

Section 4.6 Connectivity and Networks

In this section we examine the use of network techniques in obtaining results about connectivity of graphs and digraphs. In particular, we begin with the relationship between Menger's theorem and the max flow–min cut theorem. One should immediately notice a very similar style in the statements of these two results. Menger's theorem relates the maximum number of vertex disjoint paths and the minimum number of vertices in a separating set, while the Ford and Fulkerson result relates the maximum flow and the minimum capacity of a cut. Thus, both results involve the equality of two quantities, one of which is a maximum and the other a minimum. Dantzig and Fulkerson [2] showed that the max flow-min cut theorem can be used to prove Menger's theorem. We now present this proof.

Theorem 4.6.1 (Menger's Theorem) For distinct nonadjacent vertices *u* and *w* in a graph *G*, the maximum number of pairwise internally disjoint $u - w$ paths equals the minimum number of vertices in a $u - w$ separating set.

Proof. It is clear that the minimum number of vertices that separate *u* and *w* is at least as large as the maximum number of vertex disjoint $u - w$ paths, since every path from *u* to

w uses at least one vertex of the separating set.

To prove the converse, we construct a digraph D from the graph G as follows. For every $v \in V(G)$, we create two vertices v_1 and v_2 along with the arc $e^v = v_1v_2$, called an *internal arc.* For every edge $e = xy$ in G, insert two directed arcs $e' = x_2y_1$ and $e'' = y_2x_1$ in D, called *external arcs* (see Figure 4.6.1). We now define a network on D with source u_2 and sink w_1. Assign all arcs of the form $v_1 \to v_2$ a capacity of 1. Assign all other arcs an infinite capacity.

We now claim that the maximum number of disjoint $u - w$ paths in G equals the maximum flow F in the network. To see that this is indeed true, assume that there are n vertex disjoint $u - w$ paths in G. Note that each such path $u, e_1, a, e_2, b, \ldots, e_k, w$ determines a directed path in the network, namely

$$u_2, e'_1, a_1, e^a, a_2, e'_2, b_1, \ldots, w_1.$$

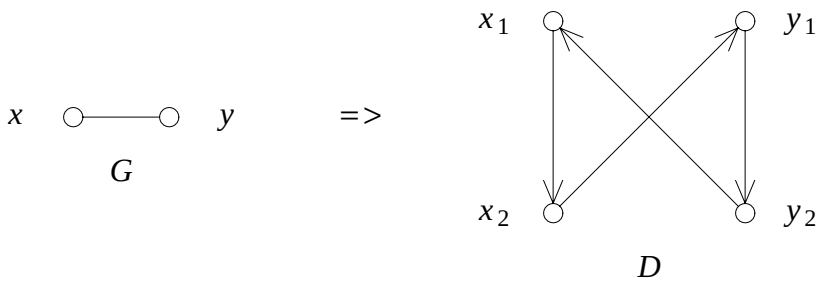


Figure 4.6.1. Construction of D.

Now let f be a flow that achieves the maximum total flow F. We can assume the flow on any arc is either 0 or 1 as the arcs of infinite capacity either enter a vertex with only one outgoing arc whose capacity is 1 or leave a vertex with only one incoming arc whose capacity is 1. Further, since the flow will stay integral, the claim is verified.

The last observation really tells us that the flow along any path can be at most 1. The flow into the sink can be decomposed into paths that each supply the sink with 1 unit of flow. Further, these paths are vertex disjoint, since the flow through any vertex is at most 1. Thus, there are F vertex disjoint paths from u_2 to w_1, and each determines a $u - w$ path in G. Hence, F is less than or equal to the maximum number of disjoint $u - w$ paths, but then we have that F equals the number of such paths.

By the max flow–min cut theorem, F equals $c(S)$ for some cut S of the network, where $u_2 \in S$ and $w_1 \in \overline{S}$. Since $c(S) = \sum_{e \in out(S)} c(e)$, the cut at S consists only of internal arcs. Every directed path from u_2 to w_1 in the network uses at least one arc of

the cut determined by S; hence, every $u - w$ path in G uses at least one vertex v such that e^v goes to $\overline{S}$. Therefore, the set

$$R = \{ v \mid v \in V \text{ and } e^v \in cut(S) \}$$

is a $u - w$ separating set. Clearly, $|R| = c(S)$. Thus, we have a $u - w$ separating set whose cardinality is F, and, thus, the minimum number of vertices in a $u - w$ separating set is at most F. This completes the proof. □

It should be clear from the proof that we can also use Dinic's algorithm for finding the connectivity of G.

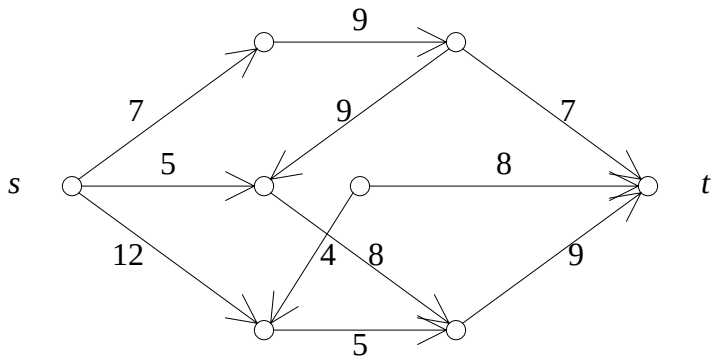
We conclude this section with a brief sketch of the proof of the edge version of Menger's theorem. The details are left to the exercises.

Theorem 4.6.2 In a graph G, the maximum number of edge disjoint $u - v$ paths equals the minimum number of edges in a $u - v$ separating set.

Proof (Sketch). Again, the idea is to form a digraph, say D from G. However, this time there is no need to split vertices, so let $V(D) = V(G)$ and let $E(D)$ contain both $e_1 = u \rightarrow v$ and $e_2 = v \rightarrow u$ whenever $e = uv \in E(G)$. Now, proceed in a fashion similar to that in the proof of Menger's theorem. □

Exercises

1. Find the maximum flow in the network below, using each of the algorithms presented in this chapter.

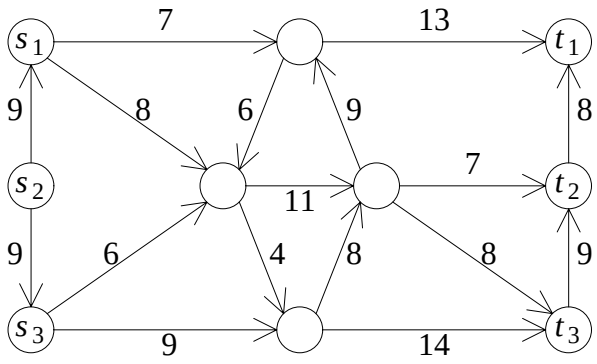


2. Use Algorithm 4.3.2 to find the maximum flow in the network of Figure 4.1.2.

3. Use Algorithm 4.4.1 to find the maximum flow in the network of Figure 4.1.2.

4. Let u and v be two vertices of a digraph $D = (V, E)$ and let $E_1 \subseteq E$ such that every $u - v$ path in D contains at least one arc of E_1. Prove that there exists a set of arcs of the form $(W, \overline{W})$ (where $W \subseteq V$) such that $u \in W, v \in \overline{W}$ and $(W, \overline{W}) \subseteq E_1$.

5. Suppose the following network has multiple sources s_1, s_2 and s_3 and they have available supplies 6, 12, and 7, respectively. Further suppose that t_1, t_2 and t_3 are all sinks with the demands 7, 12 and 6, respectively. Determine whether all these demands can be met simultaneously.

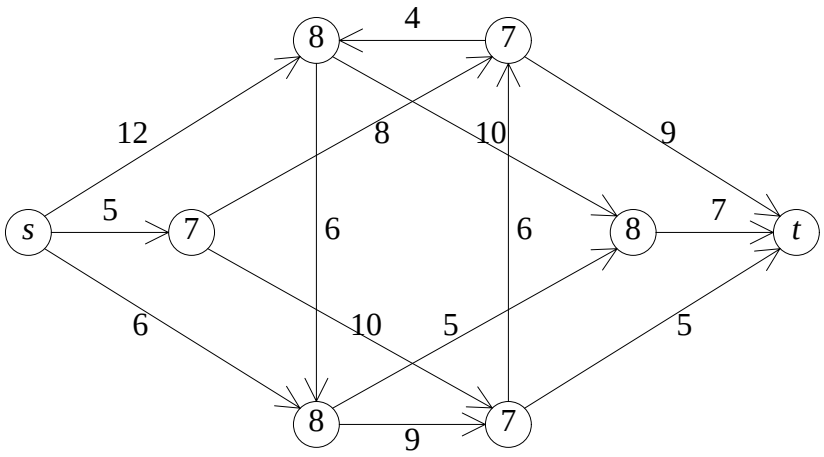


6. Let N be a network with underlying digraph D. Prove that if D contains no $s - t$ path, then the value of a maximum flow in N and the value of a minimum cut in N are both equal to zero.

7. Prove or disprove (each direction): The flows f_1 and f_2 in the network N agree on $(W, \overline{W})$ and $(\overline{W}, W)$ if, and only if, f_1 and f_2 are maximum flows in N.

8. Use networks to prove that $k(\ K_{m,n}\ ) \ = \ \min\{\ m,\ n\ \}$.

9. In the following network, in addition to the usual constraints, we have the additional constraint that each vertex (other than *s* and *t*) has an upper bound on the capacity that may flow through it. This capacity is indicated by the value inside the vertex. Find the maximum flow for this network.



10. Prove that in a network with a nonnegative lower bound on each arc, but no upper bound, there exists a legal flow if, and only if, for every arc *e*, either *e* is in a directed circuit or *e* is in a directed path from *s* to *t* (or *t* to *s*).

11. Prove that a network with both lower and upper bounds on the flow in the arcs has no legal flow if, and only if, there exists a set of vertices which includes neither the source nor the sink and is required to produce flow or absorb it.

12. A *path cover* of the vertices of a digraph $D = (V,\ E)$ (or graph) is a set of paths with the property that each vertex of D lies on exactly one path (note that a path may be trivial). Use the following network to develop an algorithm for determining the minimum number of paths necessary to cover an acyclic digraph $D = (V,\ E)$: Let $N = (V_1,\ E_1)$ where

$$V_1 = \{\ s,\ t\ \} \cup \{\ u_1, \ldots,\ u_{|V|}\ \} \cup \{\ v_1\ , \ldots,\ v_{|V|}\ \}$$

$$E_1 = \{\ s \to u_i \mid 1 \le i \le |V|\}$$
$$\cup\ \{\ v_i \to t \mid 1 \le i \le |V|\}$$
$$\cup\ \{\ u_i \to v_j \mid x_i \to x_j \in E(D)\ \}.$$

Also, set the capacities of each arc to 1. (Hint: Show that the minimum number of paths which cover V equals $|V|$, the max flow in the network.)

13. In the previous problem, was the assumption that D was acyclic necessary?

14. Suppose we ease the path cover restriction in the sense that the paths are no longer required to be vertex disjoint. Further, into the network constructed in exercise 11, insert the additional edges $\{\ u_i \to v_i \mid 1 \le i \le |V|\ \}$. Now, let the lower bound on each of these new arcs be 1 and on all other arcs be 0 and let all upper bounds be ∞. Describe an algorithm for finding the minimum number of paths covering the vertices of the original digraph D.

15. (Dilworth's theorem [3]) Two vertices are called *concurrent* if no directed path exists between them. A set of vertices is *concurrent* if its members are pairwise concurrent. Prove that the minimum number of paths which cover the vertices of D equals the maximum number of concurrent vertices. (Hint: See the previous exercise.)

16. Verify that the modified Ford and Fulkerson algorithm for networks with upper and lower bounds actually gives the desired results.

17. Complete the proof of the edge version of Menger's theorem (Theorem 4.6.2).

18. If $paths(u, v)$ is the maximum number of pairwise disjoint $u - v$ paths in a graph G, show that if G is not complete, then

$$\min_{u,v \in V(G)} paths(u, v) = \min_{uv \notin E(G)} paths(u, v);$$

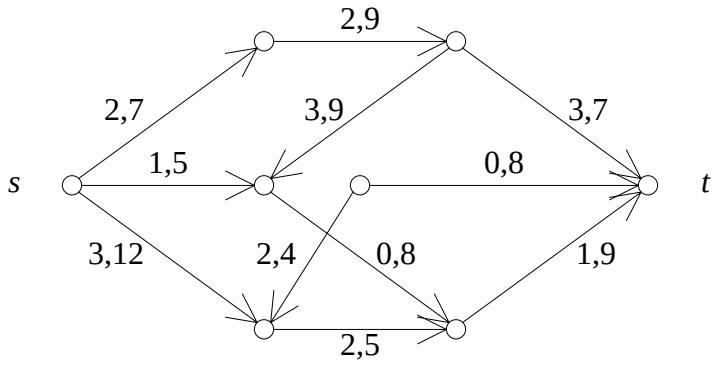
that is, the minimum value of $paths(u, v)$ occurs for some nonadjacent pair of vertices u, v.

19. Show that if the connectivity of a graph G is k, then
$$k = \min_{u,v \in V(G)} paths(u, v).$$

20. Let N be a $0 - 1$ network with no multiple arcs and let M be the maximum total flow from s to t in N. Show that the length of the first layered network (with zero flow everywhere) is at most $\frac{2|V|}{M^{1/2}} + 1$.

21. Show that for the network N of the previous problem, Dinic's algorithm has time complexity $O(|V^{2/3}|\ |E|)$.

22. Determine if the network below has a legal flow. If it does not, modify as few capacities as possible to obtain a network that does have a legal flow.



References

1. Chvátal, V., *Linear Programming.* W. H. Freeman Company, New York (1980).
2. Dantzig, G. B. and Fulkerson, D. R., On the Max-Flow Min-Cut Theorem of Networks, Linear Inequalities and Related Systems, *Annals of Math.*, Study 38, Princeton University Press, (1956), 215 – 221.
3. Dilworth, R. P., A Decomposition Theorem for Partially Ordered Sets. *Ann. Math.*, Vol. 51 (1950), 161 – 166.
4. Dinic, E. A., Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation, *Soviet Math. Dokl.*, 11(1970), 1277 – 1280.
5. Edmonds, J., and Karp, R. M., Theoretical Improvements in Algorithm Efficiency for Network Flow Problems, *J. ACM,* 19(1972), 248 – 264.
6. Ford, L. R., Jr., and Fulkerson, D. R., *Flows in Networks*. Princeton University Press, (1962).
7. Karzanov, A. V., Determining the Maximal Flow in a Network by the Method of Preflows, *Soviet Math. Dokl.*, 15(1974), 434 – 437.
8. Malhorta, V. M., Kumar, M. P., and Maheshwari, S. N., An $O(|V|^3)$ Algorithm for Finding Maximum Flows in Networks, Computer Science Program, Indian Institute of Technology, Kanpur 208016, India (1978).

9. Sleator, D. D., An *O*(nm log *n*) Algorithm for Maximum Network Flow, Stanford University, Stanford, CA, Computer Science Department report STAN-CS-80-831 (1980).

Chapter 5
Cycles and Circuits

Section 5.1 Eulerian Graphs

Probably the oldest and best known of all problems in graph theory centers on the bridges over the river Pregel in the city of Königsberg (presently called Kaliningrad in Russia). The legend says that the inhabitants of Königsberg amused themselves by trying to determine a route across each of the bridges between the two islands (*A* and *B* in Figure 5.1.1), both river banks (*C* and *D* of Figure 5.1.1) and back to their starting point using each bridge exactly one time. After many attempts, they all came to believe that such a route was not possible. In 1736, Leonhard Euler [15] published what is believed to be the first paper on graph theory, in which he investigated the Königsberg bridge problem in mathematical terms.

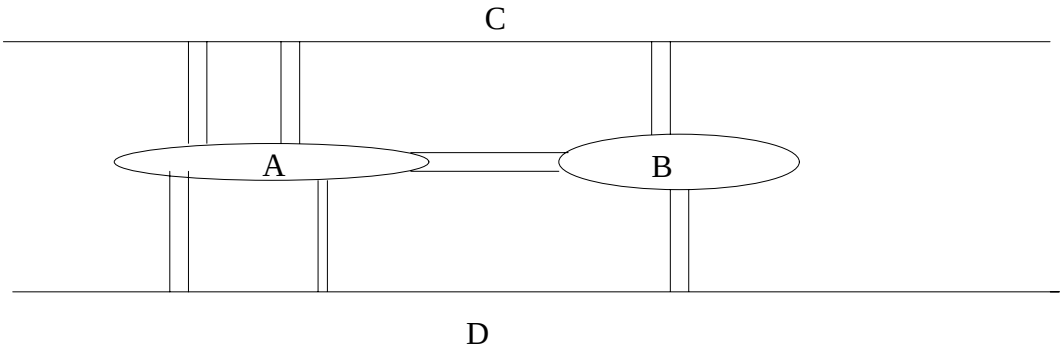


Figure 5.1.1. The bridges on the river Pregel.

The solution to the bridge problem hinges on the degrees of the vertices in the graph model for the bridges and land masses (see Figure 5.1.2). The problem seeks a circuit that contains each edge. In honor of Euler, we say a graph (or multigraph) is *eulerian* if it has a circuit containing all the edges of the graph. The circuit itself is called an *eulerian circuit.* A trail containing every edge of the graph is called an *eulerian trail.*

Several characterizations have been developed for eulerian graphs. Euler has frequently been credited with the complete equivalence of statements 1 and 2 in Theorem 5.1.1; however, he actually established only half of it ($1 \rightarrow 2$). Hierholzer [33] established the converse. The result below is a blend of the work of Euler [15], Hierholzer [33] and Veblen [46].

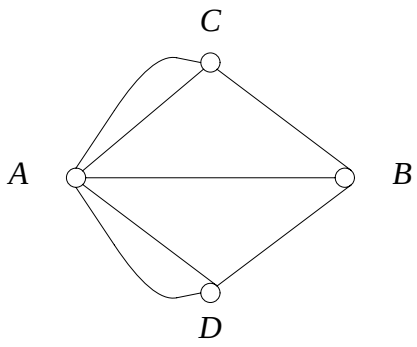


Figure 5.1.2. The multigraph of the bridges.

Theorem 5.1.1 The following statements are equivalent for a connected graph G:

1. The graph G contains an eulerian circuit.
2. Each vertex of G has even degree.
3. The edge set of G can be partitioned into cycles.

Proof. To see that (1) implies (2), let v be a vertex of G. If C is an eulerian circuit in G, then each edge of G entering v must be matched by another edge leaving v. Thus, the degree of any vertex must be even.

To establish that (2) implies (3), we use induction on the number of cycles in the graph. Clearly, the result holds if there are no cycles or there is one cycle in G. Now, assume the result holds if there are fewer than k cycles in G and suppose that G is a connected graph with all vertices having even degree and containing exactly k cycles. Delete the edges of one cycle C^* of G. Each component of the remaining graph clearly has the property that all vertices have even degree. Further, each component has fewer than k cycles, and, thus, by the induction hypothesis, their edge sets can be partitioned into cycles. The edge set of G can then be partitioned in a similar manner and along with C^* we obtain the desired partition of $E(G)$.

Finally, to see that (3) implies (1), assume that the edge set of G can be partitioned into cycles and suppose there are m such cycles. We use induction to establish that for any k $(1 \leq k \leq m)$, there is a circuit in G which contains each edge of k of the cycles and no other edge of G. The statement is clear if $k = 1$, since a single cycle suffices. Thus, we assume there is a circuit C which contains k of the cycles $(k < m)$ and no other edges of G. Since G is connected, at least one of the other cycles must contain a vertex of C.

Let C_1 be the circuit obtained by traversing that cycle, beginning at some common vertex v (and, hence, returning there), and then following C. Then clearly, C_1 contains the edges of $k + 1$ cycles and no other edges; hence, the result follows by induction. □

Since every graph contains an even number of vertices of odd degree, the following corollary is easily obtained.

Corollary 5.1.1 A connected graph G contains an eulerian trail if, and only if, at most two vertices of G have odd degree.

We can, however, extend Corollary 5.1.1 a little further. The proof is left to the exercises.

Corollary 5.1.2 Let $G = (V, E)$ be a connected graph with $2k$ $(k > 0)$ vertices of odd degree. Then E can be partitioned into exactly k open trails.

These ideas can certainly be extended to directed graphs. The literature on this subject contains a wide variety of terms for such digraphs. We shall simply call them eulerian digraphs. The following result is analogous to Theorem 5.1.1. Its proof is also left to the exercises.

Theorem 5.1.2 The following statements are equivalent for a connected digraph $D = (V, E)$:

1. The digraph D has a directed eulerian circuit.
2. For each vertex $v \in V$, $id\ v = od\ v$.
3. The arc set E can be partitioned into directed cycles.

Next, we wish to consider algorithms designed to produce eulerian circuits. Perhaps the oldest such algorithm is from Fleury [22]. Fleury's approach is to construct a trail C that will grow to be the desired eulerian circuit, constantly expanding the number of edges being used in C, while avoiding bridges in the graph formed by the edges not yet included in C. A bridge is selected only when no other choice remains. The postponing of the use of bridges is really the critical feature of this algorithm, and its purpose is to avoid becoming trapped in some component of $G - C$.

Algorithm 5.1.1 Fleury's Algorithm.

Input: A connected (p, q) graph $G = (V, E)$.
Output: An eulerian circuit C of G.
Method: Expand a trail C_i while avoiding bridges in $G - C_i$, until no other choice remains.

1. Choose any $v_0 \in V$ and let $C_0 = v_0$ and $i \leftarrow 0$.
2. Suppose that the trail $C_i = v_0, e_1, v_1, \ldots, e_i, v_i$ has already been chosen:
   a. At v_i, choose any edge e_{i+1} that is not on C_i and that is not a bridge of the graph $G_i = G - E(C_i)$, unless there is no other choice.
   b. Define $C_{i+1} = C_i, e_{i+1}, v_{i+1}$.
   c. Let $i \leftarrow i + 1$.
3. If $i = |E|$
   then halt since $C = C_i$ is the desired circuit;
   else go to 2.

Theorem 5.1.3 If G is eulerian, then any circuit constructed by Fleury's algorithm is eulerian.

Proof. Let G be an eulerian graph. Let $C_p = v_0, e_1, \ldots, e_p, v_p$ be the trail constructed by Fleury's algorithm. Then clearly, the final vertex v_p must have degree 0 in the graph G_p, and hence $v_p = v_0$, and C_p is a circuit.

Now, to see that C_p is the desired circuit, suppose instead that C_p is not an eulerian circuit of G. Thus, there must be edges of G not on C_p. Let S be the set of vertices of positive degree in G_p. Hence, $S \cap V(C_p)$ is nonempty since G is connected and $v_p \in \overline{S} = V - S$. Let i be the largest integer such that $v_i \in S \cap C_p$ but $v_{i+1} \in \overline{S}$. Since C_p ends in $\overline{S}$, it follows that $i < p$. From the definition of $\overline{S}$, each edge of G_i that joins S and $\overline{S}$ is on C_p; thus, the edge e_{i+1} is the only edge from S to $\overline{S}$ in the graph G_i. But then e_{i+1} is a bridge in G_i.

Suppose that e is any other edge of G_i that is incident to v_i. Then from step 2 of the algorithm, it follows that e must also be a bridge of G_i (and, hence, of the graph H_i, induced by S in G_i). Since $H_i \subseteq H_p$ (the graph induced by S in G_p), it follows that e is also a bridge in H_p. Further, since e_{i+1} is a bridge of G_i and v_i is the last vertex on C_p that is also in S, we see that $H_i = H_p$ and that $deg_{H_p} v = deg_{G_p} v$ for every vertex v of H_p. Thus, every vertex in H_p has even degree. But we know from exercise 15 in Chapter 2 that this implies that H_p contains no bridges, a contradiction. □

Can you determine the complexity of Fleury's algorithm?

Hierholzer [33] developed an algorithm that produces circuits in a graph G which are pairwise edge disjoint. When these circuits are put together properly, they form an eulerian circuit of G. This patching together of circuits hinges of course, on the circuits having a common vertex, and this fact follows from the connectivity of the graph. Once one circuit is formed, if all edges have not been used, then there must be one edge that is incident to a vertex of the circuit, and we use this edge to begin the next circuit. These circuits then share a common vertex.

Algorithm 5.1.2 Hierholzer's Algorithm.
Input: A connected graph $G = (V, E)$, each of whose vertices has even degree.
Output: An eulerian circuit C of G.
Method: Patching together of circuits.

1. Choose $v \in V$. Produce a circuit C_0 beginning with v by traversing at each step, any edge not yet included in the circuit. Set $i = 0$.
2. If $E(C_i) = E(G)$;
   then halt since $C = C_i$ is an eulerian circuit;
   else choose a vertex v_i on C_i that is incident to an edge not on C_i. Now build a circuit C_i^* beginning with v_i in the graph $G - E(C_i)$. (Hence, C_i^* also contains v_i.)
3. Build a circuit C_{i+1} containing the edges of C_i and C_i^* by starting at v_{i-1}, traversing C_i until reaching v_i, then traversing C_i^* completely (hence, finishing at v_i) and then completing the traversal of C_i. Now set $i \leftarrow i + 1$ and go to 2.

Tucker [45] developed an algorithm that is essentially a combination of the methods of Fleury and Hierholzer. In order to present his algorithm, we need another idea. If $e_1 = vw$ and $e_2 = vx$ are two edges of G, then the *splitting away of* e_1 *and* e_2 results in a new graph H obtained by deleting e_1 and e_2 from G and adding a new vertex z adjacent to w and x (see Figure 5.1.3). Note that G can be recovered from H by *identifying* the vertices z and v, that is, by replacing z and v with one new vertex adjacent to all the neighbors of z and v. It is clear that H is eulerian if, and only if, G is eulerian and $\{ e_1, e_2 \}$ does not form a cut set. It is also not hard to prove that H is connected if, and only if, G is connected and $\{ e_1, e_2 \}$ does not form a cut set (see exercise 6 in Chapter 5). We now present Tucker's algorithm. Repetition of the splitting away process is intended to produce the cycle partition of E promised in Theorem 5.1.1.

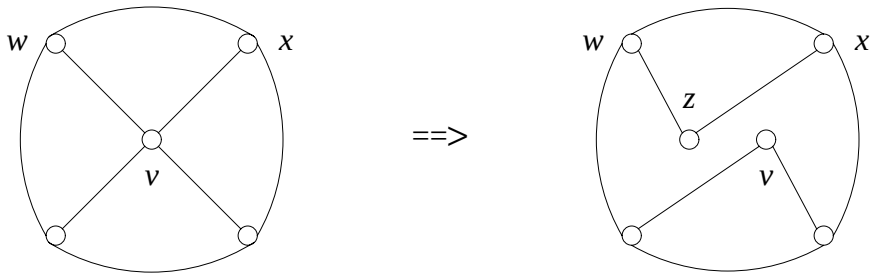


Figure 5.1.3. Splitting away of e_1 and e_2.

Algorithm 5.1.3 Tucker's Algorithm.
Input: An eulerian graph $G = (V, E)$.
Output: An eulerian circuit C of G.
Method: Split away the edges to form the desired cycle partition and then form the circuit by reassembling the cycles in a controlled manner.

1. Split away pairs of edges of G until there are only vertices of degree 2 remaining. Call the graph obtained in this manner G_1. Set $i \leftarrow 1$ and let c_i be the number of components of G_i.

2. If $c_i = 1$,
   then halt with $C = G_i$;
   else find two components T and T^* of G_i with the vertex v_i in common. Form a circuit C_{i+1}, starting at v_i and traversing T and T^*, ending again at v_i (as discussed earlier).

3. Define $G_{i+1} = G_i - \{T, T^*\} \cup C_{i+1}$. (Here, we consider the circuit C_{i+1} as being a component of G_{i+1}.) Set $T = C_{i+1}$, $i \leftarrow i + 1$, let c_i be the number of components of G_i and go to 2.

We conclude this section with the famous problem, introduced by Kwan [34] in 1960, called the *Chinese postman problem.* The problem is that the postman must traverse his mail route each day, traveling along each road and eventually returning to the post office, his starting point. Certainly, the postman desires a route that minimizes the total distance he travels.

Clearly, we can use a graph to model the mail route he must eventually cover. A solution is provided by a closed walk of minimum length that uses each edge at least once in the graph modeling the entire mail route.

It is a simple observation that if the (p, q) graph G models the postman's route, then certainly the distance the postman must travel is at least q. It is also easy to see that if the graph under consideration is a tree, then each edge must be used twice. Certainly, there is no need to use any edge three times. Hence, for the distance d in the Chinese postman problem, $q \leq d \leq 2q$. It is further clear that if the graph under consideration is an eulerian graph, then the Chinese postman problem can be solved by any of our previous eulerian algorithms. Goodman and Hedetniemi [26] noticed that the problem of finding the postman's walk is equivalent to the problem of determining the minimum number of edges that must be duplicated in order to produce an eulerian multigraph. One way to solve this problem is to consider all possible pairs of vertices of odd degree and pair them according to the minimum distances between them. Insert these edges to form the desired eulerian multigraph. We can now apply any of our algorithms for finding eulerian circuits to complete the task. However, in general this is not an efficient algorithm.

Edmonds and Johnson [13] provided a good algorithm for solving this problem, under the generalized condition that the edge distances were $w(e) \geq 0$. However, their solution is beyond the scope of this text.

Section 5.2 Adjacency Conditions for Hamiltonian Graphs

In the previous section we tried to determine a *tour* of a graph that would use each edge once and only once. It seems natural to vary this question and try to visit each vertex once and only once. In fact, a cycle (path) that contains every vertex of a graph is called a *hamiltonian cycle* (*path*).

This terminology is used in honor of Sir William Rowan Hamilton, who, in 1865, described an idea for a game in a letter to a friend. This game, called the icosian game, consisted of a wooden dodecahedron (see Figure 5.2.1) with pegs inserted at each of the twenty vertices. These pegs supposedly represented the twenty most important cities of the time. The object of the game was to mark a route (following the edges of the dodecahedron) passing through each of the cities only once and finally returning to the initial city. Hamilton sold the idea for the game. (I think his profit was the only one made on this game!)

From this unusual beginning has sprung a stream of interesting work concerning spanning cycles and paths in graphs. We cannot even begin to completely survey the extensive literature on this subject. Instead, we wish to hit some of the highlights and show some of the diverse approaches taken in what has become a very interesting and important subject.

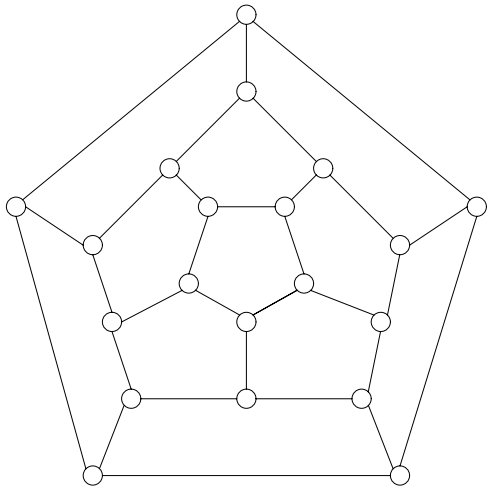


Figure 5.2.1. The graph of the dodecahedron.

Since the idea of a hamiltonian cycle "sounds" analogous to that of an eulerian circuit, one might expect that we will be able to establish some sort of corresponding theory. This, however, is definitely not the case. In fact, the hamiltonian problem is NP-complete (see [24]), and no practical characterization for hamiltonian graphs has been found. Hence, we shall not study any general algorithms for directly finding such cycles. However, there has still been a considerable amount of information discovered about hamiltonian graphs.

Perhaps the theorem that stirred the most subsequent work is that of Ore [41]. It stems from the idea that if a sufficient number of edges are present in the graph, then a hamiltonian cycle will exist. To ensure a sufficient number of edges, we try to keep the degree sum of nonadjacent pairs of vertices at a fairly high level. We can see the effect that controlling degree sums provides when we consider a vertex x of "low" degree. Since x has many nonadjacencies in the graph, the degrees of all these vertices are then forced to be "high," to ensure that the degree sum remains sufficiently large. Thus, the graph has many vertices of high degree, and so, one hopes it contains enough structure to ensure that it is hamiltonian.

Theorem 5.2.1 If G is a graph of order $p \geq 3$ such that for all pairs of distinct nonadjacent vertices x and y, $deg\ x + deg\ y \geq p$, then G is hamiltonian.

Proof. Suppose the result is not true. Then there exists a maximal nonhamiltonian graph G of order $p \geq 3$ that satisfies the conditions of the theorem. That is, G is nonhamiltonian, but for any pair of nonadjacent vertices x and y in G, the graph $G + xy$

is hamiltonian. Since $p \geq 3$, the graph G is not complete.

Now, consider $H = G + xy$. The graph H is hamiltonian and, in fact, every hamiltonian cycle in H must use the edge from x to y. Thus, there is an $x - y$ hamiltonian path P: $x = v_1, v_2, \ldots, v_p = y$ in G. Now, observe that if $v_i \in N(x)$, then $v_{i-1} \notin N(y)$, or else (see Figure 5.2.2)

$$v_1, v_i, v_{i+1}, \ldots, v_p, v_{i-1}, v_{i-2}, \ldots, v_1$$

would be a hamiltonian cycle in G. Hence, for each vertex adjacent to x, there is a vertex of $V - \{ y \}$ not adjacent to y. But then, we see that $deg\ y \leq (p - 1) - deg\ x$. That is, $deg\ x + deg\ y \leq p - 1$, a contradiction. □

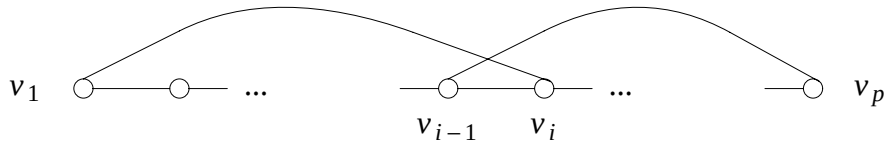


Figure 5.2.2. The path P and possible adjacencies.

We now state an immediate corollary of Ore's theorem that actually preceded it. This result is originally from Dirac [11]. A more elegant proof that inspired Ore's work was provided independently by Newman [40].

Corollary 5.2.1 If G is a graph of order $p \geq 3$ such that $\delta(G) \geq \frac{p}{2}$, then G is hamiltonian.

Following Ore's theorem, a stream of further generalizations were introduced. This line of investigation culminated in the following work of Bondy and Chvátal [4]. The next result stems from their observation that Ore's proof does not need or use the full power of the statement that each nonadjacent pair satisfies the degree sum condition.

Theorem 5.2.2 Let x and y be distinct nonadjacent vertices of a graph G of order p such that $deg\ x + deg\ y \geq p$. Then $G + xy$ is hamiltonian if, and only if, G is hamiltonian.

Proof. If G is hamiltonian, then clearly $G + xy$ is hamiltonian. Conversely, if $G + xy$ is hamiltonian but G is not, then we proceed exactly as in Ore's theorem to produce a contradiction to the degree sum condition. □

The last result inspired the following definition. The *closure* of a graph G, denoted $CL(G)$, is that graph obtained from G by recursively joining pairs of nonadjacent vertices whose degree sum is at least p until no such pair remains. The first thing we must verify is that this definition is well defined; that is, since no order of operations is specified, we must verify that we always obtain the same graph, no matter what order we use to insert these edges.

Theorem 5.2.3 If G_1 and G_2 are two graphs obtained by recursively joining pairs of nonadjacent vertices whose degree sum is at least p until no such pair remains, then $G_1 = G_2$; that is, $CL(G)$ is well defined.

Proof. Denote by $e_1, e_2, \ldots, e_i$ and $f_1, f_2, \ldots, f_j$ the sequences of edges inserted in G to form G_1 and G_2, respectively. We wish to show that each e_m is in G_2 and that each f_n is in G_1.

Let $e_k = uv$ be the first edge in the sequence $e_1, \ldots, e_i$ that is not in G_2. Consider the graph $H = G + \{ e_1, \ldots, e_{k-1} \}$. Then, from the definition of G_1 it follows that

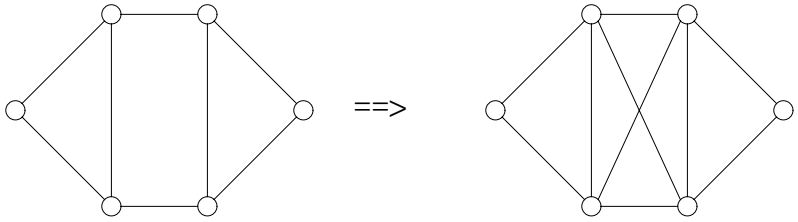
$$deg_H\ u + deg_H\ v \geq p.$$

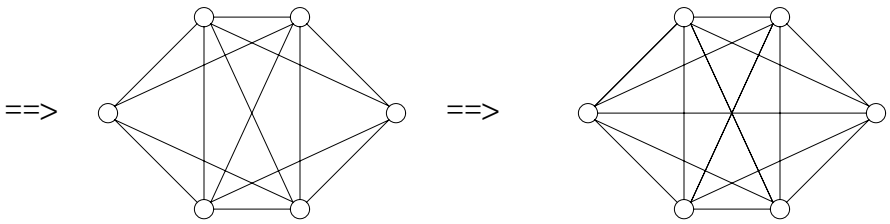
Also, by our choice of e_k, H is a subgraph of G_2. Thus,

$$deg_{G_2}\ u + deg_{G_2}\ v \geq p.$$

But this is a contradiction since u and v are nonadjacent in G_2. Similarly, each f_n is in G_1 and, hence, $G_1 = G_2$ and the closure is well defined. □

Example 5.2.1. We illustrate the construction of the closure of the graph pictured below.





Using Theorem 5.2.2, we see that the next result is immediate.

Theorem 5.2.4 A graph G is hamiltonian if, and only if, $CL(G)$ is hamiltonian.

If $CL(G) = K_p$, then it is immediate that $CL(G)$ is hamiltonian and, hence, that G is hamiltonian. This observation offers another proof of Ore's theorem and of Dirac's theorem.

In [4], a $O(V^3)$ algorithm for finding a hamiltonian cycle in a graph satisfying the degree sum condition $deg\ x + deg\ y \geq |V(G)|$ for all nonadjacent pairs of vertices x, y (call such graphs *Ore-type*) using the closure was provided. Recently, Albertson has provided a $O(V^2)$ algorithm. His approach is based on two observations taken essentially from Ore's proof technique.

Observation 1. In any Ore-type graph G, the vertices of any maximal path can be permuted to form a cycle C.

Observation 2. In any Ore-type graph G, if $x \notin V(C)$, then there is a path that contains x and all the vertices of C.

The proofs of these observations are left to the exercises. We now present Albertson's algorithm [1].

Algorithm 5.2.1 Albertson's Algorithm.
Input: An Ore-type graph G.
Output: A hamiltonian cycle in G.
Method: Use of observations 1 and 2.

1. Create a maximal path P: $u\ ,\ \ldots\ ,\ x_k\ ,\ \ldots\ ,\ v$.
2. Repeat while $|V(C)| \neq |V(G)|$
   If u is adjacent to v,

then set C: $u, \ldots, v, u$;
else find a k such that u is adjacent to x_{k+1} and v is adjacent to x_k. Set C: $u, x_{k+1}, x_{k+2}, \ldots, v, x_k, x_{k-1}, \ldots, x, u$.

3. Find $x \in V(G - C)$, and create P^*, a path containing x and all of C. Set P equal to a maximal path containing P^*.

4. endwhile

Next, we consider an alternate approach to ensuring a sufficient number of edges that are properly distributed to guarantee that a graph is hamiltonian. Recently, the idea of bounding the cardinality of the neighborhoods of nonadjacent vertices has been used to determine when many graph properties are present. In [18], this idea was introduced and used to determine a sufficient condition for a graph to be hamiltonian. If $G \neq K_p$, no connectivity condition can be forced by bounding the cardinality of neighborhood unions of nonadjacent vertices. (That is, once we have avoided the pathological case in which the neighborhood condition is satisfied vacuously, then no connectivity condition can be forced by the neighborhood condition.) For example, consider the graph $K_r \cup K_r$. Thus, we must assume some minimal connectivity condition.

Theorem 5.2.5 If G is a 2-connected graph such that for every pair of nonadjacent vertices x and y,

$$|N(x) \cup N(y)| \geq \frac{2p - 1}{3},$$

then G is hamiltonian.

Can you find an example of a graph G which Theorem 5.2.5 shows is hamiltonian, but for which $CL(G) = G$?

One can consider Dirac's theorem as providing a bound on the cardinality of the neighborhood of a single vertex, and the last result provides a bound on the cardinality of the neighborhood of a pair of nonadjacent vertices. In [18] it was conjectured that even higher connectivity conditions would allow expansion of the set of nonadjacent vertices to those with more than two vertices. Fraisse [23] verified that this conjecture was indeed true. We consider $N(S)$ to be the set of neighbors of the vertices in the set S. For convenience, we define $[x, y]$ to mean the segment of a path or cycle, following some orientation, from the vertex x to the vertex y (including both x and y). If we do not wish to include an end vertex we will use the notation (x, y), or a mixture of the two notations where appropriate.

Theorem 5.2.6 Let G be a k-connected graph of order $n \geq 3$. Suppose there exists some $t \leq k$ such that for every set S of t mutually nonadjacent vertices, $|N(S)| > \dfrac{t(n-1)}{t+1}$; then G is hamiltonian.

Proof. Suppose G is not hamiltonian and let C be a cycle of maximum length, say m, in G. Let x_0 be a vertex of G off C and fix the following ordering of C: $v_1, v_2, \ldots, v_m$. Using Menger's theorem (see exercise 14 in Chapter 2), we find that there exist t paths from x_0 to C that are disjoint except at x_0. Let $\{ w_1, w_2, \ldots, w_t \}$ denote the end vertices of these paths on C and, further, assume they occur in this order according to our ordering of C. Also, denote by x_i the successor of w_i with respect to this ordering along C and let $X = \{ x_i \mid 0 \leq i \leq t \}$. It is clear that X is a set of $t+1$ mutually nonadjacent vertices or else a cycle longer than C would exist in G. Our strategy is to produce a subset of X that does not satisfy the hypothesis of the theorem.

We now partition $V(G)$ into the following sets:

$$A = V(G) - N(X)$$

(A is the set of vertices that are not adjacent to any vertex in X);

$$B = \{v \in V(G) \mid \text{there is a unique } i \text{ such that } vx_i \in E(G)\},$$

(B is the set of vertices u such that there is a subset Y of t vertices of X with $u \notin N(Y)$); and

$$D = V(G) - (A \cup B).$$

Further, let $A_C = A \cap V(C)$ and $A_R = A - V(C)$.

Now, $X \subseteq A$ and, therefore, $x_0 \in A_R$, or else a cycle longer than C results. Further, two vertices of X have no common neighbors in $V(G) - V(C)$, or again a longer cycle results.

Claim: There are at most t vertices of D between any two consecutive vertices of A on C.

To verify this claim, suppose that a_1 and a_2 are two consecutive vertices of A on C. Since $X \subseteq A$, we may assume that a_1 and a_2 belong to some segment of C of the form $[x_i, x_j]$ (where $0 < i < j \leq t$) or $[x_t, x_1]$. Let $v \in N(x_j)$ and (letting v^+ be the successor of v on C) let $v^+ \in N(x_k)$ ($x_k \in (x_j, x_i)$) where v and v^+ are in the segment of vertices $[a_1, a_2]$. Then, we see that x_j is not x_0 or a longer cycle would exist. In fact, if $x_0 v \in E(G)$, then $v^+ x_k \notin E(G)$ for any k, or a cycle longer than C would exist (see Figure 5.2.3). Furthermore, the vertices w_i, w_j, and w_k cannot occur in that order

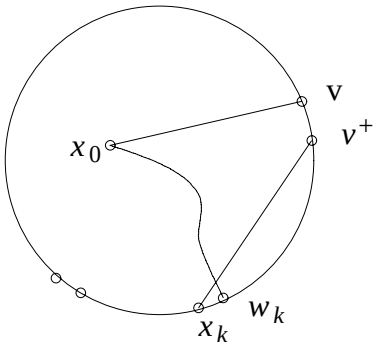


Figure 5.2.3. Extending the cycle through x_0.

along C (according to its ordering), because again a cycle longer than C would exist (see Figure 5.2.4). For example

$$x_0, \ldots, w_k, \ldots, x_j, v, \ldots, x_k, v^+, \ldots, w_j, x_0$$

is such an extension. Therefore,

$$N(x_i),\ N(x_{i-1}),\ N(x_{i-2}) \ , \ldots, \ N(x_1),\ N(x_t) \ , \ldots, \ N(x_{i+1}),\ N(x_0)$$

form consecutive segments of vertices in the segment $[\ a_1, a_2\ ]$ (which is possibly empty). Further, these segments have at most their end vertices in common. There are $t + 1$ segments; hence, there are at most t elements of D between a_1 and a_2. Thus, our claim is verified.

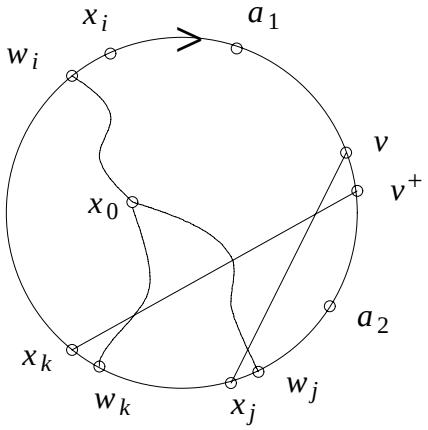


Figure 5.2.4. The ordering of $w_i,\ w_j,\ w_k$.

To complete the proof of the result, let x_j, $0 \le j \le t$ be chosen with the maximum number s of neighbors in B. Then $s \ge |B|/(t+1)$. Also, the number of vertices of B in $V(G) - V(C)$ equals $n - m - |A_R|$ (since two vertices of X have no common neighbors in $V(G) - V(C)$). From our claim, the number of vertices of B on C is at least $m - (t+1)\,|A_C|$. Thus,

$$\begin{aligned} |B| &= |B \cap C| + |B - C| \\ &\ge n - m - |A_R| + m - (t+1)\,|A_C| \\ &= n - |A_R| - (t+1)\,|A_C| \end{aligned}$$

and

$$s \ge \frac{|B|}{t+1} \ge \frac{n}{t+1} - \frac{|A_R|}{t+1} - |A_C|.$$

Since $X - \{x_j\}$ is a set of t mutually nonadjacent vertices and since $|A_R| \ge 1$,

$$\begin{aligned} |N(X - x_j)| &= n - |A| - s = n - |A_R| - |A_C| - s \\ &\le \frac{t(n - |A_R|)}{t+1} \le \frac{t(n-1)}{t+1}. \end{aligned}$$

Hence, a contradiction is reached and the result is proved. □

Example 5.2.2. The best known examples to show the sharpness of the bound in the previous result are obtained by considering $t+1$ copies of the complete graph K_r which are identified on a set of t vertices. The Petersen graph (see Figure 6.3.2) is another example. However, these examples all leave some doubt as to the best possible lower bound.

The graph $tK_r + K_t$ satisfies the hypothesis of Fraisse's Theorem, but its n-closure is itself. Thus, Theorem 5.2.6 is independent of the closure (Theorem 5.2.2).

It is interesting to note that the type of vertex set used can play a role here. In the last theorem independent sets of vertices were considered. However, something a bit different happens if we consider arbitrary sets of vertices.

Theorem 5.2.7 [17] If G is a 2-connected graph of sufficiently large order n and if for each pair of arbitrary vertices $S = \{a, b\}$ we have that

$$|N(S)| \geq \frac{n}{2},$$

then G is hamiltonian.

It is known that $n > 10$ suffices in Theorem 5.2.7. The Petersen graph is a small order (10) counterexample.

It is natural now to ask what happens if the set of vertices induces a clique? What properties in general are important to the selection of the set? Some of these questions have been considered, but the last and most important of them remains open. The interested reader should see [28] and [35] for more information.

Section 5.3 Related Hamiltonian-like Properties

In this section we wish to investigate several properties closely related to that of being hamiltonian. Some of these are stronger properties, in the sense that the graphs having these properties are also hamiltonian, while others are weaker. It is not surprising that truly applicable characterizations of these properties are not known. In some cases, very little is actually known at all about the classes of graphs we will describe.

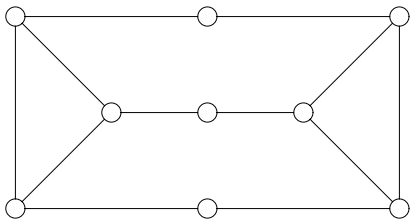


Figure 5.3.1. A homogeneously traceable nonhamiltonian graph.

To begin with, we say a graph is *traceable* if it contains a hamiltonian path. Clearly, every hamiltonian graph is also traceable, and the graphs P_n show that the converse of this statement does not hold. There are two other rather elusive classes of graphs that essentially lie between the hamiltonian and traceable classes. A graph G is *homogeneously traceable* if there is a hamiltonian path beginning at every vertex of G, while G is *hypohamiltonian* if G is not hamiltonian but $G - v$ is hamiltonian for every vertex v of G. It is easy to see that every hypohamiltonian graph is also homogeneously

traceable. The graph of Figure 5.3.1 is homogeneously traceable, but not hypohamiltonian.

Skupien introduced homogeneously traceable graphs in [44], and the existence of homogeneously traceable nonhamiltonian graphs for all orders $p \geq 9$ was shown in [6]. In [32], Herz, Gaudin and Rossi showed that hypohamiltonian graphs exist, while in [31], Herz, Duby and Vigué showed that there is no hypohamiltonian graph of order 11 or 12. Lindgren [37] and Sousselier (see [31]) independently showed that infinitely many hypohamiltonian graphs exist. Both of these classes have remained elusive in the sense that very few results are known about them, especially in view of the vast number of results known about hamiltonian graphs. Some simple facts will be explored in the exercises.

More success has been had with properties stronger than that of being hamiltonian. We say a graph G is *hamiltonian connected* if every two vertices of G are joined by a hamiltonian path. Clearly, every hamiltonian connected graph of order at least 3 is hamiltonian; the graphs C_n $(n \geq 4)$ show that the converse is not true. We can use a generalization of the idea of the closure to obtain a sufficient condition for a graph to be hamiltonian connected. For a (p, q) graph G, let the $(p + 1)$-closure denoted $CL_{p+1}(G)$, be that graph obtained from G by recursively joining pairs of nonadjacent vertices whose degree sum is at least $p + 1$. We obtain the following analog of Theorem 5.2.4, also from Bondy and Chvátal [4].

Theorem 5.3.1 Let G be a graph of order p. If $CL_{p+1}(G)$ is complete, then G is hamiltonian connected.

Proof. If $p = 1$, the result is clear. Thus, assume that $p \geq 2$ and that G is a graph of order p such that $CL_{p+1}(G) = K_p$. Let x and y be two vertices of G. We show that x and y are joined by a hamiltonian path in G, and since they are an arbitrary pair of vertices, the result will follow.

Let H be the graph obtained from G by inserting a new vertex w and the edges wx and wy. Then H has order $p^* = p + 1$.

Since $deg_H u + deg_H v \geq p^*$ for all nonadjacent vertices u and v in G, it follows that $< V(G) >$ in $CL(H)$ must be isomorphic to K_p. Thus, if $u \in V(G)$ and $uw \notin E(H)$, then $deg_{CL(H)} u + deg_{CL(H)} w \geq p^*$, and, hence, $CL(H)$ is isomorphic to K_{p^*}. Then by Ore's theorem, H is hamiltonian. But any hamiltonian cycle in H must contain the edges wx and wy, and, thus, x and y are end vertices of a hamiltonian path in G. □

The next two corollaries are analogs of the theorems of Ore and Dirac and are immediate from our previous result, although they can also be proved directly.

Corollary 5.3.1 If G is a graph of order p such that for every pair of distinct nonadjacent vertices x and y in G, $deg\ x + deg\ y \geq p + 1$, then G is hamiltonian connected.

Corollary 5.3.2 If G is a graph of order p such that $deg\ x \geq \frac{p + 1}{2}$ for every vertex x, then G is hamiltonian connected.

Yet another hamiltonian-like property is the following: A connected graph $G = (V, E)$ is said to be *panconnected* if for each pair of distinct vertices x and y, there exists an $x - y$ path of length l, for each l satisfying $d(x, y) \leq l \leq |V| - 1$. If a graph G is panconnected, it is clearly hamiltonian connected and, thus, hamiltonian. It is also easy to see that there are hamiltonian graphs that are not panconnected (for example, cycles). Can you find a graph that is hamiltonian connected but not panconnected? In a result similar to Dirac's theorem, Williamson [47] provided a sufficient condition for a graph to be panconnected in terms of minimum degree.

Theorem 5.3.2 If G is a graph of order $p \geq 4$ such that for every vertex $x \in V(G)$, $deg\ x \geq \frac{p + 2}{2}$, then G is panconnected.

Our final properties are somewhat related. We say a graph G of order p is *pancyclic* if it contains a cycle of every length l, $(3 \leq l \leq p)$. We say G is *vertex pancyclic* if each vertex of G lies on a cycle of each length l, $(3 \leq l \leq p)$. Clearly, every pancyclic and every vertex pancyclic graph is hamiltonian. It is also clear that every vertex pancyclic graph is pancyclic. Can you find a graph that is pancyclic, but not vertex pancyclic? A sufficient condition for a graph to be pancyclic was provided by Bondy [3].

Theorem 5.3.3 If G is a hamiltonian (p, q) graph with $q \geq \frac{p^2}{4}$, then either G is pancyclic or p is even and G is isomorphic to $K_{p/2,\ p/2}$.

Many hamiltonian results, similar to those we have studied, exist for digraphs. We explore some of these in the exercises.

Section 5.4 Forbidden Subgraphs

In this section we consider another approach that has seen considerable study. Goodman and Hedetniemi [27] noticed that forbidding particular induced subgraphs offered a new type of hamiltonian result. If $F = \{H_1, \ldots, H_k\}$ is a family of graphs, we say that G *is F-free* if G does not contain any of the graphs in the set F as an induced subgraph. If F is a single graph H we simply say that G is H-free. We define $Z_1 = K_{1,3} + e$.

Theorem 5.4.1 ([27]) If G is a 2-connected $\{K_{1,3}, Z_1\}$-free graph, then G is hamiltonian.

Proof. Suppose G is not hamiltonian. Since G is 2-connected, it contains cycles. Let $C: x_1, x_2, \ldots, x_n, x_1$ be a cycle in G of maximum length. Then, there must exist a vertex x not on C that is adjacent to a vertex, say x_i, of C. Then, $< x, x_i, x_{i-1}, x_{i+1} >$ induces either a $K_{1,3}$ or a Z_1, unless at least two of the edges xx_{i-1}, xx_{i+1}, or $x_{i-1}x_{i+1}$ are present in G. But then no matter how these edges are selected, one of xx_{i-1} or xx_{i+1} must be present, and a cycle longer than C is produced, a contradiction to our choice of C. Thus, G is hamiltonian. □

Theorem 5.4.1 inspired several others. One of the earliest and strongest of these is now stated. The graph N (known as the net) is seen in Figure 5.4.1.

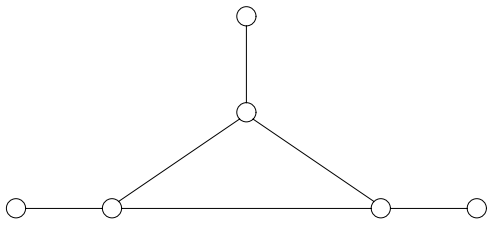


Figure 5.4.1. The graph N.

Theorem 5.4.2 ([12]) If G is

1. connected and $\{K_{1,3}, N\}$-free, then G is traceable.
2. 2-connected and $\{K_{1,3}, N\}$-free, then G is hamiltonian.

Note that if H is an induced subgraph of some graph S and G is H-free, then G is also S-free, for if G contained an induced S, it clearly would contain an induced H as well. Thus, we get the following corollary to Theorem 5.4.2. The graph B is shown in Figure 5.4.2.

Corollary 5.4.1 If G is a 2-connected graph that is $\{R, S\}$-free where $R = K_{1,3}$ and S is any one of N, C_3, Z_1, B, P_4, P_3, K_2 or K_1, then G is hamiltonian.

In the list of graphs given in Corollary 5.4.1, we note that if P_3 is forbidden, the graph must clearly be complete, thus it has any hamiltonian property you wish. Further, if the induced subgraphs of P_3 are forbidden, namely K_2 or K_1, we either have that G must be K_1 or G is empty. In either case we arrive at trivial cases that satisfy all hamiltonian properties. For this reason in what remains we will ignore P_3 and its induced subgraphs as simply being trivial and therefore "uninteresting".

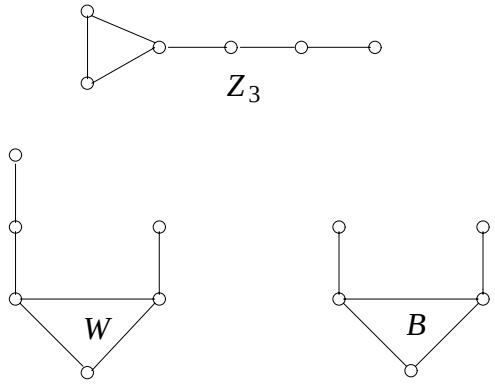


Figure 5.4.2. Common forbidden subgraphs.

After Theorem 5.4.2 was announced, the search for other familes of forbidden subgraphs began in earnest. Over the course of the next decade a variety of such results were discovered. We summarize the most important of these for our purposes in the next result. See Figure 5.4.2 for the graphs W and Z_3, where by Z_i we mean a triangle with a path containing exactly i edges from one of its vertices.

Theorem 5.4.3 If G is a 2-connected graph and G is

i. ([5]) $\{K_{1,3}, P_6\}$-free, or

ii. ([2]) $\{K_{1,3}, W\}$-free, or

iii. ([20]) $\{K_{1,3}, Z_3\}$-free and of order $p \geq 10$,
then G is hamiltonian.

Recently, a characterization of all pairs of graphs that, when forbidden, imply a 2-connected graph is hamiltonian was given in [16].

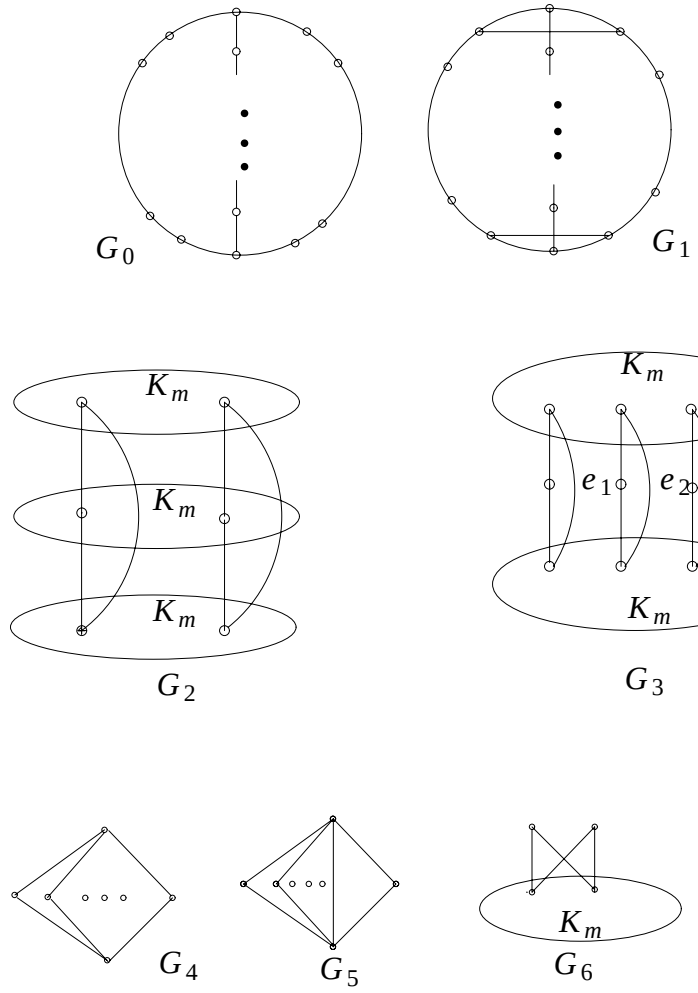


Figure 5.4.3. 2-Connected, nonhamiltonian graphs.

Theorem 5.4.4 Let R and S be connected graphs ($R, S \neq P_3$) and G a 2-connected graph of order $n \geq 10$. Then G is (R, S)-free implies G is hamiltonian if, and only if, $R = K_{1,3}$ and S is one of the graphs $C_3, P_4, P_5, P_6, Z_1, Z_2, Z_3, B, N$ or W.

Proof: That each of the pairs implies G is hamiltonian follows from Theorems 5.4.2 and 5.4.3 and our earlier remarks about induced subgraphs of forbidden graphs.

Now consider the graphs $G_0, \ldots, G_6$ of Figure 5.4.3. Each is 2-connected and nonhamiltonian (and really represents an infinite family of such graphs). Without loss of generality assume that R is a subgraph of G_1.

Case 1: Suppose that R contains an induced P_4.

Since G_4, G_5, and G_6 are all P_4-free, then S must be an induced subgraph of each of them. But if S is an induced subgraph of G_4, then either S is a star or S contains an induced C_4. However, G_5 is C_4-free, hence S must be a star. Since the only induced star in G_6 is $K_{1,3}$, we have that $S = K_{1,3}$.

Case 2: Suppose that R does not contain an induced P_4.

Then, using G_0 we see immediately that R must be a tree containing at most one vertex of degree 3 and since R contains no induced P_4, we see that $R = K_{1,3}$. Thus, for the remainder of the proof we assume without loss of generality that $R = K_{1,3}$.

Now, S must be an induced subgraph of G_1, G_2, and G_3 (each of which is claw-free). The fact that S is an induced subgraph of G_1 implies that S is a path or S is K_3, possibly with a path off each of its vertices. Suppose that S is a path. Since S is an induced subgraph of G_3 which is P_7-free, we see that if S is a path, it is one of P_4, P_5 or P_6.

We now assume that S contains a K_3, possibly with a path off each of its vertices. Note that G_3 is Z_4-free. Further, any triangle in G_2 with a 3-path off one of its vertices can have no paths off its other vertices (leaving Z_3, Z_2, Z_1, and K_3). Again examining G_2 we see it contains no triangle with a path of length 2 from one of its vertices and a path of length 1 from the other two vertices. Next, the graph G^*_3 obtained by deleting the three edges e_1, e_2 and e_3 from G_3 is still claw-free and contains no induced K_3 with a 2-path from two vertices. (leaving B or W). The only remaining possibility is a path of length 1 off each of the vertices of K_3, that is, the graph N. □

Note that similar results are known for traceable graphs as well as for several other hamiltonian type properties. These will be explored in more detail in the exercises.

Section 5.5 Other Types of Hamiltonian Results

In this section we wish to consider several other types of results concerning hamiltonian and hamiltonian-like graphs. These use conditions that are often very different from those we have seen thus far. We begin with an investigation of the powers of a graph.

The *nth power* G^n of a connected graph G is that graph with $V(G^n) = V(G)$ and in which uv is an edge of G^n if, and only if, $1 \le d_G(u, v) \le n$. The graphs G^2 and G^3 are called the *square* and *cube* of G, respectively. Figure 5.5.1 shows the *subdivision graph* of the graph $K_{1,3}$, that is, the graph $S(K_{1,3})$ obtained by *subdividing* each edge of $K_{1,3}$. The graph $S(K_{1,3})$ is formed from $K_{1,3}$ when each edge $e = xy$ is removed and a new vertex w is inserted along with the edges wx and wy. Figure 5.5.1 also shows the square of $S(K_{1,3})$.

Since higher powers of a graph G tend to contain more edges than G itself, it is reasonable to ask if these powers will eventually become hamiltonian, even if G is not. Nash-Williams and Plummer conjectured that this was indeed the case for the squares of 2-connected graphs. In the now classic paper [21], Fleischner verified that this was indeed the case.

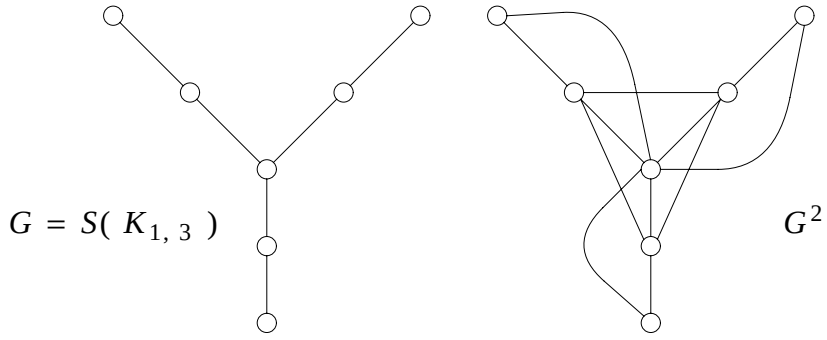


Figure 5.5.1. The graph $S(K_{1,3})$ and its square.

Theorem 5.5.1 Let G be a 2-connected graph. Then G^2 is hamiltonian.

The proof of Fleischner's theorem is beyond the scope of most texts and, hence, will not be presented. However, this work opened the door for others to investigate properties

in powers of graphs. In [7], Fleischner's result was strengthened to show that the square of every 2-connected graph is actually hamiltonian connected (see the exercises).

Let us switch attention now to hamiltonian properties of line graphs. In [30], Harary and Nash-Williams characterized when the line graph $L(G)$ of a graph G is hamiltonian. In order to do this, we define a *dominating circuit* C of a graph G to be a circuit with the property that every edge of G is incident to a vertex of C.

Theorem 5.5.2 Let G be a graph without isolated vertices. Then $L(G)$ is hamiltonian if, and only if, G is isomorphic to $K_{1,\,n}$ ($n \geq 3$) or G contains a dominating circuit.

The major impact of Theorem 5.5.2 has been its use in proving other results.

In [29], the idea of dominating circuits is used to help establish a result that combines ideas from throughout this section. Suppose that H is a connected graph and that $L(H)$ is not hamiltonian. Therefore, H must not contain a dominating circuit. Among all circuits of H, let C be one with the maximum number of vertices. Let $V = V(C)$ and let $U = V(H) - V$. Since C is not a dominating circuit, $< U >$ is not an empty graph. Thus, since H is connected, there exists at least one edge from U to V in H. With this in mind, we present the following lemma [29].

Lemma 5.5.1 No $U - V$ edge of H lies on a triangle T of H.

Proof. Suppose that the result failed to hold; then one of the following cases would have to occur.

Case 1: Exactly one vertex of the triangle T lies on C and, thus, a circuit longer than C would exist, a contradiction.

Case 2: Exactly two vertices of the triangle T lie on C and they are consecutive on C (say they are joined by e). Then deleting e and inserting the other two edges of T creates a circuit longer than C, again a contradiction.

Case 3: Exactly two vertices of T lie on C, but these two vertices are not consecutive. Then a longer circuit can be found using all of C and all of T, once again producing a contradiction.

This completes the cases, and, thus, the lemma follows. □

Using this lemma, we can obtain a result concerning hamiltonian properties in the second iterated line graph of a graph [29].

Theorem 5.5.3 Let G be a connected graph of order $p \geq 3$ which does not contain a vertex cutset consisting only of vertices of degree 2. Then $L^2(G)$ (that is, $L(L(G))$) is hamiltonian.

Proof. Let $H = L(G)$ and suppose that $L(H) = L^2(G)$ is not hamiltonian. Then, as above, there exists a partition of $V(H)$ as $V \cup U$ such that (by the lemma) no $U - V$ edge in H lies on a triangle. Let $e_1 f_1, \ldots, e_r f_r$ ($r \geq 1$) be the $U - V$ edges in H and let v_i be the common end vertex of e_i and f_i in G ($1 \leq i \leq r$). Since $e_i f_i$ does not lie in a triangle of H, $deg_G\ v_i = 2$, ($1 \leq i \leq r$). Further, since the graph

$$H - \{ e_i f_i \mid 1 \leq i \leq r \}$$

is disconnected, then $G - \{ v_i \mid 1 \leq i \leq r \}$ is also disconnected. Therefore, we reach a contradiction since $\{ v_i \mid 1 \leq i \leq r \}$ is a cut set of G consisting entirely of vertices of degree 2. Thus, the result follows. □

With the aid of Theorem 5.5.3, the following corollary, originally from Chartrand and Wall [8], is immediate.

Corollary 5.5.1 If G is a connected graph such that $\delta(G) \geq 3$, then $L^2(G)$ is hamiltonian.

Section 5.6 The Traveling Salesman Problem

Consider the dilemma of a traveling salesman. He must visit each city in his region and return to his home office on a regular basis. He seeks the route that allows him to visit each city at least once (exactly once would be even better) and return home, with the added property that this route also covers the least distance. Clearly, a weighted graph can be used to model the possible routes. Because we can insert edges with infinite distances, we can also restrict our attention to complete graphs.

Unfortunately for the traveling salesman, this problem is NP-complete (see, for example, [24]). Thus, efforts have concentrated on approximation algorithms and the use of heuristics to try to make some gains. The heuristic one usually thinks of first, the *nearest neighbor* approach, begins with a single vertex, adds the edge of minimum distance, and continues by trying to build from either end of this path by repeatedly taking the nearest neighbor. Unfortunately, to close the path into a cycle is often very expensive, and we can also ignore many short edges this way. Thus, this approach is very unreliable.

To try to overcome this difficulty, you might try beginning with some short cycle and expand this cycle by inserting the vertex that causes the cycle length to increase the least. This technique is called the *shortest insertion heuristic*. Once again, difficulties can arise. Part of the problem stems from the fact that an arbitrarily weighted graph need not satisfy the "reasonable rules" of distance. That is, the triangle inequality need not hold. However, if the triangle inequality does hold, some progress can be made. In the algorithm that follows, we assume both a single vertex and a K_2 are (degenerate) cycles.

Algorithm 5.6.1 The Shortest Insertion Algorithm
Input: A weighted graph $G = (V, E)$ satisfying the triangle inequality.
Output: A hamiltonian cycle C that approximates the salesman cycle.

1. Select any vertex and consider it a 1-cycle C_1 of G. Set $i \leftarrow 1$.
2. If $i = p$, then halt since $C = C_p$ is the desired cycle;
   else if C_i has been selected $(1 \le i \le p)$, then find a vertex v_i not on C_i that is closest to a consecutive pair of vertices w_i and w_{i+1} of C_i.
3. Let C_{i+1} be formed by inserting v_i between w_i and w_{i+1} on C_i and go to step 2.

The type of heuristics we have just considered constructs a tour whose value is hoped to be close to optimal. These are known as *tour construction heuristics*. Other heuristics take a tour and try to improve it (known as *tour improvement heuristics*). One of the most commonly used improvement procedures is that of edge exchanges.

In general, r edges in a tour are exchanged for r edges not in the tour, as long as the result remains a tour and the length of the new tour is less than the length of the old tour. Exchange procedures have come to be called *r-opt procedures*, where r is the number of edges exchanged at each iteration. Figure 5.6.1 illustrates the exchange for 2-opt and 3-opt.

In an r-opt algorithm, all feasible exchanges of r edges are tested until there is no exchange that improves the current situation. The solution is than said to be r-optimal (see [36]). In general, the larger the value of r, the more likely it is that the final tour is optimal. However, the number of operations necessary to test all possible r exchanges increases dramatically as the number of cities increase. For this reason, $r = 2$ and $r = 3$ are the most commonly used values.

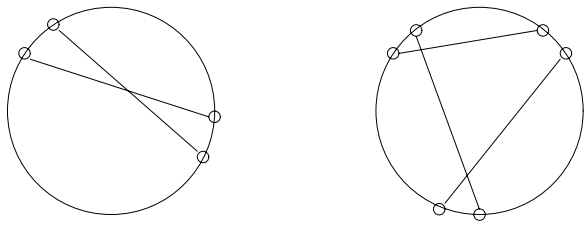


Figure 5.6.1. The 2-opt and 3-opt exchanges

Section 5.7 Short Cycles and Girth

In this section we explore a few results about nonhamiltonian cycles in graphs. Many of the same techniques that aided us in finding spanning cycles can be of use here. We begin with a result from [4].

Theorem 5.7.1 If G is a graph of order $p \geq 3$ and $5 \leq t \leq p$, then the graph G contains a C_t if, and only if, for every pair of nonadjacent vertices u and v such that $deg\ u + deg\ v \geq 2p - t$, $G + uv$ contains a C_t.

Proof. If G contains C_t, then clearly so does $G + uv$.

To prove the converse, suppose that $G + uv$ contains a C_t, but G does not. Then there exists a path P of length $t - 1$ in G joining u and v. Let $U = V(G) - V(P)$. Then it is clear that $|U| = p - t$. If u and v are each adjacent to all of U, then that still accounts for only $2(p - t)$ adjacencies. Thus, they must have at least t adjacencies on P. But now, we simply apply the technique used in the proof of Ore's theorem to produce a C_t using the vertices of P. □

We may also use neighborhoods to obtain results about nonspanning cycles.

Theorem 5.7.2 ([19]) If G is 2-connected of order p satisfying the property that for every pair of nonadjacent vertices u and v

$$|N(u) \cup N(v)| \geq s,$$

then G contains a cycle of length at least $s + 2$ or (if $p < s + 2$) G is complete.

Proof If $p < s + 2$, then the neighborhood condition implies that G must be complete.

Now, assume that $p \geq s + 2$. Let G be a 2-connected graph of order p satisfying the neighborhood condition. Let $x_1, x_2, \ldots, x_t$ be the vertices in order along a path P of maximum length in G. Let x_i and x_j (possibly $i = j$) be vertices on P that are adjacent to x_1, and suppose that, among all possible longest paths P and vertices x_i, we have chosen ones that maximize i. Then x_1 and x_{j-1} are adjacent only to vertices in

$$W = \{ x_1, \ldots, x_i \}$$

(since clearly x_{j-1} is an end vertex of a path having the same length as P). Suppose $i \leq s + 1$. Then by the neighborhood condition, we see that x_1 and x_{j-1} must be adjacent. Taking j to be $i, i - 1, \cdots$ in turn, we see that x_1 is adjacent to each of $x_2, \ldots, x_i$. All vertices $x_1, \ldots, x_{i-1}$ are, thus, end vertices of paths of the same length as P, and so they are adjacent only to vertices of W, which contradicts the 2-connectedness of G. Thus, $i \geq s + 2$, and G contains a cycle of length $s + 2$ or more. □

We now turn our attention specifically to the shortest cycle in a graph. The *girth* of a graph G, denoted $g(G)$, is the length of a shortest cycle in G. We can bound the order of G from below using the girth and the minimum degree of G. Let $f(g, \delta)$ be the minimum order of a graph G with $g(G) \geq g$ and $\delta(G) \geq \delta$. It is not immediately obvious that there exist graphs satisfying these restrictions. We shall provide an upper bound on $f(g, \delta)$ (and, hence, prove the existence of such graphs) in Chapter 9 using probabilistic techniques. For now, we shall accept their existence and try to learn more about them. One can obtain an easy lower bound on $f(g, \delta)$.

Theorem 5.7.3 For $\delta, g \geq 3$,

$$f(g, \delta) \geq \begin{cases} \dfrac{\delta(\delta - 1)^d - 2}{\delta - 2} & \text{if } g = 2d + 1, \\ \dfrac{(\delta - 1)^d - 1}{\delta - 2} & \text{if } g = 2d. \end{cases} \tag{5.7.1}$$

Proof. If g is odd, say $g = 2d + 1$, note that the subgraph induced by those vertices within a distance d of a vertex x must be a tree, since any cycle within this graph would have length less than g. Since there are at least $\delta(\delta - 1)^j$ vertices at distance $j + 1$ from x, we see that

$$f(g, \delta) \geq 1 + \delta \sum_{j=0}^{d} (\delta - 1)^j.$$

Since this is a geometric series, the desired result follows.

Now, suppose that g is even, say $g = 2d$. Then consider the vertices within a distance d of the edge xy and proceed in a manner similar to the odd case to obtain the bound. □

In certain cases we can find the exact value of $f(g, \delta)$, especially when the graph is δ-regular. For example, the 2-regular graphs of girth g and minimum order p are simply the cycles C_p. It is also easy to see that the r-regular graphs of girth $g = 3$ and minimum order are simply the complete graphs K_{r+1}. When $g = 4$ and the graph is r-regular, then the complete bipartite graphs $K_{\delta, \delta}$ have minimum order and these conditions. It has become common to call r-regular graphs of girth g and minimum order the (g, r)-*cages.* It turns out that (g, r)-cages are difficult to find. The first really interesting case is the $(5, 3)$-cage; that is, we want a 3-regular graph of girth 5 and minimum possible order. Using our tree structure (from the proof of Theorem 5.7.3), we see that such a graph has at least 10 vertices. The graph of Figure 5.7.1 has exactly 10 vertices and satisfies the other conditions as well (a fact you must convince yourself of). The interesting point here is that this graph is isomorphic to the Petersen graph and is the unique $(5, 3)$-cage (see the exercises).

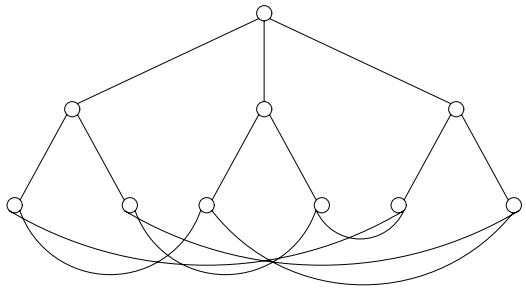


Figure 5.7.1. The $(5, 3)$-cage.

We next state a result summarizing some of the best known work on bounding $f(g, \delta)$.

Theorem 5.7.4

1. For $\delta \geq 2$, $f(4, \delta) = 2\delta$.

2. For $\delta \geq 2$, $f(g, \delta) \geq \delta^2 + 1$. Furthermore, for $\delta \neq 57$, equality holds if, and only if, $\delta = 2$, 3 or 7.
3. When $\delta = p^m + 1$, for some prime p and positive integer m, then

$$f(6, \delta) = \frac{2(\delta - 1)^3 - 2}{\delta - 2}.$$

We conclude with a table of orders for the known cages (Table 5.7.1).

r/g	5	6	7	8	9	12
3	10	14	24	30		126
4	19	26				
5	30	40	50			

Table 5.7.1 Known orders for cages.

Section 5.8 Disjoint Cycles

In this section we wish to explore the situation of when several cycles may be found in a graph. At this point in time we only care that these cycles are disjoint (both vertices and edges). We begin with a result due to Pósa [43]. Our concern is in finding a pair of vertex disjoint cycles of unspecified order. We define $s(n)$ to be the minimum number of edges so that every graph on n vertices contains two vertex disjoint cycles.

Theorem 5.8.1 For $n \geq 6$, $s(n) = 3n - 5$.

Proof. We first note that $3n - 6$ edges is not enough to guarantee the existence of two vertex disjoint cycles. To see this, we need only consider $K_{1, 1, 1, n-3}$ and note that any cycle in this graph must use at least two of the vertices in the partite sets of cardinality 1. Hence, two disjoint cycles cannot exist.

To establish that every $(n, 3n - 5)$ graph must contain two disjoint cycles, we proceed inductively on n. For $n = 6$, we note that $3n - 5 = 13$. This means that the graph in question is K_6 minus two edges. It is easy to check that no matter how these two edges are deleted, two vertex disjoint triangles remain.

Now, assume the result is true for all such graphs of order less than n and consider an $(n, 3n - 5)$ graph G. Since $\sum deg\ v_i = 6n - 10$, we see that G must have a vertex of degree at most 5. Call such a vertex x. We now consider cases for the possible values of $deg\ x$.

Suppose that $deg\ x = 5$ and let $N(x) = \{ v_1, v_2, \ldots, v_5 \}$. If $| E(<N[x]>) | \geq 13$, then as in the anchor step, G contains two disjoint triangles. Otherwise, some v_i, say v_1, has two nonadjacencies in $N(x)$. Without loss of generality, say v_1 is not adjacent to v_2 and v_3.

Now, consider the graph $H = G - x + \{ v_1v_2, v_1v_3 \}$. Note that H is an $(n - 1, 3n - 8)$ graph and so by the induction hypothesis, H must contain two disjoint cycles, say C_1 and C_2. If C_1 and C_2 do not use the edges we inserted to form H, then they must also exist in our original graph G. Thus, since at most one of these cycles can use the edges we inserted, we must have one cycle intact inside G. We now wish to show that the other cycle from H can be modified to form a second cycle in G.

If this cycle uses v_1v_2 (or v_1v_3) only, then replacing this edge with the vertex x and edges v_1x, xv_2 produces the desired cycle in G. If both v_1v_2 and v_1v_3 are used, replace them with x and xv_2, xv_3 to form the desired cycle. In any case, two disjoint cycles in G are found.

If $deg\ x = 4$, a similar argument applies. If $deg\ x \leq 3$, then when x is removed from G, no other edges need be added to be able to apply the inductive hypothesis, and two disjoint cycles are found in $G - x$ immediately. Thus, in all cases, the result holds. □

This result is sharp since the graph $K_3 + nK_1$ has $p = n + 3$ vertices, $3p - 6$ edges and does not contain 2 disjoint cycles.

For $K_{1,3}$-free graphs, Matthews [38] proved that if $q \geq p + 6$, then the graph contains 2 disjoint cycles. We will prove a weaker (and much shorter) result.

Theorem 5.8.2 If G is a (p, q) $K_{1,3}$-free graph with $q \geq p + 6$, then G contains 2 disjoint cycles or $\Delta(G) < 6$.

Proof: Suppose that $\Delta(G) \geq 6$ and also that the result fails, hence any two cycles in G intersect. Since $deg\ u \geq 6$ for some vertex u, then by Theorem 1.6.2 and the fact that G is $K_{1,3}$-free, we see that $N(u)$ must contain a triangle, say T. But u and any three of the remaining vertices in $N(u) - V(T)$ cannot induce a $K_{1,3}$, thus a triangle disjoint from T exists. □

This result is also sharp since the graph K_5 with a path of length n attached to any one of its vertices has $p + 5$ edges, is $K_{1,3}$-free and does not contain 2 disjoint cycles.

For the case of finding k disjoint cycles, it was shown in [14] that for $k \geq 1$ and $p \geq 24k$, then every graph with $q \geq (2k - 1)p - 2k^2 + k$ contains either k disjoint cycles or $G = K_{2k-1} + (p - 2k + 1)K_1$. This result was improved for $K_{1,3}$-free graphs by Chen, Markus and Schelp [9].

Theorem 5.8.3 Let G be a $K_{1,3}$-free graph and let $k \geq 1$. If

$$q \geq p + (3k - 1)(3k - 4)/2 + 1$$

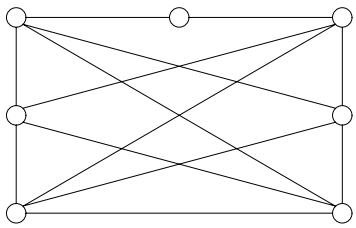
then G contains k disjoint cycles.

Finally, Corrádi and Hajnal [10] showed the following.

Theorem 5.8.4 If G is an (n,q) graph with $n \geq 3k$ and $\delta(G) \geq 2k$, then G contains k disjoint cycles.

Exercises

1. Prove Corollary 5.1.2.
2. Prove Theorem 5.1.2.
3. Determine the complexity of Algorithm 5.1.1.
4. Use each of Algorithms 5.1.1, 5.1.2 and 5.1.3 to find an eulerian cycle in the graph below.



5. Determine an algorithm to accomplish the splitting away of two edges.
6. If H is the graph obtained from G by splitting away $e_1 = vw$ and $e_2 = vx$, prove that H is connected if, and only if, G is connected and $\{ e_1, e_2 \}$ does not form a cut set.

7. Prove that a nontrivial connected graph G is eulerian if, and only if, every edge of G lies on an odd number of cycles.
8. Prove that a nontrivial connected digraph D is eulerian if and only if E can be partitioned into subsets $E_1, E_2, \ldots, E_k$ such that the graph induced by E_i is a cycle for each i, $(1 \le i \le k)$.
9. Prove Observations 1 and 2.
10. Show that if $G = (V, E)$ is hamiltonian, then for every proper subset S of V, the number of components in $G - S$ is at most $|S|$
11. Show that if G is not 2-connected, then G is not hamiltonian.
12. Characterize when the graph $K_{p_1,p_2,\ldots,p_n}$ is hamiltonian.
13. Prove or disprove: If G and H are hamiltonian, then $G \times H$ is hamiltonian.
14. Prove or disprove: If G and H are hamiltonian, then $G[H]$ is hamiltonian.
15. Let the n-cube be the graph $Q_n = K_2 \times Q_{n-1}$ (where $Q_1 = K_2$). Prove that if $n \ge 2$, Q_n is hamiltonian.
16. Let G be a graph with $\delta(G) \ge 2$. Show that G contains a cycle of length at least $\delta(G) + 1$.
17. Suppose that G is a (p, q) graph with $p \ge 3$. Show that if $q \ge \dfrac{p^2 - 3p + 6}{2}$, then G is hamiltonian.
18. Show that if G is a (p, q) graph with $q \ge \binom{p-1}{2} + 3$, then G is hamiltonian connected.
19. Show that if G is hamiltonian connected, then G is 3-connected.
20. Show that if a (p, q) graph G is hamiltonian connected and if $p \ge 4$, then $q \ge \left\lfloor \dfrac{3p + 1}{2} \right\rfloor$.
21. Give an example of a graph that is pancyclic but not panconnected.
22. Find an example of a graph that is hamiltonian connected but not panconnected.
23. Find an example of a graph that is pancyclic but not vertex pancyclic.

24. Can we remove the restriction that D be strongly connected from Meyniel's theorem?

25. Show that every complete graph with directed edges is traceable.

26. Show that K_n with strong directed edges is vertex pancyclic.

27. Give an example of a hamiltonian connected digraph that satisfies the conditions of Theorem 5.4.2 but does not have $od\ v \geq \frac{p+1}{2}$ and $id\ v \geq \frac{p+1}{2}$ for every vertex v.

28. Show that the Petersen graph is homogeneously traceable nonhamiltonian and also hypohamiltonian.

29. Show that homogeneously traceable nonhamiltonian graphs exist for all orders $p \geq 9$.

30. Show that if $G = (V, E)$ is a homogeneously traceable nonhamiltonian graph and $x \in V$, then x is adjacent to at most one vertex of degree 2.

31. Show that if $C_{p-1}(G) = K_p$, then G is traceable.

32. Prove Corollary 5.4.2.

33. Prove that the graph G^2 of Figure 5.5.1 is not hamiltonian.

34. Show (without using Fleischner's theorem) that if G is 2-connected, then G^3 is hamiltonian. (Hint: Consider spanning trees).

35. Use Fleischner's theorem to show that if G is 2-connected, then G^2 is hamiltonian connected. (Hint: Consider five copies of G along with two additional vertices x and y joined to an arbitrary pair of vertices u and v in each copy of G).

36. Prove Theorem 5.5.2.

37. Prove that if G is a graph of order $p \geq 3$ such that the vertices of G can be labeled $v_1, v_2, \ldots, v_p$ so that

$$\begin{array}{l} j<k,\ \ j+k \geq p,\ \ v_j v_k \notin E(G) \\ \quad deg\ v_j \leq j,\ \ deg\ v_k \leq k-1 \end{array} \quad \Rightarrow \quad deg\ v_j + deg\ v_k \geq p.$$

then G is hamiltonian.

38. Let G be a graph of order $p \geq 3$, the degrees d_i of whose vertices satisfy $d_1 \leq d_2 \leq , \ldots, \leq d_p$. If

$$d_j \leq j < \frac{p}{2} \quad \Rightarrow \quad d_{p-j} \geq p - j,$$

then G is hamiltonian.

39. Prove that if G is a graph of order $p \geq 3$ such that for every integer j with $1 \leq j < \frac{p}{2}$, the number of vertices of degree not exceeding j is less than j, then G is hamiltonian.

40. Prove that if G has order $p \geq 3$ and if $k(G) \geq \beta(G)$ = the maximum number of mutually nonadjacent vertices, then G is hamiltonian.

41. (Ghouila-Houri [25]) Let D be a strongly connected digraph such that $deg\ x \geq p$ for every vertex x of D. Prove D is hamiltonian.

42. (Woodall [48]) Let D be a digraph of order $p \geq 3$ such that whenever x and y are distinct vertices and $x \rightarrow y$ is not an arc of D, then $od\ x + id\ y \geq p$. Prove D is hamiltonian.

43. (Meyniel [39]) Let D be a strongly connected digraph of order $p \geq 3$ such that for every distinct pair of nonadjacent vertices x and y, $deg\ x + deg\ y \geq 2p - 1$. Prove D is hamiltonian.

44. (Overbeck-Larisch [42]) Let D be a digraph of order $p \geq 2$ such that for every pair of distinct vertices x and y such that $x \rightarrow y$ is not an arc of D, $od\ x + id\ y \geq p + 1$. Prove D is hamiltonian connected.

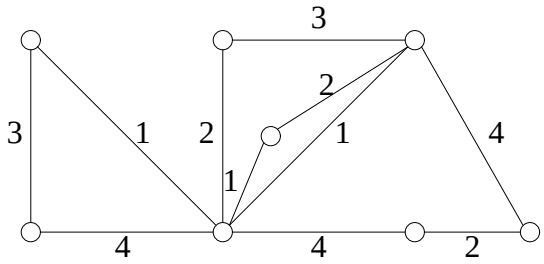
45. Determine all pairs of graphs (R, S) that when forbidden, imply a 2-connected graph is pancyclic.

46. Determine all pairs of graphs (R, S) that when forbidden, imply a 2-connected graph is panconnected.

47. Determine all pairs of graphs (R, S) that when forbidden, imply a connected graph is traceable.

48. Show that the only single graph that when forbidden, implies a 2-connected graph is hamiltonian, is P_3.

49. Determine the minimum salesman's walk in the following graph.



50. Show that the greedy approach to the traveling salesman problem can be arbitrarily bad.

51. Prove that the graph of Figure 5.7.1 is the unique $(5,\ 3)$-cage and is isomorphic to the Petersen graph.

52. Find the $(6,\ 3)$-cage and show that it is unique.

53. Prove Theorem 5.7.4(1).

54. Prove that if G is a $K_{1,3}$-free graph that does not contain k disjoint cycles, then $\Delta(G) \leq 3k - 1$.

55. Prove that if G is a $K_{1,3}$-free graph with $\Delta(G) \leq 5$ that G contains two disjoint cycles.

56. Prove that if a (p, q)-graph G is $K_{1,3}$-free, $k \geq 1$ and $q \geq p + (3k - 1)(3k - 4)/2 + 1$, then G contains k disjoint cycles.

References

1. Albertson, M. O., Finding Hamiltonian Cycles in Ore Graphs, preprint.

2. Bedrossian, P., Forbidden Subgraph and Minimum Degree Conditions for Hamiltonicity, Ph.D. Thesis, Memphis State University, 1991.

3. Bondy, J. A., Pancyclic Graphs. *J. Combin. Theory*, 11B(1971), 80 – 84.

4. Bondy, J. A., and Chvátal, V., A Method in Graph Theory. *Discrete Math.* 15(1976), 111 – 136.

5. Broersma, H.J., Veldman, H.J., Restrictions on Induced Subraphs Ensuring Hamiltonicity or Pancyclicity of $K_{1,3}$-free Graphs, *Contemporary Methods in Graph Theory* (R. Bodendiek), BI-Wiss.-Verl., Mannheim-Wien-Zurich, (1990) 181 – 194.

6. Chartrand, G., Gould, R. J., and Kapoor, S. F., On Homogeneously Traceable Nonhamiltonian Graphs. *Annals of the N. Y. Acad. of Sci.,* Vol. 319(1979), 130 – 135.

7. Chartrand, G., Hobbs, A., Jung, A., Kapoor, S.F., and Nash-Williams, C. St. J. A., The Square of a Block is Hamiltonian Connected. *J. Combin. Theory,* 16B(1974), 290 – 292.

8. Chartrand, G. and Wall, C. E., On the Hamiltonian Index of a Graph. *Studia Sci. Math. Hungar.,* 8(1973), 43 – 48.

9. Chen, G., Markus, L.R., and Schelp, R. H., Vertex Disjoint Cycles for Star Free Graphs. *Australasian J. of Combinatorics,* 11(1995) 157-169.

10. Corrádi, K. and Hajnal, A., On the Maximal Number of Independent Circuits in a Graph. *Acta Math. Acad. Sci. Hungar.,* 14(1963), 423 – 439.

11. Dirac, G. A., Some Theorems on Abstract Graphs. *Proc London Math. Soc.,* 2(1952), 69 – 81.

12. Duffus, D., Gould, R. J., and Jacobson, M. S., Forbidden Subgraphs and the Hamiltonian Theme. *The Theory and Applications of Graphs,* Ed. by Chartrand et al., Wiley-Interscience, New York (1981), 297 – 316.

13. Edmonds, J., and Johnson, E. L., Matching, Euler Tours and the Chinese Postman. *Math. Programming,* 5(1973), 88 – 124.

14. Erdös, P. and Pósa, L., On the Maximal Number of Disjoint Circuits of a Graph, *Math. Debrecen* 9 (1962), 3 – 12.

15. Euler, L., Solutio Problematis ad Geometriam Situs Pertinentis. *Comment. Acadamiae Sci. I. Petropolitanae,* 8(1736), 128 – 140.

16. Faudree, R.J., Gould, R.J., Characterizing Forbidden Pairs for Hamiltonian Properties, *Discrete Math.* (to appear).

17. Faudree, R.J., Gould, R.J., Jacobson, M.S. and L. Lesniak, Neighborhood Unions and a Generalization of Dirac's Theorem. *Discrete Math.* 105, (1992), 61 – 71.

18. Faudree, R. J., Gould, R. J., Jacobson, M. S., and Schelp, R. H., Neighborhood Unions and Hamiltonian Properties of Graphs. *J. Combin. Theory B* (in press).

19. Faudree, R. J., Gould, R. J., Jacobson, M. S., and Schelp, R. H., Extremal Problems Involving Neighborhood Unions. *J. Graph Theory,* 11 (1987), 555 – 564.

20. Faudree, R.J., Gould, R.J, Ryjacek, Z., Schiermeyer, I., Forbidden Subgraphs and Pancyclicity (preprint).

21. Fleischner, H., The square of Every Two-Connected Graph is Hamiltonian. *J. Combin. Theory,* 16B(1974), 29 – 34.

22. Fleury [see Lucas, E., *Recreations Mathematiques IV*, Paris, 1921].

23. Fraisse, P., A New Sufficient Condition for Hamiltonian Graphs. *J. Graph Theory,* 10(1986), 405 – 409.

24. Garey, M. R., and Johnson, D. S., *Computers and Intractability*. W. H. Freeman and Co., San Francisco (1979).

25. Ghouila-Houri, A., Une Condition Suffisante d'existence d'un Circuit Hamiltonien. *C. R. Acad. Sci. Paris,* 156(1960), 495 – 497.

26. Goodman, S. E., and Hedetniemi, S. T., Eulerian Walks in Graphs. *SIAM J. Comput.,* 2(1973), 16 – 27.

27. Goodman, S. E., and Hedetniemi, S. T., Sufficient Conditions for a Graph to be Hamiltonian. *J. Combin. Theory*, B(1974), 175 – 180.

28. Gould, R. J., *Updating the Hamiltonian Problem - A Survey*, Journal of Graph Theory, Vol. 15, No. 2, (1991), 121-157.

29. Gould, R. J., and Hendry, G., personnel communication (1984).

30. Harary, F., and Nash-Williams, C. St. J. A., On Eulerian and Hamiltonian Graphs and Line Graphs. *Canad. Math. Bull.,* 8(1965), 701 – 710.

31. Herz, J. C., Duby, J. J. and Vigué, F., Recherche Systé Matique des Graphs Hypohamiltoniens. *Theory of Graphs,* International Symposium, Rome (1966) Gordon and Breach, New York (1967), 153 – 159.

32. Herz, J. C., Gaudin, T. and Rossi, P., Solution de Probleme No. 29. *Rev. Francaise Rech. Operationelle*, 8(1964), 214 – 218.

33. Hierholzer, C., Ueber die Möglichkeit, einen Linienzug Ohne Wiederholung und Ohne Unterbrechnung zu Umfahren. *Math. Ann.,* 6(1873), 30 – 42.

34. Kwan, M. K., Graphic Programming Using Odd or Even Points. *Chinese Math.,* 1(1962), 273 – 277.

35. Lesniak, L., Neighborhood Unions and Graphical Properties. *Graph Theory, Combinatorics, and Applications* (ed. Alavi, Chartrand, Oellerman, Schwenk) Vol. 2 (1991), 783 – 800.

36. Lin, S., Computer Solutions of the Traveling Salesman Problem. *Bell System Tech. J.* 44 (1965), 2245 – 2269.

37. Lindgren, W. F., An Infinite Class of Hypohamiltonian Graphs. *Amer. Math. Monthly,* 74(1967), 1087 – 1089.

38. Matthews, M.M., Extremal Results for $K_{1,3}$-free Graphs, *Congressus Numer.* 49(1985), 49 – 55.

39. Meyniel, M., Une Condition Suffisante d'existence d'un Circuit Hamiltonien dans un Graph Oriente. *J. Combin. Theory,* 14B(1973), 137 – 147.

40. Newman, D. J., A Problem in Graph Theory. *American Math. Monthly,* 65(1958), 611.

41. Ore, O., A Note on Hamilton Circuits. *Amer. Math. Monthly,* 67(1960), 55.

42. Overbeck-Larisch, M., Hamiltonian Paths in Oriented Graphs. *J. Combin. Theory,* 21B(1976), 76 – 80.

43. Pósa, L. (See Erdös, P., Extremal Problems in Graph Theory. *A Seminar in Graph Theory.* Holt, Rinehart, and Winston, New York (1967).

44. Skupien, Z., Homogeneously Traceable and Hamiltonian Connected Graphs, manuscript (1976).

45. Tucker, A. C., A New Application Proof of the Euler Circuit Theorem. *Amer. Math. Monthly,* 83(1976), 638 – 640.

46. Veblen, O., An Application of Modular Equations in Analysis Situs. *Ann. Math.,* (2), 14(1912-13), 86 – 94.

47. Williamson, J. E., Panconnected Graphs II. *Period. Math. Hungar.* 8(1977), 105 – 116.

48. Woodall, D. R., Sufficient Conditions for Circuits in Graphs. *Proc. London Math. Soc.,* 24(1972), 739 – 755.

Chapter 6
Planarity

Section 6.1 Euler's Formula

In Chapter 1 we introduced the puzzle of the three houses and the three utilities. The problem was to determine if we could connect each of the three utilities with each of the three houses so that none of the utility lines crossed. We attempted a drawing of the graph model $K_{3,3}$ in Figure 1.1.2. In this chapter we will show that no such drawing is possible in the plane.

A (p, q) graph G is said to be *embeddable in the plane* or *planar* if it is possible to draw G in the plane so that the edges of G intersect only at end vertices. If such a drawing has been done, we say that a *plane* embedding of the graph has been found.

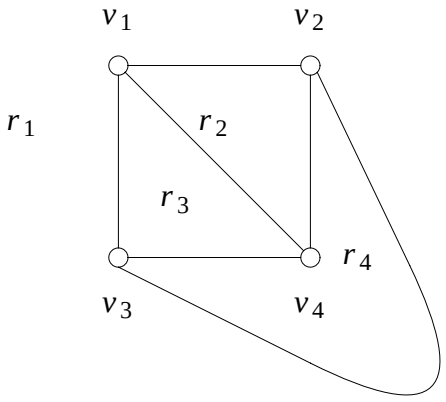


Figure 6.1.1. A plane embedding of K_4.

Given a plane graph G, a *region* of G is a maximal section of the plane for which any two points may be joined by a curve. Intuitively, a region is the connected section of the plane bounded (often enclosed) by some set of edges of G. In Figure 6.1.1, the regions of G are labeled r_1, r_2, r_3 and r_4. The region r_1 is bounded by the edges v_1v_2, v_2v_3 and v_3v_1, while r_2 is bounded by v_1v_2, v_2v_4 and v_4v_1. Further, note that every plane graph has a region similar to the region r_1. This region, which is actually not enclosed, is called the *exterior region.* It is also the case that given any planar graph G, there is an embedding of G with any region as the exterior region.

One of the most useful results for dealing with planar graphs relates the order, size and number of regions of the graph. This result is originally from Euler [1].

Theorem 6.1.1 If G is a connected plane (p, q) graph with r regions, then $p - q + r = 2$.

Proof. We proceed by induction on the size of the graph. If $q = 0$, then since G is connected, $p = 1$ and $r = 1$, and the result follows. Now, assume the result is true for all connected plane graphs with fewer than q edges ($q \geq 1$) and suppose that G is a connected plane (p, q) graph. If G is a tree, then $p = q + 1$ and $r = 1$, and the result follows easily. If G is not a tree, then let e be an edge of G on some cycle C and consider $H = G - e$. Since e is not a bridge, H is clearly connected and planar. Further, H has p vertices, $q - 1$ edges and $r - 1$ regions; thus, by the induction hypothesis,

$$p - (q - 1) + (r - 1) = 2.$$

But then, inserting e back into H to form G results in one more edge and one more region, so G clearly satisfies the formula; that is,

$$p - q + r = 2. \qquad \square$$

Since p, q and 2 are all fixed constants for a given planar graph G, it follows from Euler's formula that any two planar embeddings of a connected graph must have the same number of regions. Thus, we can simply refer to the number of regions of the planar graph without regard to the embedding.

It seems reasonable to think that we cannot simply continue to insert edges into a plane graph indefinitely while still maintaining its planarity. Thus, we would like to obtain an upper bound on the number of edges that a planar graph of given order can contain. If we continue to insert edges into a planar graph G, until, for every pair of nonadjacent vertices x and y, the graph $G + xy$ is nonplanar, we say the graph G is a *maximal planar graph.* But under what conditions can we insert an edge into G and have it remain planar? If we consider a plane embedding of G with a region r and if r is bounded by a cycle containing four or more edges, then we can certainly insert one of the missing edges between two vertices on this boundary. Further, there is no problem using this region for the embedding of the new edge. Thus, as long as we can find a region whose boundary contains four or more edges (or if the region is not enclosed, as in the case of a tree), we can continue to insert edges. For this reason, maximal planar graphs are sometimes called *triangulated planar graphs* or simply *triangulations* (see Figure 6.1.2). With this in mind, we can now develop a relationship between the order and size of maximal planar graphs.

Theorem 6.1.2 If G is a maximal planar (p, q) graph with $p \geq 3$, then

$$q = 3p - 6.$$

Proof. Let r be the number of regions of G. Note that the boundary of every region is a triangle and that each edge of G lies on the boundary of two such regions. Thus, if we

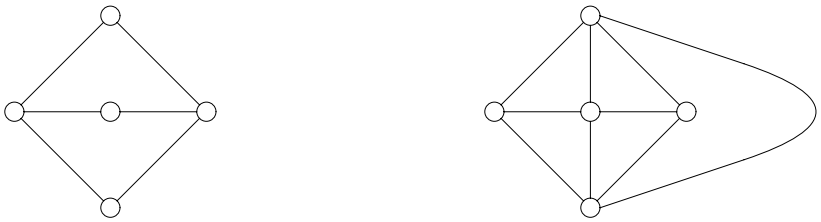


Figure 6.1.2. Inserting edges into $K_{2,3}$ to obtain a maximal planar graph.

sum the number of edges on the boundary of a region over all regions, we obtain $3r$. We note that this sum also counts each edge twice; thus, we obtain the relation $3r = 2q$. Now, applying Euler's formula, we see that

$$p - q + \frac{2q}{3} = 2$$

or

$$q = 3p - 6. \quad \square$$

Corollary 6.1.1 If G is a planar (p, q) graph with $p \geq 3$, then

$$q \leq 3p - 6.$$

Proof. Merely add edges to G until it is maximal planar. The resulting graph has order p and size $q^* \geq q$. Further, from Theorem 6.1.2 we see that

$$q \leq q^* = 3p - 6,$$

and the result follows. □

Another important consequence of Euler's formula is the following result.

Corollary 6.1.2 Every planar graph G contains a vertex of degree at most 5, that is, $\delta(G) \leq 5$.

Proof. Suppose $G = (V, E)$ is a planar (p, q) graph with $V = \{ x_1, \ldots, x_p \}$. If $p \leq 6$, the result must hold, so suppose that $p \geq 7$. We know that $q \leq 3p - 6$; hence,

$$\sum_{i=1}^{p} deg\ x_i = 2q \leq 6p - 12.$$

But if all p vertices of G have degree 6 or more, then $2q \geq 6p$, a contradiction. Hence, G contains a vertex of degree 5 at most. □

We are finally in a position to determine two very important nonplanar graphs. The first is the graph of our utilities problem, namely $K_{3,3}$, and the second is the complete graph K_5 (see Figure 6.1.3).

Corollary 6.1.3 The graph $K_{3,3}$ is nonplanar.

Proof. Suppose that $K_{3,3}$ is planar and let G be a plane embedding of $K_{3,3}$. Since $K_{3,3}$ has no triangles (in fact, no odd cycles at all), every region of G must contain at least four edges. Thus,

$$4r \leq 2q = 18.$$

But then, $r \leq 4$. Hence, by Euler's formula,

$$2 = p - q + r \leq 6 - 9 + 4 = 1,$$

a contradiction. □

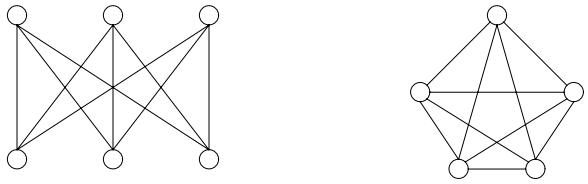


Figure 6.1.3. The nonplanar graphs $K_{3,3}$ and K_5.

Corollary 6.1.4 The graph K_5 is nonplanar.

Proof. If K_5 were planar, then by Corollary 6.1.1, K_5 would satisfy the bound $q \leq 3p - 6$. But then we would have that

$$10 = |E(K_5)| \leq 3(5) - 6 = 9,$$

again producing a contradiction. □

Section 6.2 Characterizations of Planar Graphs

There are actually a surprisingly large number of different characterizations of planar graphs. These range from topological in nature, to geometric, to structural. They entail techniques ranging from finding certain kinds of subgraphs to constructing alternate kinds of graphs and recognizing properties of these graphs. We shall consider three basic characterizations in this section and another in Chapter 9.

The oldest and most famous of all the characterizations of planar graphs is that of Kuratowski [5] and is topological in nature. The fundamental idea rests on the graphs $K_{3,3}$ and K_5 that we showed were nonplanar in the previous section. Kuratowski proved that these are the two "fundamental" nonplanar graphs. That is, he showed that any nonplanar graph must contain as a subgraph a graph that is closely related to at least one of these two graphs and that planar graphs do not contain such subgraphs. To understand what we mean by closely related requires a bit more terminology.

Recall that by a *subdivision* of an edge $e = xy$, we mean that the edge xy is removed from the graph and a new vertex w is inserted in the graph, along with the edges wx and wy. The essential idea behind Kuratowski's theorem is: If there is a problem drawing G in the plane because the edge xy will cross other edges, then there is still a problem drawing the path x, w, y (and conversely). Thus, in a sense, the original edge is a simpler structure to deal with, but the subdivided edge is just as large a problem as far as planarity is concerned.

We say a graph H is *homeomorphic from* G if either H is isomorphic to G or H is isomorphic to a graph obtained by subdividing some sequence of edges of G. We say G is *homeomorphic with* H if both G and H are homeomorphic from a graph F. The relation "homeomorphic with" is an equivalence relation on graphs (see the exercises). Hence, the set of graphs may be partitioned into classes; two graphs belong to the same class if, and only if, they are homeomorphic with each other. In Figure 6.2.1 we show a graph homeomorphic from $K_{3,3}$. Can you identify the vertices that correspond to the subdivided edges and the vertices that correspond to those of the original $K_{3,3}$?

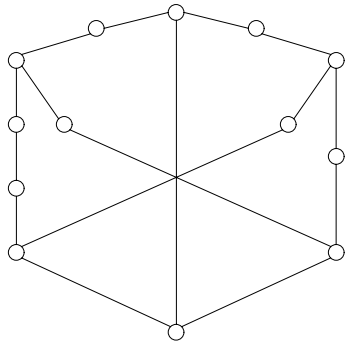


Figure 6.2.1. A subdivision of $K_{3,3}$.

We now present Kuratowski's characterization of planar graphs.

Theorem 6.2.1 (Kuratowski's Theorem [5]) A graph G is planar if, and only if, G contains no subgraph homeomorphic with K_5 or $K_{3,3}$.

Proof. Clearly a graph is planar if, and only if, each of its components is planar. Thus, we will restrict our attention to connected graphs. It is also straightforward to show that a graph is planar if, and only if, each of its blocks is planar (see exercise 4 in Chapter 6). Thus, we further restrict our attention to blocks other than K_2.

We know from our previous work that if a graph is planar then it does not contain a subgraph homeomorphic with K_5 or $K_{3,3}$. Thus, to complete the proof, it is sufficient to show that if a block contains no subgraph homeomorphic with K_5 or $K_{3,3}$, then it is planar. Suppose instead that this is not the case. Then among all nonplanar blocks not containing subgraphs homeomorphic with either K_5 or $K_{3,3}$, let G be one of minimum size.

We first claim that $\delta(G) \geq 3$. To prove this claim, note that since G is a block, it contains no vertices of degree 1. Suppose that G contained a vertex v with $deg\ v = 2$ and further suppose that v is adjacent with u and w. Consider two cases:

Case 1: Suppose uw is an edge of G. Then $G - v$ is still a block. Further, since $G - v$ is a subgraph of G, it follows that $G - v$ contains no subgraph homeomorphic with K_5 or $K_{3,3}$. But G is a nonplanar block of minimum size with this property; thus, $G - v$ must be planar. However, in any plane embedding of $G - v$, the vertex v and its incident edges vu and vw can be inserted so that the resulting graph (namely, G) is also plane. This contradicts the fact that G was nonplanar.

Case 2: Suppose that uw is not an edge of G. Then the graph $G_1 = G - v + uw$ is a block with size less than that of G. Furthermore, G_1 contains no subgraph F homeomorphic with either K_5 or $K_{3,3}$. Suppose instead that it did; then such a subgraph must contain the edge uw or G itself would have contained a subgraph homeomorphic with K_5 or $K_{3,3}$, which is not possible. But then $F - uw + vu + vw$ is a subgraph of G that is homeomorphic from F and, hence, implies that G contains a subgraph homeomorphic with either K_5 or $K_{3,3}$, again a contradiction.

In either case, we arrive at a contradiction, and, hence, $\delta(G) \geq 3$ as claimed.

Now, by exercise 32 in Chapter 2, we see that G is not a minimal block. Thus, there exists an edge $e = uv$ such that $H = G - e$ is still a block. Since H also has no subgraphs homeomorphic with K_5 or $K_{3,3}$ and has fewer edges than G, we see that H must be planar. Since H is a block other than K_2, Theorem 2.2.4 implies that H possesses cycles containing both u and v. Thus, suppose that H is a plane graph with a cycle C containing u and v and that C has been chosen with a maximum number of interior regions, say

$$C: \ u = v_0, v_1, \ldots, v_i = v, \ldots, v_n = u, \text{where } 1 < i < n - 1.$$

For convenience, let the *interior subgraph* (*exterior subgraph*) of H be the subgraph of G induced by those edges lying in the interior (exterior) of C. With this in mind, we note the following observations about H.

Observation 1. Since G is nonplanar, both the interior and exterior subgraphs are nonempty; otherwise, the edge e could be inserted into H in the appropriate region to obtain a plane embedding of G.

Observation 2. No two vertices of the set $\{ v_0, v_1, \ldots, v_i \}$ are connected by a path in the exterior subgraph of H. If they were, this would contradict our choice of C as having the maximum number of interior regions. A similar statement applies to the set $\{ v_i, v_{i+1}, \ldots, v_n \}$. The observations about these two sets, along with the fact that $H + e$ is nonplanar, imply the existence of a $v_j - v_k$ path P in the exterior subgraph of H, where $0 < j < i < k < n$. Further, note that no vertex of P different from v_j and v_k is adjacent to a vertex of C other than v_j or v_k. If it were, we would again contradict our choice of C as having the maximum number of interior regions. Furthermore, any path joining a vertex of P with a vertex of C must contain at least one of v_j or v_k.

Let H_1 be the component of $H - \{ v_l \mid 0 \le l < n, l \ne j, k \}$ containing P. By our choice of C, we know that H_1 cannot be inserted in the interior of C in a plane manner. This fact, along with the assumption that G is nonplanar, implies that the interior of H must contain one of the following:

a. A $v_r - v_s$ path P_1, $0 < r < j$, $i < s < k$ or equivalently $j < r < i$ and $k < s < n$, none of whose vertices different from v_r and v_s belongs to C. In this case, we see that the graph contains a subgraph homeomorphic with $K_{3,3}$ with partite sets $\{ v_j, v_s, v_i \}$ and $\{ v_k, v_0, v_r \}$ (see Figure 6.2.2).

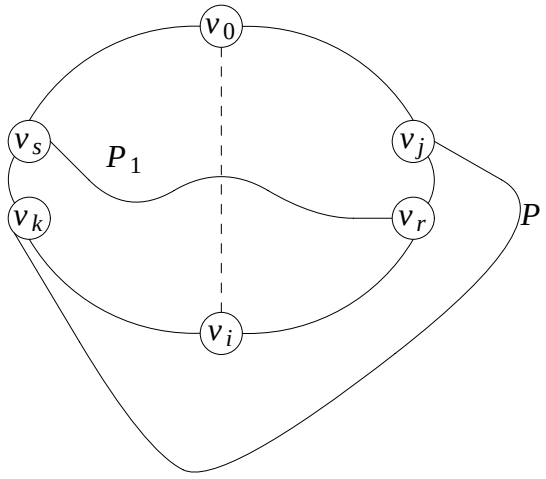


Figure 6.2.2. The case a.

b. A vertex $x \notin V(C)$ that is connected to C by three internally disjoint paths such that the end vertex of one such path P_1 is one of v_0, v_i, v_j and v_k. If P_1 ends at v_0, the end vertices of the other paths are v_r and v_s where $j \le r < i$ and $i < s \le k$, but not both $r = j$ and $s = k$, hold. If P_1 ends at any of v_i, v_j or v_k, we obtain three analogous cases. In this case we again find a subgraph

homeomorphic with $K_{3,3}$ with partite sets $\{ x, v_k, v_i \}$ and $\{ v_0, v_s, v_r \}$ (see Figure 6.2.3).

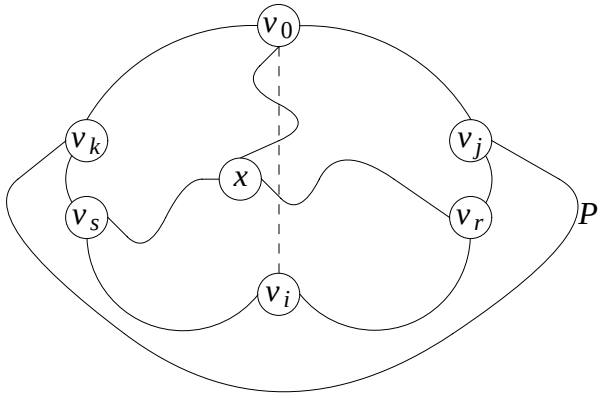


Figure 6.2.3. The case b.

c. A vertex $x \notin V(C)$ that is connected to C by three internally disjoint paths P_1, P_2 and P_3 where the end vertices of the paths other than x are three of v_0, v_i, v_j and v_k, say v_0, v_i and v_j, respectively, together with a $v_t - v_k$ path P_4 ($v_t \neq v_0, v_i, x$) where v_t is on P_1 or P_2, and P_4 is disjoint from P_1 and P_2, and C except for v_t and v_k. The remaining choices for P_1, P_2 and P_3 produce three analogous cases. In this case, we again find a subgraph homeomorphic with $K_{3,3}$ using the sets $\{v_0, v_t, v_j\}$ and $\{x, v_k, v_i\}$ (see Figure 6.2.4).

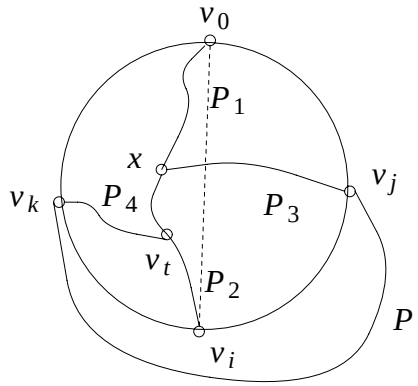


Figure 6.2.4. The case c.

d. A vertex $x \notin V(C)$ that is connected to vertices v_0, v_i, v_j and v_k by four internally disjoint paths. In this case a subgraph homeomorphic with K_5 is produced (see Figure 6.2.5).

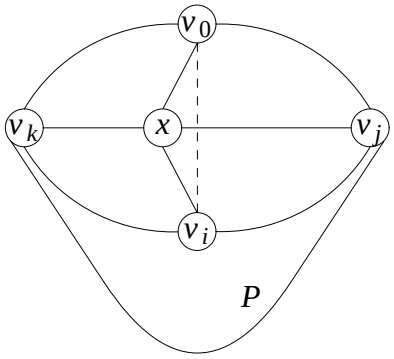


Figure 6.2.5. The case d.

These cases exhaust the possibilities, and each produces a subgraph homeomorphic with either K_5 or $K_{3,3}$. But our assumptions are contradicted; hence, no such graph G can exist and the proof is completed. □

Next, we consider a more constructive look at planar graphs centering on the cycles of the graph. If $G_1 = (V_1, E_1)$ is a subgraph of $G = (V, E)$, then a *piece* of G relative to G_1 is either an edge $e = uv$ where $e \notin E_1$ and $u, v \in V(G_1)$ or a connected component of $(G - G_1)$ plus any edges incident to vertices of this component (see Figures 6.2.6 and 6.2.7).

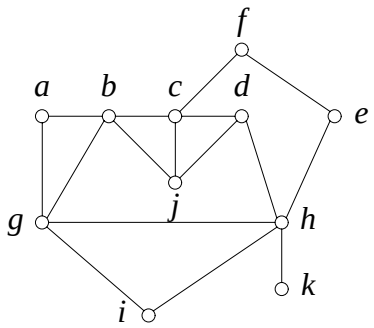


Figure 6.2.6. A plane graph G with initial cycle C: b, c, d, h, i, g, b.

For any piece P relative to G_1, the vertices of P in G_1 are called the *contact vertices of P*. If a piece has two or more contact vertices, it is called a *segment.* If C is a cycle of G, then C (when embedded) divides the plane into two regions, one interior to C, the other exterior. Two segments are *incompatible* if when placed in the same region of the plane determined by C, at least two of their edges cross. An example of this is shown in Figure 6.2.8.

Note that since pieces that are not segments have only one contact vertex, their embeddings are relatively easy. In order to manage the many potential segments of a graph, we introduce another graph whose structure depends on the cycle C. This new

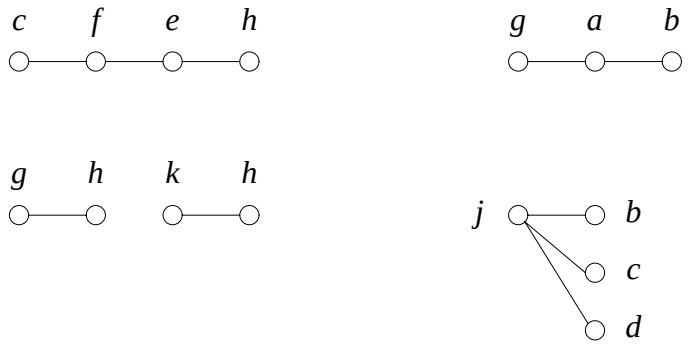


Figure 6.2.7. Pieces and segments of the plane graph G relative to C.

graph is called an *auxiliary graph* and is denoted $P(C)$. The graph $P(C)$ is constructed as follows: $P(C)$ has a vertex corresponding to each segment of the graph G relative to C, and an edge joins two such vertices if, and only if, the corresponding segments are incompatible. In attempting to embed the segments of G relative to C, we clearly must have incompatible segments in different regions. The compatible segments are independent vertices in $P(C)$. If $P(C)$ were actually a bipartite graph, then the segments represented by the vertices in one partite set could be embedded in the same region (either inside C or outside C) without conflict. Further, the segments represented by the vertices of the other set could be embedded in the second region without conflict and without affecting the segments in the first region. That this is both a necessary and sufficient condition is shown in our next result (Demoucron, Malgrange and Pertuiset [2]).

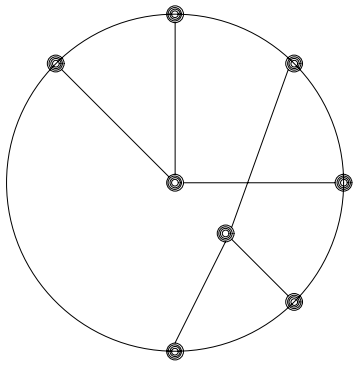


Figure 6.2.8. Incompatible segments.

Theorem 6.2.2 The graph G is planar if, and only if, for every cycle C of G, the auxiliary graph $P(C)$ is bipartite.

Proof. Suppose G is not planar. Then by Kuratowski's theorem, G contains a subgraph homeomorphic with either K_5 or $K_{3,3}$. Suppose G contains either K_5 or $K_{3,3}$ (the generalization to graphs homeomorphic with these is similar). In either case, we can select a cycle C such that $P(C)$ is not bipartite. For K_5 the segments are two edges and a vertex and its incident edges (see Figure 6.2.9 for example), while in $K_{3,3}$, the segments are all single edges (Figure 6.2.10). In either situation, any two of the three segments are incompatible. Thus, $P(C)$ is not bipartite.

Conversely, if G is planar, the fact that $P(C)$ is bipartite follows from our earlier discussion. □

Example 6.2.1. Consider K_5 with $V(K_5) = \{1, 2, 3, 4, 5\}$ and let C be the cycle 1, 2, 3, 4, 1. Then C and the segments of $K_5 - C$ are shown in Figure 6.2.9. It is easy to see that any two segments are incompatible; thus, $P(C) = K_3$, which is clearly not bipartite. Thus, we have another proof that K_5 is nonplanar. A similar argument applies to $K_{3,3}$ (see Figure 6.2.10). □

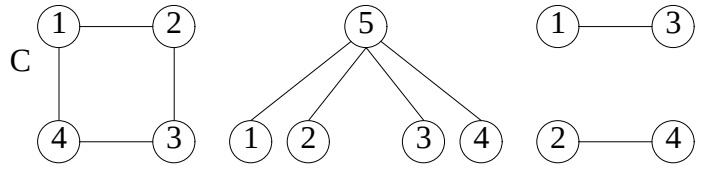


Figure 6.2.9. The cycle C and its relative segments in K_5.

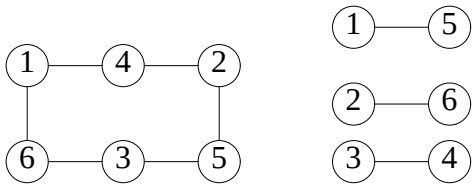


Figure 6.2.10. The cycle C and its relative segments in $K_{3,3}$.

Another characterization of planar graphs is from Whitney [6]. This characterization is of a more geometric nature. Given a plane graph G, we construct a pseudograph G^* as follows: For each region of G, we associate a vertex of G^*. Two vertices of G^* are joined by an edge corresponding to each edge that belongs to the boundary of both corresponding regions of G. Further, a loop is added at any vertex of G^* for each bridge of G that belongs to the boundary of the corresponding region. Note that each edge of G^* can be drawn in such a way that it crosses the associated edge of G (but no other edge of G). The pseudograph G^* is called the (*geometric*) *dual of* G (see Figure 6.2.11). It is clear from our discussion that G^* is planar; however, it may not be unique. Rather, it depends on the embedding selected for G.

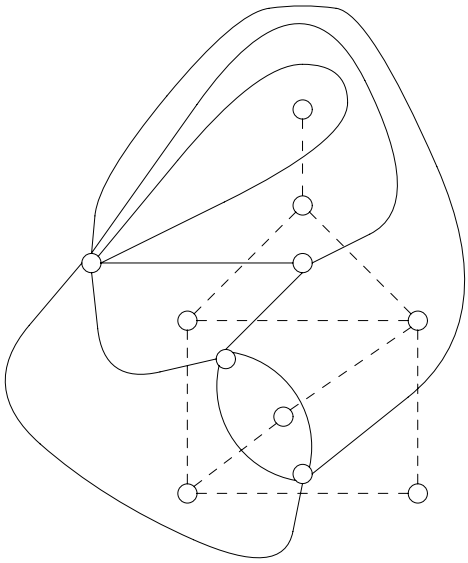


Figure 6.2.11. A graph (dashed lines) and its geometric dual (solid lines).

We require a slightly stronger idea in order to make use of duals in characterizing planar graphs. This idea is combinatorial in nature. Let G and $\tilde{G}$ be two graphs with a $1-1$ correspondence between their edges and let C be any cycle in G. The graph $\tilde{G}$ is called the *combinatorial dual of* G if, and only if, the set of edges $\tilde{C}$ in $\tilde{G}$ corresponding to the edges of C in G form an edge cut set. It is important to note that this definition makes no reference to planar graphs or the possible embedding. We now wish to investigate the relationship between the two duals we have introduced.

Theorem 6.2.3 Every planar graph G has a planar combinatorial dual.

Proof. We already know that every planar graph has a planar geometric dual. Hence, given an embedding of G, construct the geometric dual G^* as before. Now, given any cycle of G, this cycle must divide the plane into two regions. Thus, we can partition the vertices of G^* into two nonempty subsets A and B, according to their locations inside C or outside C. Removal of the set of edges C^* which correspond in G^* to the edges of C clearly separates A and B. Thus, C^* is an edge cut set of G^*.

Given any edge cut set C^* of G^*, we would like to determine the corresponding set of edges C in G and show that C forms a cycle in G. Suppose this is not the case. By our construction, only one vertex of G^* lies in any region determined by the plane embedding of G. Consider the edges incident to some vertex v of G. Each of these edges lies on the boundary of two regions of G in this embedding, and each of these regions contains one vertex of G^*. (Of course, if the edge is a bridge, the two regions are equal.) The end vertices of the corresponding edges of G^* are then determined, and these edges bound a region of G^*. Hence, every vertex of G lies in one region of G^*. If C is not a

cycle of G, then there are at least two edges of C whose end vertices are not adjacent to other vertices of C. If C^* was a cut set, then these end vertices would have to lie in the same region of G^*. But this is a contradiction, since each region contains only one vertex of G. □

What we have shown is that if G_1 is planar, then the combinatorial dual actually coincides with the geometric dual. Our construction makes the following corollary immediate.

Corollary 6.2.1 If G is a graph with geometric dual G^*, then $(G^*)^* = G$.

We can now show that the existence of a combinatorial dual characterizes planar graphs.

Theorem 6.2.4 A graph G is planar if, and only if, G has a combinatorial dual.

Proof. From Theorem 6.2.3, we know that every planar graph has a combinatorial dual. Thus, we need only show that every nonplanar graph has no combinatorial dual.

From our definition of combinatorial dual, we see that G has a combinatorial dual if, and only if, each of its subgraphs has a combinatorial dual. Also, if a graph has a combinatorial dual, then so does any graph homeomorphic from it. Further, G is nonplanar if, and only if, it contains a subgraph homeomorphic with K_5 or $K_{3,3}$. Thus, it suffices to show that these graphs have no combinatorial dual.

To see that K_5 has no combinatorial dual, suppose instead that it did, say $\tilde{K}_5$. Since K_5 has ten edges and is an ordinary graph with edge connectivity 4, we observe that there are no 2−cycles and no edge cut sets with two or three edges. In fact, K_5 does have edge cut sets with four or six edges. These properties imply that $\tilde{K}_5$ has ten edges, no vertices of degree less than 3 and no 2-cycles, but it does contain 4− and 6−cycles. However, these properties are mutually incompatible.

To see that $K_{3,3}$ has no combinatorial dual, we again suppose that it did have such a dual, say $\tilde{K}_{3,3}$. Clearly, $\tilde{K}_{3,3}$ has no 2-cycles and no odd cycles, but it certainly contains 4− and 6−cycles. Thus, $\tilde{K}_{3,3}$ has no edge cut set with fewer than four edges. Also, the degree of every vertex of $\tilde{K}_{3,3}$ is at least 4. Thus, there are at least twelve edges in $\tilde{K}_{3,3}$. But there are only nine edges in $K_{3,3}$, and, hence, by our construction there can be only nine edges in $\tilde{K}_{3,3}$, producing the desired contradiction. □

Section 6.3 A Planarity Algorithm

In the next two sections, we will discuss two well-known planarity algorithms. The first, from Demoucron, Malgrange, and Pertuiset [2], is somewhat easier to understand and uses the ideas of segments. In the next section, we will discuss a linear planarity algorithm.

In any attempt at an algorithm to determine if a graph is planar, there are several preliminary tests that can really help simplify the process. These include:

1. If $|E| > 3p - 6$, then the graph must be nonplanar.
2. If the graph is disconnected, consider each component separately.
3. If the graph contains a cut vertex, then it is clearly planar if, and only if, each of its blocks is planar. Thus, we can limit our attention to 2 –connected graphs.
4. Loops and multiple edges change nothing; hence, we need only consider graphs.
5. A vertex of degree 2 can certainly be replaced by an edge joining its neighbors. This *contraction* of all vertices of degree 2 constructs a homeomorphic graph with the smallest number of vertices. Certainly, a graph is planar if, and only if, the contraction is planar.

Thus, if we do some preprocessing, our task will sometimes be greatly reduced. Certainly tests 4 and 5 have the greatest effect on the graph under consideration, while test 1 can produce the most rapid benefit.

The key concept in the Demoucron, Malgrange and Pertuiset algorithm is the following idea: Let $\hat{H}$ be a plane embedding of the subgraph H of G. If there exists a planar embedding of G (say $\hat{G}$) such that $\hat{H} \subseteq \hat{G}$, then $\hat{H}$ is said to be *G-admissible.* As an example, consider the graph $G = K_5 - e_1$ where $V = \{1, 2, 3, 4, 5\}$ and the missing edge e_1 is from vertex 1 to vertex 5. The graphs of Figure 6.3.1 show a G-admissible and a G-inadmissible embedding of the subgraph $H = G - e_2$ where e_2 is the edge from vertex 1 to vertex 2.

Let S be any segment of G relative to a subgraph H. Then S can be drawn in a region r of $\hat{H}$, provided all the contact vertices of S lie in the boundary of r. This allows us to extend the embedding of $\hat{H}$ to include at least part of S. The strategy of the algorithm is to find a sequence of subgraphs $\hat{H}_1, \hat{H}_2, \ldots, \hat{H}_{|E|-p+2} = G$ (do you know why there are this many potential subgraphs?) such that $H_i \subset H_{i+1}$ and such that $\hat{H}_i$ is G-admissible (if possible). In this way, a planar embedding of G (if one exists) can be constructed, or we will discover some segment S which cannot be compatibly embedded in any region.

The algorithm begins by finding any cycle in G and embedding it. This cycle is the first subgraph, $\hat{H}_1$. The algorithm then attempts to find the set of segments relative to the

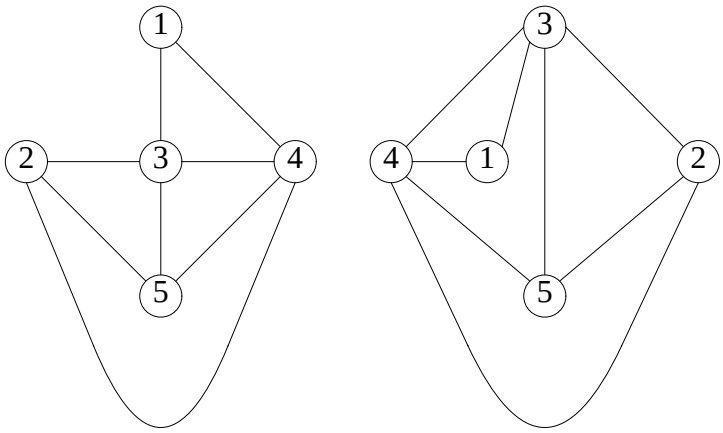


Figure 6.3.1. An admissible and inadmissible embedding of H.

current $\hat{H}_i$. For each such segment S, the set $R(S, \hat{H}_i)$ is found, where $R(S, \hat{H}_i)$ is the set of regions in which S can be compatibly embedded in $\hat{H}_i$. If there exists a segment S which has only one such region r, then $\hat{H}_{i+1}$ is constructed by drawing a path P between two of the contact vertices of S in the region r. If no such segment exists, then a path P between two contact vertices of any segment is constructed. In either case, the path P divides the region in which it is embedded into two regions. The process is then repeated if necessary. We now state the Demoucron, Malgrange and Pertuiset (DMP) algorithm [2].

Algorithm 6.3.1 DMP Planarity Algorithm.
Input: A preprocessed block (after applying tests $1-5$).
Output: The fact that the graph is planar or nonplanar.
Method: Look for a sequence of admissible embeddings beginning with some cycle C.

1. Find a cycle C and a planar embedding of C as the first subgraph $\hat{H}_1$. Set $i \leftarrow 1$ and $r \leftarrow 2$.
2. If $r = |E| - p + 2$,
   then stop;
   else determine all segments S of $\hat{H}_i$ in G, and for each segment S determine $R(S, \hat{H}_i)$.
3. If there exists a segment S with $R(S, \hat{H}_i) = \varnothing$,
   then stop and say G is nonplanar;
   else if there exists a segment S such that $|R(S, \hat{H}_i)| = 1$,
   then let $\{ R \} = R(S, \hat{H}_i)$;
   else let S be any segment and R any region in $R(S, \hat{H}_i)$.

4. Choose a path P in S connecting two contact vertices. Set $H_{i+1} = H_i \cup P$ to obtain the planar embedding $\hat{H}_{i+1}$ with P placed in R.
5. Set $i \leftarrow i + 1$, $r \leftarrow r + 1$ and go to step 2.

Example 6.3.1. We now determine an embedding of $G = K_5 - \{ e_1, e_2 \}$, using Algorithm 6.3.1, where $V = \{ 1, 2, 3, 4, 5 \}$ and where e_1 is the edge from vertex 4 to vertex 5 and e_2 is the edge from vertex 2 to vertex 4.

In step 1 we determine any cycle $C = H_1$ of G, for example; the cycle 1, 2, 3, 4, 1, and we embed it in the plane as shown in Figure 6.3.2 as $\hat{H}_1$. We also set $i = 1$ and $r = 2$.

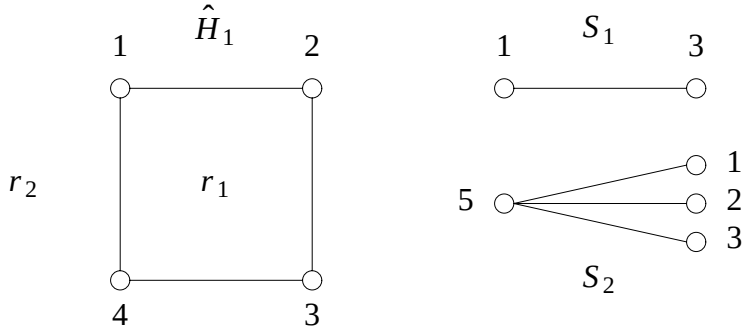


Figure 6.3.2. The first graph, $\hat{H}_1$, and its segments.

In step 2 we determine the segments relative to $\hat{H}_1$. These consist of the edge from 1 to 3 and the vertex 5 and its incident edges 1 to 5, 2 to 5 and 3 to 5. Call these segments S_1 and S_2, respectively. Then

$$R(S_1, \hat{H}_1) = \{ r_1, r_2 \} = R(S_2, \hat{H}_1).$$

Since both sets contain more than one region, we can select either segment for the embedding. Suppose we select S_2. Next, we must choose a path between two contact vertices, say this path P is 1, 5, 2, and we embed it in any region in $R(S_2, \hat{H}_1)$. Without loss of generality, suppose it is embedded in the exterior region r_2 of $\hat{H}_1$. We obtain the graph $\hat{H}_2$ shown in Figure 6.3.3. This clearly creates a third region r_3, and the regions are now as shown in Figure 6.3.3. Next, update $r \leftarrow 3$ and $i \leftarrow 2$ and go to step 2. Again, select the segments relative to $\hat{H}_2$: the edge from 1 to 3 and the edge from 3 to 5. In step 3, since the edge from 3 to 5 can only be placed in r_2, it is the segment selected and embedded as shown in Figure 6.3.4, forming $\hat{H}_3$.

Finally, the only remaining segment is the edge from 1 to 3, and it can be drawn in region r_1 or r_2. The choice of region determines the final embedding. We select region r_1, and in this case we obtain the embedding $\hat{H}_4$, shown in Figure 6.3.5.

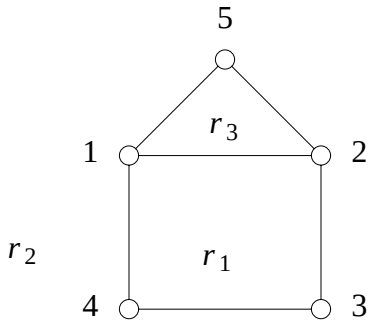


Figure 6.3.3. The graph $\hat{H}_2$ formed by inserting P in the exterior region.

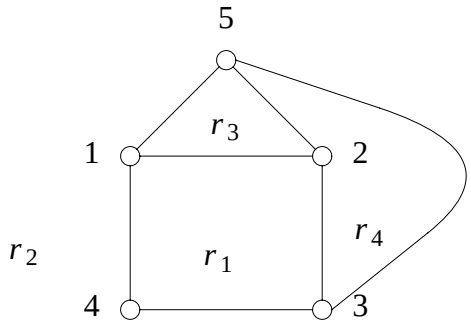


Figure 6.3.4. The graph $\hat{H}_3$ formed by inserting the edge from 3 to 5 in r_2.

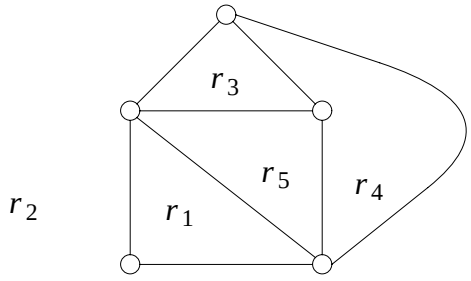


Figure 6.3.5. The graph $\hat{H}_4 = K_5 - \{ e_1, e_2 \}$.

Section 6.4 The Hopcroft-Tarjan Planarity Algorithm

In this section we examine the most effective planarity algorithm (Hopcroft and Tarjan [4]). This algorithm has complexity $O(|V|)$ and is therefore a very effective method of testing for planarity. We will not be able to prove all aspects of the algorithm here, but rather we will try to establish enough for the reader to have a general understanding of the algorithm without necessarily having enough details to immediately begin programming it. For more details on the data structures necessary to program the algorithm, see [4]. To study the algorithm, we will break it into a number of separate sections and try to describe each of these sections individually.

The fundamental idea of the Hopcroft-Tarjan planarity algorithm is the typical one for planarity testing: Break the graph into a cycle and paths, and then fit these paths together in such a way as to find the embedding or the fact that the graph is not planar. We seek a cycle C and a collection of paths P_i so that when we embed C in the plane, the paths P_i can be embedded either inside C or outside C. If this can be done without interference, then G is planar. The difference in this algorithm is the strategy for finding the paths. In order to find the initial cycle and the subsequent paths, we will first perform a depth-first search on G and then, using the information obtained from this search, impose an ordering of the edges in each of the adjacency lists for the vertices of G. This will allow us to select paths with special properties that will facilitate the testing for planarity.

We begin by reordering the vertices and their adjacency lists. This reordering will make it possible for us to actually produce the paths easily and in such a way that paths from the same segment are generated consecutively. To do this, we perform a depth-first search on G, viewing the result as a digraph D_G, with its tree arcs and back arcs. From this point on we identify vertices by their number value $num(v)$, assigned by the DFS. We also assign each vertex a lowpoint value $lowpt(v)$, defined to be the number of the lowest vertex reachable from v or any of its descendents in the DFS tree using at most one back edge. When it is not possible to do this, then $lowpt(v) = v$. In a similar fashion, let $nextlopt(v)$ be the next lowest vertex below v, excluding $lowpt(v)$ reachable in the same manner. If there is no such vertex, then set $nextlopt(v) = v$. More formally, then, let S_v be the set of all vertices reachable from v using zero or more tree arcs and at most one back arc. Then we define

$$lowpt(v) = \min (S_v)$$

and

$$nextlopt(v) = \min (\{v\} \cup (S_v - lowpt(v)).$$

We now restate a result fundamental to the use of these functions and previous examined in Lemmas 2.2.3 and 2.2.4. Recall that it also supplies us with a means of finding cut vertices.

Theorem 6.4.1 If $G = (V, E)$ is a connected graph with DFS search tree T and back edges B, then $u \in V$ is a cut vertex if, and only if, there exist vertices $v, w \in V$ such that $u \to v \in T$ and w is not a descendent of v in T and $lowpt(v) \geq num(u)$.

The last result implies that $num(v) \geq nextlopt(v) > lowpt(v)$, except when v is the root of the DFS tree T, in which case

$$lowpt(v) = nextlopt(v) = num(v) = 1.$$

Next, we reorder the adjacency lists of D_G so that during a DFS on D_G, the paths we desire will be generated in a useful order. In order to accomplish this goal, we need another function defined on $E(G)$. Let $e = v \to w \in E(D_G)$ and define

$$\phi(e) = \begin{cases} 2\ w & \text{if } e \text{ is a back edge} \\ 2\ lowpt(w) & \text{if } e \text{ is a tree edge and } nextlopt(w) \geq v \\ 2\ lowpt(w) + 1 & \text{if } e \text{ is a tree edge and } nextlopt(w) < v. \end{cases}$$

Now, we sort all arcs $v \to w$ incident to v into nondecreasing order based on their ϕ values and use this ordering in the adjacency list of each vertex. These new adjacency lists are called $adj(v)$ for each vertex v. The important point to notice here is that in $adj(v)$, a back edge going to a vertex of lower DFS number always precedes a back edge going to a vertex of higher DFS number and tree edges generally appear in nondecreasing order of their abilities to lead to a vertex below v using a single back edge. To illustrate what we have done so far, consider the graph of Figure 6.4.1.

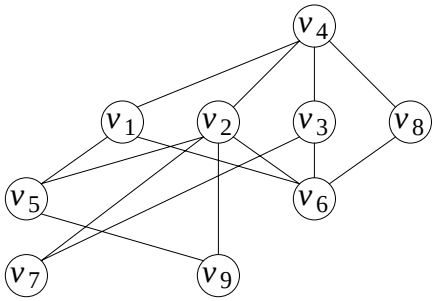


Figure 6.4.1. A graph G to test for planarity.

The digraph of Figure 6.4.2 shows D_G obtained from a DFS beginning at v_1. The search tree is shown with dashed arcs, and the back arcs are solid. The numbering alongside the vertices indicates the DFS number. Table 6.4.1 lists each vertex, its DFS number, *lowpt* value and *nextlopt* value.

Table 6.4.2 lists the arcs and their ϕ values. Now we can order the adjacency lists. This is shown in Table 6.4.3. Now that we have properly prepared the graph and its adjacency

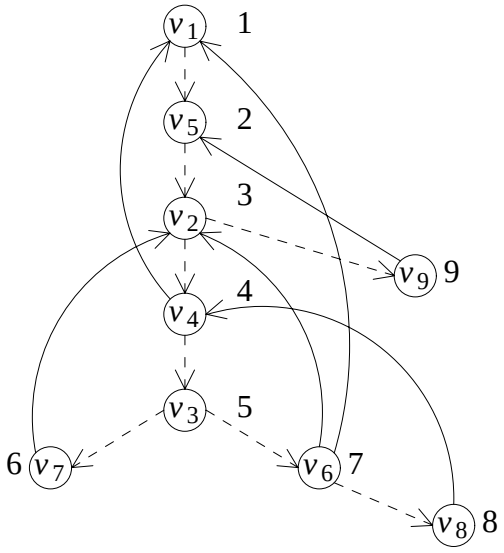


Figure 6.4.2. The DFS tree in dashed edges; back edges are solid.

vertex v	$num(v)$	$lowpt(v)$	$nextlopt(v)$
v_1	1	1	2
v_5	2	1	2
v_2	3	1	2
v_4	4	1	3
v_3	5	1	3
v_7	6	3	6
v_6	7	1	3
v_8	8	4	8
v_9	9	2	9

Table 6.4.1 Table of DFS numbers and function values.

lists, we are ready to use the lists to help us select both the first cycle and the subsequent paths. This decomposition, performed on D_G, will provide the pieces necessary to test for planarity. So, starting at vertex 1 (from now on we identify the vertex with its DFS number), follow tree edges until we reach a vertex z with the property that the first adjacency in $adj(z)$ is a back edge. We shall see that this back edge must go to vertex 1, thus forming the cycle we seek. Call the cycle C. In our example, $z = 4$, and the cycle $P_0 = C$ is $1, 2, 3, 4, 1$.

Now, starting at $z = 4$, follow the next adjacency indicated by $adj(z)$. Continue following tree edges until the first adjacency on a list forces us to traverse a back edge. This completes the first path of the decomposition. In our example this is the path

arc	ϕ	arc	ϕ	arc	ϕ
$v_1 \rightarrow v_5$	2	$v_5 \rightarrow v_2$	2	$v_2 \rightarrow v_4$	2
$v_3 \rightarrow v_7$	6	$v_3 \rightarrow v_6$	3	$v_6 \rightarrow v_8$	8
$v_7 \rightarrow v_2$	6	$v_8 \rightarrow v_4$	8	$v_6 \rightarrow v_2$	6
$v_2 \rightarrow v_9$	4	$v_9 \rightarrow v_5$	4	$v_4 \rightarrow v_3$	3
$v_4 \rightarrow v_1$	2	$v_6 \rightarrow v_1$	2		

Table 6.4.2 Table of arcs and ϕ values.

vertex v	$adj(v)$	vertex v	$adj(v)$
v_1	v_5	v_2	v_4 v_9
v_3	v_6 v_7	v_4	v_1 v_3
v_5	v_2	v_6	v_1 v_2 v_8
v_7	v_2	v_8	v_4
v_9	v_5		

Table 6.4.3 The reordered adjacency lists.

P_1: 4, 5, 7, 1.

A new path is now started at the initial vertex of the back arc that ended the previous path. If this vertex has no unused arcs, we back up to the predecessor of this vertex on the present path. This process is continued until we have used all the arcs in D_G. In our example we find P_2: 7, 3, then P_3: 7, 8, 4, followed by P_4: 5, 6, 3 and P_5: 3, 9, 2. We now state the path decomposition algorithm.

Algorithm 6.4.1 Path Decomposition.
Input: The DFS numbered digraph D_G along with tree and back edges and ordered adjacencies.
Output: The path decomposition $P_0 = C, P_1, \ldots, P_k$.
Method: Use the ordered adjacencies from a base vertex s to find the paths.

1. Set $i \leftarrow 0$, $P_i \leftarrow \phi$.
2. Call PATH(1).

Procedure PATH(v)

1. For each $w \in adj(v)$ do the following:
2. Set $P_i \leftarrow P_i \cup \{ v \rightarrow w \}$.

3. If $v < w$,
   then PATH(w);
   else $i \leftarrow i + 1$ and $P_i \leftarrow \phi$.

The path decomposition obtained using this algorithm has many special properties ideal for use in the planarity test. We state some of these properties below; their proofs are left to the exercises.

1. The number of paths in the decomposition is $|E| - |V|$.
2. Every path P_i has only its end vertices in common with the union of the previously generated paths.
3. Algorithm 6.4.1 always selects the unused back edge to the lowest numbered vertex.
4. Let P_i and P_j be two paths with initial and terminal vertices s_i, t_i and s_j, t_j, respectively. If s_i is an ancestor of s_j (or possibly equal), then $t_i \leq t_j$.
5. Let P_i and P_j be two paths with the same initial and terminal vertices s and t. Let v_i be the second vertex on P_i and v_j be the second vertex on P_j. If $s \rightarrow x_i$ is not a back edge and $nextlopt(x_i) < s$, then $s \rightarrow x_j$ is not a back edge and $nextlopt(x_j) < s$.

It is the final property that forces us to use the *nextlopt* function to break ties between two tree edges that start from the same vertex and have equal *lowpt* values for their terminal vertices.

This algorithm also uses the idea of a segment, but in a special way. Here, a *segment* of D_G with respect to C is either a single back edge with both vertices on C or the subgraph consisting of a tree edge $v \rightarrow w$ with $v \in C$ and $w \notin C$ and the directed subtree in T rooted at w along with all the back edges from this subtree. The vertex v of C at which a segment originates is called a *base vertex*. Algorithm 6.4.1 generates segments in decreasing order of their base vertices. All paths belonging to the same segment must be embedded together in the same region with respect to C. This is the motivation for trying to group these paths together in the generation process.

In our example graph, we find two segments. The first is formed from the paths P_1, P_2, P_3 and P_4, while the second is just the path P_5.

Now, we want to embed the segments. Initially, C divides the plane into two regions, one inside C and the other outside. We say the segment S is embedded *inside* C if the order of the edges encountered in a clockwise sweep around the base vertex v_i is $v_{i-1} \rightarrow v_i$, $v_i \rightarrow w$, $v_i \rightarrow v_{i+1}$. We say the segment is embedded *outside* C if the order is $v_{i-1} \rightarrow v_i, v_i \rightarrow v_{i+1}, v_i \rightarrow w$.

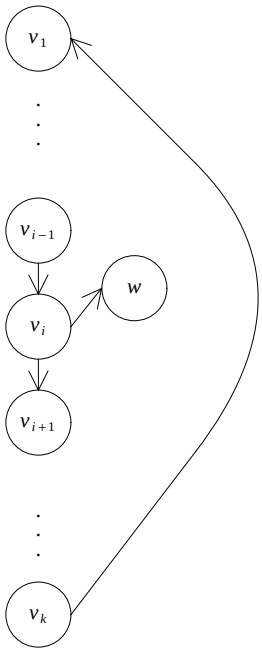


Figure 6.4.3. An embedding inside C.

After embedding C, we wish to embed the segments one by one in the order in which they are generated by Algorithm 6.4.1. In order to embed a segment, we consider the paths that compose the segment, and embed them one by one. By examining previously embedded paths, we can determine if the next path can be embedded in the region. If it cannot, then all the previously embedded segments that are blocking our embedding are moved to the other region. (Moving a segment to the other region may force the movement of even more segments.) If we still cannot embed the path after this rearrangement of segments, then we conclude that G is not planar. If, however, the path can be embedded, then we do so and continue to try to embed the rest of the segment, applying the embedding algorithm recursively. If we are successful in embedding this segment, we move on to the next segment.

The key to the embedding is the ability to easily test whether the first path of a segment can be embedded on a specific side of C. This can be done with the use of the following result.

Theorem 6.4.2 Let P be the first path in the current segment S and let P have base vertex v_i and terminal vertex v_j. If all segments generated prior to S have already been embedded, then P can be embedded in a region of C if there is no previously embedded back edge $x \rightarrow v_t$ in this region also entering C between vertices v_i and v_j. That is, no back edge $x \rightarrow v_t$ satisfies $v_j < v_t < v_i$. Furthermore, if there is such a back edge,

then *S* cannot be embedded in this region of *C*.

This result makes it clear that the initial and terminal vertices of a path are all we really need to know when testing whether a given path can be embedded. (This is really the idea we mentioned earlier in connection with Kuratowski's theorem: An edge and a path are just as difficult to embed.) This easy criterion enhances the use of this algorithm.

So far we have talked about embedding the first path of the segment *S*. To determine what to do with the rest of the segment, we note that it can be embedded properly if, and only if, $S \cup C$ is planar. To determine the planarity of $S \cup C$, we apply the embeddability criterion recursively, until all paths in *S* are embedded in the plane or until we learn that we cannot embed *G*.

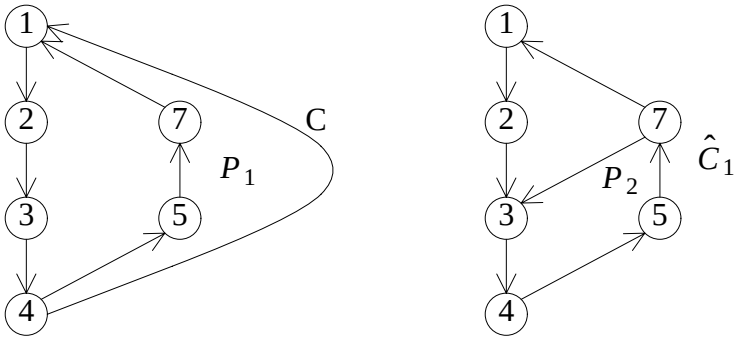


Figure 6.4.4. The first two paths embedded inside their respective cycles.

To perform the recursion on successive paths, we proceed as follows. If the path *P* has just been embedded, then *P* and the tree edges from the final vertex of *P* to the initial vertex of *P* form the new cycle $\hat{C}$. Removing $\hat{C}$ from $\hat{G} = S \cup C$ may further partition the remaining digraph into segments. These segments are again handled recursively. The first few cycles and embeddings of our example graph are easy (see Figures 6.4.4 and 6.4.5).

However, in attempting to embed P_4 inside $\hat{C}_3$, we find a conflict. Further, a conflict also exists outside of $\hat{C}_3$. But we cannot yet conclude that the graph is nonplanar. Working recursively, we now move the last segment embedded (one level earlier), outside of its cycle. Thus, P_3 is moved outside $\hat{C}_2$. The path P_4 still fails inside $\hat{C}_3$, but it can be properly embedded outside $\hat{C}_3$, as shown in Figure 6.4.6. The final segment is also easily embedded.

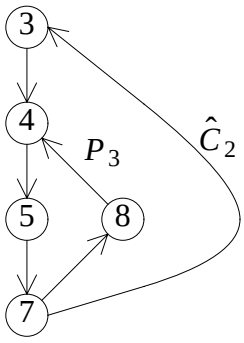


Figure 6.4.5. The third path is embedded inside $\hat{C}_2$.

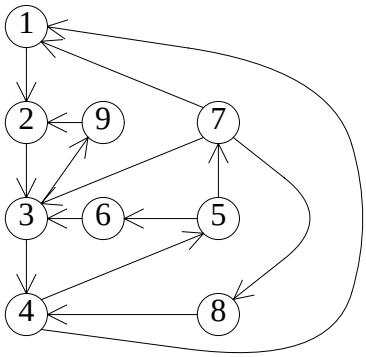


Figure 6.4.6. The final embedding.

We now present the Hopcroft-Tarjan algorithm.

Algorithm 6.4.2 The Hopcroft-Tarjan Planarity Algorithm.
Input: A 2–connected graph with $|E| \leq 3|V| - 6$.
Output: A determination of the planarity of the graph.
Method: Recursive testing of the paths from the path decomposition. They are embedded inside or outside the initial cycle and moved if necessary.

Procedure Planar(G)

1. Perform a DFS on G to obtain D_G.
2. Find a cycle C in D_G.
3. Construct a planar representation for C.
4. For each segment S formed in $G - C$, do the following:
   Apply the algorithm recursively to determine if $S \cup C$ is planar. That is, call

Planar($S \cup C$).
If $S \cup C$ is planar and S may be added to the planar representation already constructed, then add it; otherwise, halt and state that G is nonplanar.

To determine the complexity of the Hopcroft-Tarjan algorithm, note that the path-finding algorithm requires $O(|V| + |E|)$ operations. The embedding consists of the end vertices of the $|E| - |V|$ paths generated off the cycle. The embedding is done completely with stack operations (storing end vertices of paths), and pushing and popping entries takes constant time per entry. Since the total number of entries in any stack is $O(|V| + |E|)$, then we see that the entire algorithm takes only $O(|V| + |E|)$ operations. However, since we have restricted use of the algorithm to preprocessed graphs (recall preprocessing steps $1-5$ of the last section) satisfying the restriction that $|E| \leq 3|V| - 6$, we see that the algorithm has complexity $O(|V|)$.

Section 6.5 Hamiltonian Planar Graphs

In Chapter 5 we studied many sufficient conditions implying that a graph was hamiltonian. However, until now we have said very little about necessary conditions for hamiltonian graphs. In this section we will show that such conditions do exist and that when we include planarity as an added condition, a very nice theorem (Grinberg [3]) results.

In order to present this result, we need a bit more terminology. Let G be a hamiltonian plane graph of order p. Further, let C be a fixed hamiltonian cycle in G. We say that an edge e is a *diagonal* with respect to C if e is not on C. Let r_i denote the number of regions of G containing exactly i edges that lie interior to C and let $\hat{r}_i$ denote the number of regions containing exactly i edges lying exterior to C. Then, using this notation, we can present Grinberg's theorem [3].

Theorem 6.5.1 Let G be a plane graph of order p with a hamiltonian cycle C. Then, with respect to this cycle,

$$\sum_{i=3}^{p} (i-2)(r_i - \hat{r}_i) = 0.$$

Proof. Consider a diagonal on the interior of C. This diagonal divides the region in which it is embedded into two regions. Therefore, if there are m diagonals in the interior of C, then

$$\sum_{i=3}^{p} r_i = m + 1.$$

But this implies that

$$m = \left(\sum_{i=3}^{p} r_i \right) - 1.$$

Let N denote the sum over all $m + 1$ interior regions of the number of edges bounding these regions. Then

$$N = \sum_{i=3}^{p} i r_i.$$

However, N also counts each interior diagonal twice and each edge of C once. Thus,

$$N = 2m + p.$$

Substituting for N and m in our last equation, we obtain

$$\sum_{i=3}^{p} i r_i = 2 \sum_{i=3}^{p} r_i - 2 + p.$$

Thus,

$$\sum_{i=3}^{p} (i - 2) r_i = p - 2.$$

A similar argument applied to the exterior regions produces the fact that

$$\sum_{i=3}^{p} (i - 2) \hat{r}_i = p - 2.$$

But the last two equations together imply that

$$\sum_{i=3}^{p} (i - 2)(r_i - \hat{r}_i) = 0. \quad \square$$

Example 6.5.1. Suppose that we are given the plane graph of Figure 6.5.1 with each region being indicated as r. Then there are five regions; two are triangular and three are six-sided. We will use Grinberg's theorem to prove that this graph is not hamiltonian. Note that we could simply check all ten possibilities for the regions being interior or exterior; however, the task can be handled in an easier manner.

First, suppose that the two triangular regions were in opposite regions with respect to a hamiltonian cycle C. Then in Grinberg's formula, both $r_3 = 1$ and $\hat{r}_3 = 1$ so that this term would contribute zero to the sum. But, then the only other nonzero term involves the three six-sided regions, and there is no way to obtain a contribution of zero from this term. Thus, there can be no hamiltonian cycle in which one three-sided region is interior to C and the other is exterior to C.

Next, suppose that both three-sided regions are interior to C so that $r_3 = 2$. Then, this term contributes 2 to the sum in Grinberg's formula. The only way to negate this term is in the contribution of the six–sided regions. However, it is easy to see that no placement of the six–sided regions will result in a zero sum in Grinberg's formula. A similar argument applies when $\hat{r}_3 = 2$; thus, we conclude that the graph of Figure 6.5.1 is not hamiltonian. □

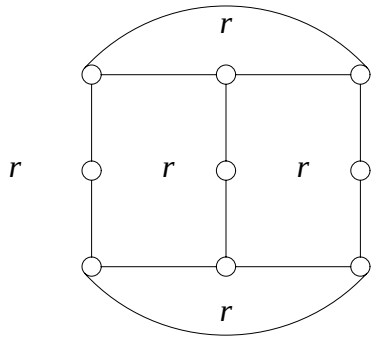


Figure 6.5.1. A plane nonhamiltonian graph.

Exercises

1. Show that if G is a plane (p, q) graph with r regions, then $p - q + r = 1 + k(G)$ where $k(G)$ is the number of components of G.
2. An $(f, d)-$ *regular polyhedron graph* is a plane graph that is d-regular $(d \geq 3)$ and each of its faces has f sides. Use Euler's formula to show that there are only five regular polyhedron graphs. (Hint: The dodecahedron is a $(5, 3)-$regular polyhedron graph.)
3. Show that "homeomorphic with" is an equivalence relation on the set of graphs.
4. Show that a graph is planar if, and only if, each of its blocks (maximal 2-connected subgraphs) is planar.
5. Let G be a maximal planar graph of order $p \geq 4$. Also let p_i denote the number of vertices of degree i in G, where $i = 3, 4, \ldots, \Delta(G) = n$. Show that
$$3p_3 + 2p_4 + p_5 = p_7 + 2p_8 + \cdots + (n - 6)p_n + 12.$$
6. Show that any maximal planar (p, q) graph contains a bipartite subgraph with $2\frac{q}{3}$ edges.
7. Find an example of a planar graph that contains no vertex of degree less than 5.
8. Prove that every planar graph of order $p \geq 4$ contains at least four vertices of degree at most 5.
9. Show that the Petersen graph (Figure 7.3.2) contains a subgraph homeomorphic with $K_{3,3}$ and is therefore not planar.
10. Show that if G is a connected planar (p, q) graph with girth (shortest cycle length) $g(G) = k \geq 3$, then $|E| \leq \frac{k(p - 2)}{(k - 2)}$.

11. Use the last result to again show that the Petersen graph is not planar.

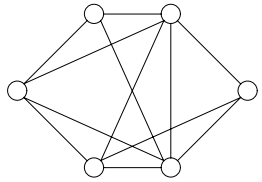
12. A graph is self-dual if it is isomorphic to its own geometric dual. Show that if G is self-dual, then $2|V| = |E| + 2$. Further, show that not every graph with this property is self-dual.

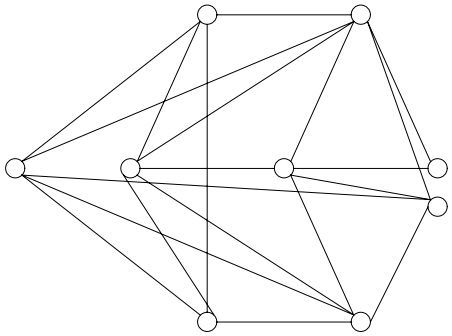
13. If G is a connected plane graph with spanning tree T and $E^* = \{ e^* \in E(G^*) \mid e \notin E(T) \}$, show that $T^* = \left(E^* \right)$ is a spanning tree of G^*.

14. Show that if $|V(G)| \geq 11$, then at least one of G and $\overline{G}$ is nonplanar.

15. Show that the average degree in a planar graph is actually less than 6. (Note that this provides an alternate proof to Corollary 6.1.2.)

16. Use the DMP algorithm to test the planarity of the following graphs.





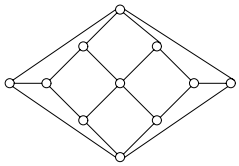
17. Prove path properties 1 – 5 relating to the Hopcroft-Tarjan planarity algorithm.

18. Use the Hopcroft-Tarjan planarity algorithm to test the planarity of the graphs from exercise 16 in Chapter 6.

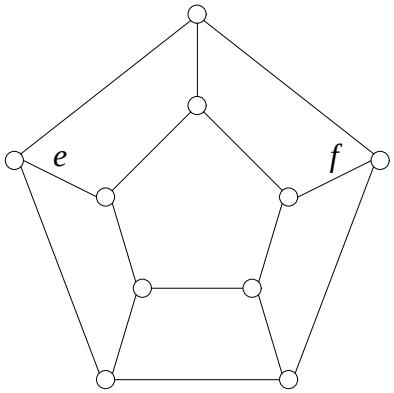
19. Prove Theorem 6.4.1.

20. How might you actually keep track of the paths both inside and outside of a given cycle? How much information must actually be recorded?

21. Use Grinberg's theorem to show that the graph below is not hamiltonian.



22. Show that no hamiltonian cycle in the graph below can contain both of the edges *e* and *f*.



References

1. Euler, L., Demonstratio Nonnullarum Insignium Proprietatum Quibus Solida Hedris Planis Inclusa Sunt Praedita. *Novi Comm. Acad. Sci. Imp. Petropol*, 4(1758), 140 – 160.
2. Demoucron, G., Malgrange, Y., and Pertuiset, R., Graphes Planaires: Reconnaissance et Construction de Représentations Planaires Topologiques. *Rev. Francaise Recherche Opeŕationnelle,* 8(1964), 33 – 47.
3. Grinberg, E. J., Plane Homogeneous Graphs of Degree Three without Hamiltonian Circuits. *Latvian Math. Yearbook,* 4(1968), 51 – 58.
4. Hopcroft, J., and Tarjan, R. E., Efficient Planarity Testing. *JACM*, 21(1974), 549 – 568.
5. Kuratowski, G., Sur le Probléme des Courbes Gauches en Topologie. *Fund. Math.*, 15(1930), 271 – 283.
6. Whitney, H., Planar Graphs. *Fund. Maths.*, 21(1933), 73 – 84.

Chapter 7
Matchings and r-Factors

Section 7.0 Introduction

Suppose you have your own company and you have several job openings to fill. Further, suppose you have several candidates to fill these jobs and you must somehow decide which candidates are to fill which jobs. Let's try to model this problem using graphs. The most natural model takes the form of a bipartite graph. Suppose that corresponding to each of *m* open jobs we associate a vertex and say we call these vertices $j_1, j_2, \ldots, j_m$. Also, corresponding to each of *n* job applicants we associate a vertex, say $a_1, \ldots, a_n$. Now, we join vertex a_i to vertex j_k if, and only if, applicant a_i is qualified for job j_k. We clearly have created a bipartite graph. A solution to our hiring dilemma is to find a set of edges that "match" each job to some distinct applicant. Clearly, our problem would make even more sense if we were to somehow rate the applicants and their "suitability" to handle each job. That is, we associate a measure of suitability (or unsuitability) with each edge in our model. An optimal solution, then, would be to find a set of job assignments that maximizes (or minimizes) the sum of these measures (generally called weights). We will consider this enhancement later. For now, we will be satisfied with merely finding suitable pairings.

We shall begin with a detailed investigation of such pairings in bipartite graphs. Our goal is to find an effective method of determining the best possible pairing, whether it be in terms of most edges used or in terms of optimizing some weight function. We shall investigate both theoretic and algorithmic approaches. We shall ultimately see that this area is a meeting point for many different ideas in discrete mathematics. This will provide us with a chance to use diverse techniques and apply our results in many interesting and unusual ways.

Section 7.1 Matchings and Bipartite Graphs

More formally, two distinct edges are *independent* if they are not adjacent. A set of pairwise independent edges is called a *matching.* Thus, to solve our job assignment problem, we seek a matching with the property that each job j_i is incident to an edge of the matching. In most situations, it is not merely a matching that we want, but the largest possible matching with respect to some measurable quantity. Here, we wish the maximum number of jobs to be filled, but in other situations there may be better ways to measure how successfully we have formed our matching. In *G*, a matching of maximum cardinality is called a *maximum matching* and its cardinality is denoted $\beta_1(G)$. A matching that pairs all the vertices in a graph is called a *perfect matching.*

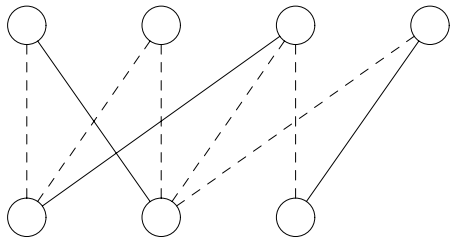


Figure 7.1.1. The solid edges form a maximum matching.

In a study of matchings, several useful observations will actually take us a long way toward our goal. Berge [2] made perhaps the most applicable of these observations. Following his terminology, we define an edge to be *weak with respect to a matching M* if it is not in the matching. A vertex is said to be *weak with respect to M* if it is only incident to weak edges. An *M-alternating path* in a graph *G* is a path whose edges are alternately in a matching *M* and not in *M* (or conversely). An *M-augmenting path* is an alternating path whose end vertices are both weak with respect to *M*. Thus, an *M*-augmenting path both begins and ends with a weak edge. If it is clear what matching we are using, we will simply say alternating path or augmenting path. The graph of Figure 7.1.2 contains a matching *M* with edges 23, 54 and 78. An augmenting path containing these edges is shown with nonmatching edges dashed. With this terminology in mind, we will find the following lemma extremely useful.

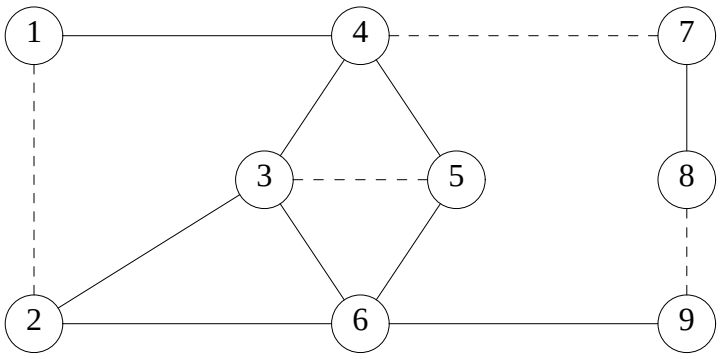


Figure 7.1.2. Augmenting path 1, 2, 3, 5, 4, 7, 8, 9 for *M*

Lemma 7.1.1 Let M_1 and M_2 be two matchings in a graph *G*. Then each component of the spanning subgraph *H* with edge set

$$E(H) = (M_1 - M_2) \cup (M_2 - M_1)$$

is one of the following types:

1. An isolated vertex.
2. An even cycle with edges alternately in M_1 and M_2.
3. A path whose edges are alternately in M_1 and M_2 and such that each end vertex of the path is weak with respect to exactly one of M_1 and M_2.

Proof. It is easily seen that $\Delta(H) \leq 2$, since no vertex can be adjacent to more than one edge from each matching. Thus, the possible components are paths, cycles or isolated vertices.

Now, consider a component in H that is not an isolated vertex. It is easily seen that in any such component, the edges must alternate, or the definition of matching would be violated. Hence, if the component is a cycle, it must be even and alternating. Finally, assume the component is a path. Then, we must only show that each end vertex is weak with respect to exactly one of the matchings. Clearly, each end vertex is already adjacent to an edge of one of the matchings. Suppose it was adjacent to an edge e from the other matching, without loss of generality, say $e \in M_1 - M_2$. Since we can now extend the path in question, we violate the fact that our vertex was an end vertex of a path and that this path was a component of H. Thus, we must have one of the three possibilities listed above. □

We now present Berge's [2] characterization of maximum matchings.

Theorem 7.1.1 A matching M in a graph G is a maximum matching if, and only if, there exists no M-augmenting path in G.

Proof. Let M be a matching in G and suppose that G contains an M-augmenting path

$$P\colon v_0, v_1, \ldots, v_k,$$

where k is clearly odd. If M_1 is defined to be

$$M_1 = (M - \{ v_1v_2, v_3v_4, \ldots, v_{k-2}v_{k-1} \}) \cup \{ v_0v_1, v_2v_3, \ldots, v_{k-1}v_k \},$$

then M_1 is a matching in G, and it contains one more edge than M; thus, M is not a maximum matching.

Conversely, suppose that M is not a maximum matching and there does not exist an M-augmenting path and let M_1 be a maximum matching in G. Now, consider the spanning subgraph H, where $E(H)$ is the symmetric difference of M and M_1 (that is, $(M - M_1) \cup (M_1 - M)$). By Lemma 7.1.1, we know the possibilities for the components of H. By our earlier observations, we know that some alternating path in H must contain more edges of M_1 than M, since M_1 contains more edges than M. But, then, this path must be an M-augmenting path in G, contradicting our assumptions that

there were no augmenting paths in G. □

The situation presented in the job assignment problem is very common. One often wishes to find a matching that uses every vertex in some set. Given a matching M, we will say that a set S *is matched under* M if every vertex of S is incident to an edge in M. For bipartite graphs, Hall [11] first determined necessary and sufficient conditions under which a set could be matched.

Theorem 7.1.2 Let $G = (X \cup Y, E)$ be a bipartite graph. Then X can be matched to a subset of Y if, and only if, $|N(S)| \geq |S|$ for all subsets S of X.

Proof. Suppose that X can be matched to a subset of Y. Then, since each vertex of X is matched to a distinct vertex of Y, it is clear that $|N(S)| \geq |S|$ for every subset S of X.

Conversely, suppose that G is bipartite and that X cannot be matched to a subset of Y. We wish to construct a contradiction to the assumed neighborhood conditions. Thus, consider a maximum matching M in G. By our assumptions, the edges of M are not incident with all the vertices of X. Let u be a vertex that is weak with respect to M and let A denote the set of all vertices of G connected to u by an M-alternating path. Since M is a maximum matching, it follows from Berge's theorem (Theorem 7.1.1) that u is the only weak vertex of A. Let $S = A \cap X$ and $T = A \cap Y$.

Clearly, the vertices of $S - \{ u \}$ are matched with vertices of T; therefore, $|T| = |S| - 1$ and $T \subseteq N(S)$. In fact, T must equal $N(S)$ since every vertex in $N(S)$ is connected to u by an alternating path. But then $|N(S)| = |S| - 1 < |S|$, contradicting our neighborhood assumption. □

An easy and well-known corollary to Hall's theorem can now be presented.

Corollary 7.1.1 If G is a k-regular bipartite graph with $k > 0$, then G has a perfect matching.

Proof. Let $G = (X \cup Y, E)$ be a k-regular bipartite graph. Then $k|X| = k|Y| = |E|$, and since $k > 0$, we see that $|X| = |Y|$. For any $A \subseteq V(G)$, let E_A be the set of edges of G incident with a vertex of A. Let $S \subseteq X$ and consider E_S and $E_{N(S)}$. By the definition of $N(S)$, we see that $E_S \subseteq E_{N(S)}$. Thus,

$$k|N(S)| = |E_{N(S)}| \geq |E_S| = k|S|,$$

and so $|N(S)| \geq |S|$. Thus, by Hall's theorem, X can be matched to a subset of Y. But since $|X| = |Y|$, we see that G must contain a perfect matching. □

Hall's theorem is a very flexible and useful result. It can be seen from many different points of view, and it can be stated in many ways. We shall now state it in set theoretic terms. To do this, we need some terminology. Given sets $S_1, \ldots, S_k$, we say any element $x_i \in S_i$ is a *representative* for the set S_i which contains it.

Our purpose is to find a collection of distinct representatives for the sets $S_1, \ldots, S_k$. This collection is usually known as a *system of distinct representatives* or a *transversal* of the sets. From a graph point of view, we could use a vertex s_i to represent each set S_i. We could also use a distinct vertex u_j to represent each of the elements x_j in each of the sets. We then join vertices s_i and u_j if, and only if, the element x_j is in the set S_i. In this way, we see that $N(s_i) = \{ u_j \mid x_j \in S_i \}$. It is now easy to see that finding a system of distinct representatives is equivalent to finding a matching of the s_i's into a subset of the u_j's. We now restate Hall's theorem in set terms.

The SDR Theorem A collection $S_1, S_2, \ldots, S_k$, $k \geq 1$ of finite nonempty sets has a system of distinct representatives if, and only if, the union of any t of these sets contains at least t elements for each t, $(1 \leq t \leq k)$.

Another popular version of Hall's theorem takes the form of a statement on marriage. Our goal this time is to match as many men to women as possible so that the maximum number of couples can be married. This matching of men to women is the reason Hall's theorem is often called the marriage theorem.

The Marriage Theorem Given a set of n men and a set of n women, let each man make a list of the women he is willing to marry. Then each man can be married to a woman on his list if, and only if, for every value of k $(1 \leq k \leq n)$, the union of any k of the lists contain at least k names.

We now consider a related result from König [12] and Egerváry [6]. A set C of vertices is said to *cover* the edges of a graph G (or be an *edge cover*), if every edge in G is incident to a vertex in C. The minimum cardinality of an edge cover in G is denoted $\alpha(G)$. In Figure 7.1.3 we see a bipartite graph with a matching (dashed edges). The solid vertices form a cover in this graph.

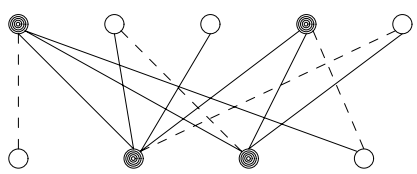


Figure 7.1.3. A matching and cover in a graph.

The König-Egerváry theorem relates matchings and covers. The proof technique is

reminiscent of those already seen in this section.

Theorem 7.1.3 If $G = (X \cup Y, E)$ is a bipartite graph, then the maximum number of edges in a matching in G equals the minimum number of vertices in a cover for $E(G)$, that is, $\beta_1(G) = \alpha(G)$.

Proof. Let a maximum matching in G contain $\beta_1(G) = m$ edges and let a minimum cover for $E(G)$ contain $\alpha(G) = c$ vertices. Note that $c \geq m$ always holds.

Let M be a maximum matching in G. Also let W be those vertices of X that are weak with respect to M. Note that $|M| = |X| - |W|$. Let S be those vertices of G that are connected to some vertex in W by an alternating path. Define $S_X = S \cap X$ and $S_Y = S \cap Y$.

From the definition of S and the fact that no vertex of $S_X - W$ is weak, we see that $S_X - W$ is matched under M to S_Y and that $N(S_X) = S_Y$. Since $S_X - W$ is matched to S_Y, we see that $|S_X| - |S_Y| = |W|$.

Let $C = (X - S_X) \cup S_Y$. Then C is a cover for $E(G)$, for if it were not, there would be an edge vw in G such that $v \in S_X$ and $w \notin S_Y = N(S_X)$. Hence,

$$|C| = |X| - |S_X| + |S_Y| = |X| - |W| = |M|$$

Thus, $c = m$, and the proof is complete. □

The form of the König-Egerváry theorem should by now be a tipoff that something deeper is going on here. The min-max form that we saw in Menger's theorem and in the max flow-min cut theorem is once again present. Thus, we should expect that these results are closely related (see the exercises) and that flows could be used to prove results about matchings. We now investigate this connection.

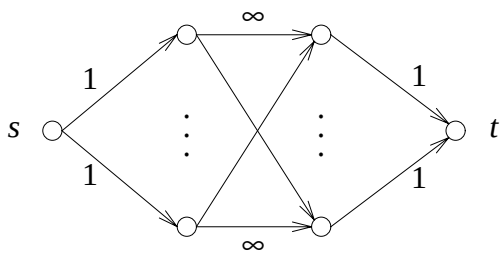


Figure 7.1.4. The network N_G.

Given a bipartite graph $G = (X \cup Y, E)$, we construct a network N_G (see Figure 7.1.4) corresponding to G by first orienting all edges of G from X to Y. Now, insert a source vertex s with arcs to all vertices of X and a sink vertex t with arcs from all vertices of Y. We assign the capacity of all arcs out of s or into t as 1. The capacities of all arcs from X to Y are set to ∞. With this network in mind, we are now able to show a

connection between matchings and flows.

Theorem 7.1.4 In a bipartite graph $G = (X \cup Y, E)$, the number of edges in a maximum matching equals the maximum flow in the network N_G.

Proof. Let M be a maximum matching in G. For each edge xy in M, we use the directed path s, x, y, t to flow 1 unit from s to t in N_G. It is clear that these paths are all disjoint except for s and t. Thus, $F \geq |M| = \beta_1(G)$.

Now let f be an integral flow function on the network N_G corresponding to G. All the directed paths between s and t have the form s, x, y, t. If such a path is used to carry flow from s to t, then no other arc can be used to carry flow to y. Also, no other arc can be used to carry flow out of x. Then the set of edges xy for which $f(x \rightarrow y) = 1$ determines a matching in G. Thus, $\beta_1(G) = |M| \geq F$, and this, combined with our previous observations, shows that $\beta_1(G) = |M| = F$. □

It is a simple matter now to deduce from the max-flow min-cut theorem and Theorem 7.1.3 that $\alpha(G)$ must equal the capacity of a minimum cut. But we can do more than just state this equality; we can use cuts to determine the cover. Suppose that $(C, \overline{C})$ is a cut of minimum capacity in N_G. If we let $A = X \cap \overline{C}$ and $B = Y \cap C$, then it is easy to see that $A \cup B$ is a cover for G. Further, since

$$c(s, A) + c(B, t) = |A \cup B|$$

(that is, since the capacity of the arcs from s to A and those from B to t total $|A \cup B|$), we see that $A \cup B$ must be a minimum cover. Thus, we can use flows and cuts to find not only maximum matchings but also minimum covers as well. Does all this remind you of the way we selected the cover in the proof of Theorem 7.1.3?

Section 7.2 Matching Algorithms and Marriage

It turns out that in the setting of bipartite graphs, we can easily apply Berge's ideas and use augmenting paths to build maximum matchings. We now present a labeling algorithm to find a maximum matching in a bipartite graph. This algorithm assumes a given matching M is known and attempts to extend M by finding an augmenting path. This is done by trying to follow all possible augmenting paths. As we go, we "mark" vertices using edges not in M while walking from X to Y, and we mark vertices using edges in M while walking from Y to X. Hence, we essentially trace all possible alternating paths as we go. This algorithm is a special case of the network flow algorithm of Ford and Fulkerson [9].

Algorithm 7.2.1 A Maximum Matching in a Bipartite Graph.

Input: Let $G = (X \cup Y, E)$ be a bipartite graph and suppose that $X = \{ x_1, \ldots, x_m \}$ and $Y = \{ y_1, \ldots, y_n \}$. Further, let M be any matching in G (including the empty matching).

Output: A matching larger than M or the information that the present matching is maximum.

Method: We now execute the following labeling steps until no step can be applied.

1. Label with an * all vertices of X that are weak with respect to M. Now, alternately apply steps 2 and 3 until no further labeling is possible.
2. Select a newly labeled vertex in X, say x_i, and label with x_i all unlabeled vertices of Y that are joined to x_i by an edge weak with respect to M. Repeat this step on all vertices of X that were labeled in the previous step.
3. Select a newly labeled vertex of Y, say y_j, and label with y_j all unlabeled vertices of X which are joined to y_j by an edge in M. Repeat this process on all vertices of Y labeled in the previous step.

Notice that the labelings will continue to alternate until one of two possibilities occurs:

$E1$: A weak vertex in Y has been labeled.
$E2$: It is not possible to label any more vertices and $E1$ has not occurred.

If ending $E1$ occurs, we have succeeded in finding an M-augmenting path, and we can construct this path by working backwards through the labels until we find the vertex of X which is labeled *. The purpose of the labels is to allow us to actually determine an M-augmenting path. We can then extend our matching as in Theorem 7.1.1 and repeat the algorithm on our new matching. Our next theorem shows that if $E2$ occurs, M is already a maximum matching. The proof is reminiscent of that of the König-Egerváry theorem.

Theorem 7.2.1 Suppose that Algorithm 7.2.1 has halted with ending $E2$ occurring and having constructed matching M. Let U_X be the unlabeled vertices in X and L_Y the labeled vertices in Y. Then $C = U_X \cup L_Y$ covers the edges of G, $|C| = |M|$, and M is a maximum matching in G.

Proof. Suppose that C does not cover the edges of G. Then there must exist an edge from $L_X = X - U_X$ to $U_Y = Y - L_Y$. Suppose there was such an edge, call it $e = xy$, where $x \in L_X$ and $y \in U_Y$. If e is not in M, then since x is labeled, it follows from step 2 that y is labeled, and this condition contradicts the fact that L_Y contains all the labeled vertices of Y. Thus, $e \in M$, and so it follows from step 3 that the label on x is y. It also follows from the algorithm that y must be labeled; and that in fact, it must have received that label prior to x receiving its label. But this condition again contradicts the fact that

L_Y contains all the labeled vertices of Y. Thus, we conclude there are no edges from $X - U_X$ to $Y - L_Y$, and so it must be the case that C covers all the edges of G.

Now, consider $y \in L_Y$. Since y is labeled and $E1$ has not happened, y must be incident with an edge of M (exactly one such edge since M is a matching). Suppose that xy is this edge. By step 3, the vertex x must be labeled, so x is not in U_X. Consider some $x_1 \in U_X$. Since x_1 is not labeled, it must be incident with an edge of M, or it would have received the label * in step 1. Since M is a matching, x_1 is incident with exactly one edge of M. Let this edge be x_1y_1. If y_1 were labeled, by step 3 we would see that x_1 would also be labeled, but $x_1 \in U_X$. Then y_1 must be unlabeled, and thus, none of the edges of M which are incident to vertices in U_X are the same as any of the edges of M with incidences in L_Y. Since every edge of M has an end vertex in either U_X or L_Y, there must be as many edges in M as vertices in C; that is, $|C| = |M|$. Since C covers the edges of G, by Theorem 7.1.3, M must be a maximum matching, and so the proof is complete. □

Example 7.2.1. We now apply Algorithm 7.2.1 to the bipartite graph of Figure 7.2.1.

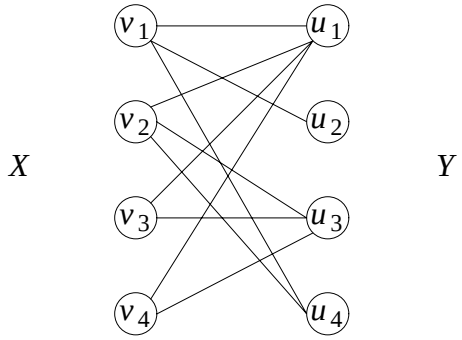


Figure 7.2.1. A bipartite graph $G = (X \cup Y, E)$.

We select the edge v_1u_1 as our initial matching M. We now apply Algorithm 7.2.1.
Step 1: Label v_2, v_3, v_4 with *.
Step 2: Select v_2 and label u_1, u_3, u_4 with v_2.
Step 3: Select u_1 and label v_1 with u_1.
Step 2: Select v_1 and label u_2 with v_1.
Note that no other labeling is possible.

Since the labeling included weak vertices in Y, condition $E1$ holds. Note that the path $P: v_2, u_3$ is augmenting; thus our new matching is now $M = \{v_1u_1, v_2u_3\}$, and we repeat Algorithm 7.2.1 on this M.

To see what this algorithm is really doing, we can trace what happens in each pass of the algorithm. We began labeling v_2 and followed by labeling the weak neighbors u_1, u_3 and u_4. From these vertices we looked instead for edges in the matching M and labeled v_1. Then, we again reversed our thinking and looked for weak neighbors of this

vertex. In Figure 7.2.2 we picture the situation after the labeling was completed. Note the layering of vertices and the fact that edges between consecutive layers were introduced in the same step of the algorithm. The tree that has been "grown" by the algorithm has the property that each path from the initial vertex to a leaf is an alternating path. When the tree has been grown to its utmost, the algorithm halts, and any path from the root (the vertex labeled *) to a weak leaf is augmenting. We retrace any such path by following the labels assigned to the vertices. It has become customary to call such a tree a *hungarian tree.* Note that such a tree has been started for every vertex labeled *, but not all have been successful in finding an augmenting path.

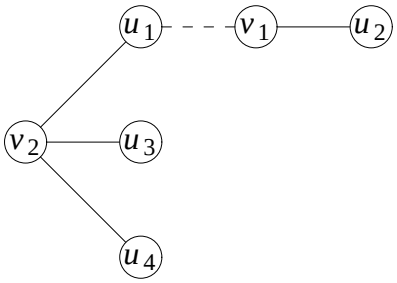


Figure 7.2.2. A hungarian tree grown in pass one.

The second pass of Algorithm 7.2.1 (see Figure 7.2.3) produces:
Step 1: Label v_3, v_4 with *.
Step 2: Select v_4 and label u_1, u_3 with v_4.
Select v_3; nothing more can be labeled.
Step 3: Select u_1 and label v_1 with u_1. Select u_3 and label v_2 with u_3.
Step 2: Select v_1 and label u_2, u_4 with v_1.

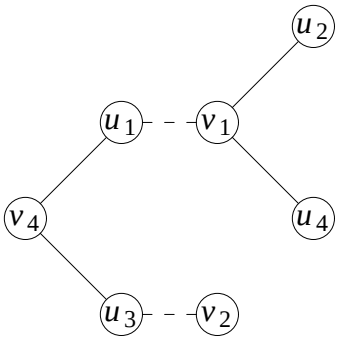


Figure 7.2.3. The hungarian tree rooted at v_4 in pass two.

The labeling halts, and we have found an augmenting path, namely P: u_4, v_1, u_1, v_4. Using P, we extend the matching to

$$M = \{ v_2u_3, u_4v_1, u_1v_4 \},$$

and we repeat Algorithm 7.2.1 again on M.

Step 1: Label v_3 with *.
Step 2: Select v_3 and label u_1, u_3 with v_3.
Step 3: Select u_1 and label v_4 with u_1;
then select u_3 and label v_2 with u_3.
Step 2: Select v_2 and label u_4 with v_2;
then select v_4 and note that no further labeling can be done.
Step 3: Select u_4 and label v_1 with u_4.
Step 2: Select v_1 and label u_2 with v_1.
Step 3: No labeling is possible, and the algorithm halts.

We now interchange edges on the path P: $u_2, v_1, u_4, v_2, u_3, v_3$ to obtain the maximum matching $M = \{ u_1v_4, v_3u_3, v_2u_4, v_1u_2 \}$ (see Figure 7.2.4).

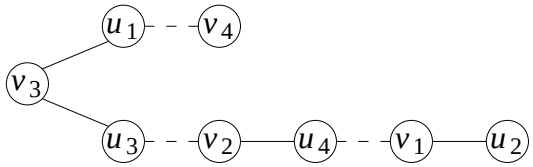


Figure 7.2.4. The hungarian tree from pass three.

We are now ready to consider a strengthening of the job assignment problem. We wish to include information about the relative suitabilities of the job candidates for the various jobs. The problem to be considered will be restricted to the case in which there are n candidates for n jobs, and each candidate has a measure of suitability for each job. That is, we have assigned a weight function to the edges of $K_{n,n}$. It is clear that it may not be possible to assign each applicant to the job he or she is best suited for, since two applicants might be best suited for the same job. Thus, our goal is to find the overall best solution, that is, the solution with the optimal sum of the weights assigned to the edges of the matching.

To attack this problem, we will find it more convenient to have our weight function represent a measure of the applicant's unsuitability for the job. Then, the larger the weight, the more unsuitable the applicant is for the job. For any matching M, we define the *weight of the matching* to be $W(M) = \sum_{e \in M} w(e)$. Thus, an optimal solution will be a perfect matching with $W(M)$ a minimum. We now present an algorithm for finding such a solution.

We begin by representing our graph in matrix form, $U = [w_{i,k}]$ where $w_{i,k}$ is the weight of the edge joining j_i and a_k (that is, the unsuitability of applicant k to job i). An

example of such a matrix is now given.

U	a_1	a_2	a_3	a_4
j_1	4	6	14	11
j_2	7	2	8	9
j_3	3	13	1	4
j_4	5	2	0	13

It is important to note that our solution is unchanged if we subtract the same number from all members of some row or some column. This follows since only one entry will be selected from any row or column; hence, the value of $W(M)$ for any matching M will be reduced by the same amount. Thus, we can make the entries in our unsuitability matrix easier to deal with by first subtracting from each row the minimum entry in that row. The resulting matrix still has all nonnegative entries, which we hope are smaller than before. Our example matrix thus becomes:

U	a_1	a_2	a_3	a_4
j_1	0	2	10	7
j_2	5	0	6	7
j_3	2	12	0	3
j_4	5	2	0	13

Now, subtract from each column the smallest entry in that column to obtain a further reduced unsuitability matrix. Our example is then:

U	a_1	a_2	a_3	a_4
j_1	0^*	2	10	4
j_2	5	0^*	6	4
j_3	2	12	0	0^*
j_4	5	2	0^*	10

Our problem now is to select numbers from the table, no two in the same row or column, with as small a sum as possible. Since our entries are all nonnegative, the smallest sum we could hope for is zero. Thus, if n zeros can be found, no two in the same row or column, an optimal solution will be obtained. In our example, a solution is easily found. We select the entries starred above.

The suspicious reader is now asking what happens if at this stage we cannot find a suitable set of n "independent zeros," or must this always be the case. The answer is that we are not always sure of having enough zeros at this stage to represent a perfect matching in our graph. Sometimes, further adjustments must be made. Consider the following unsuitability matrix.

U	a_1	a_2	a_3	a_4
j_1	6	8	2	7
j_2	5	8	13	9
j_3	2	8	10	9
j_4	4	12	8	11

Then, after reducing the rows followed by the columns, we are left with the following matrix.

U	a_1	a_2	a_3	a_4
j_1	4	3	0	1
j_2	0	0	8	0
j_3	0	3	8	3
j_4	0	5	4	3

This matrix does not contain four independent zeros since all the zeroes are contained in the first column and the first two rows. This can be seen by crossing with a line the rows and columns containing zeros. In the graph, then, the independent zeros represent the edges of the matching, while the lines drawn show the vertices "covered" by the vertex corresponding to the row or column in which the line was drawn. Our adjustment procedure is as follows:

1. Let *m* be the smallest number that is not included in any of our crossed rows or columns.
2. Subtract *m* from all uncrossed numbers.
3. Leave numbers which are crossed once unchanged.
4. Add *m* to all numbers which are crossed twice.

This procedure produces at least one more zero in the uncrossed portion of our matrix and leaves all the zeros unchanged, unless they happen to be crossed twice. Can you explain why this adjustment procedure works? Our example becomes:

U	a_1	a_2	a_3	a_4
j_1	7	3	0	1
j_2	3	0	8	0
j_3	0	0	5	0
j_4	0	2	1	0

The procedure described here will always yield a set of *n* independent zeros after a finite number of repetitions. The algorithm presented above to solve our optimal matching problem is usually known as the hungarian algorithm, in honor of König and Egerváry. An alternate form of the König-Egerváry theorem can now be stated. We derive its proof from the first form.

Theorem 7.2.2 Let S be any $m \times n$ matrix. The maximum number of independent

zeros which can be found in S is equal to the minimum number of lines (either rows or columns) which together cover all the zeros of S.

Proof. Construct a bipartite graph $G = (X \cup Y, E)$ modeling our matrix as follows. Let the vertices of X correspond to the rows of our matrix and the vertices of Y to the columns. We join x_i and y_j if, and only if, entry i, j of our matrix is zero. Then, a maximum independent set of zeros corresponds to a maximum matching of G, and a minimum set of lines covering all the zeros corresponds to a minimum covering of G. Thus, by Theorem 7.1.3 the result follows. □

Suppose we now consider the job assignment problem from the greedy point of view. Can we simply begin with the edge of minimum cost and somehow extend to a matching of minimum cost? The answer is that we can if we are careful about our point of view. Rather than build a matching by greedily taking edges, we shall set our view on the vertices involved. Given a bipartite graph $G = (V_1 \cup V_2, E)$, we say a subset I of V_1 is *matching-independent* for matchings of V_1 into V_2 if there is a matching which matches all the elements of I to elements of V_2. We wish to build a maximum sized matching-independent set in a greedy fashion. Thus, if we have a set I that is matching-independent, we would add to I the vertex x of V_1 having cheapest incident edge that still allows us to match $I \cup \{ x \}$ into V_2. When we can no longer do this, we stop. The question of interest now is: How do we know that we have formed a maximum sized matching-independent set when this process halts? That this is indeed the case can be concluded from the next result.

Theorem 7.2.3 Matching-independent sets for matchings of V_1 into V_2 satisfy the following rule:

If I and J are matching-independent subsets of V_1 and $|I| < |J|$ then there is an element x of J such that $I \cup \{ x \}$ is matching-independent.

Proof. Suppose that M_1 is a matching of I into V_2 and M_2 is a matching of J into V_2. Then, by Lemma 7.1.1, the spanning subgraph H with $E(H) = (M_1 - M_2) \cup (M_2 - M_1)$ has connected components of only three possible types. Since $|M_2| > |M_1|$ at least one of these components must be of type 3 in Lemma 7.1.1. Thus, there is a path P whose edges are alternately in M_2 and M_1 and whose first and last edges are in M_2. Each vertex of P incident to an edge from M_1 is also incident to an edge from M_2. Further, there is a vertex x in V_1 (and J) incident to an edge from M_2, and x is not incident to any edge from M_1. Now the set of edges

$$M = (M_1 - E(P)) \cup (E(P) - M_1)$$

forms a matching with one more edge than M_1. Also, M is a matching of $I \cup \{ x \}$ into V_2. Thus, $I \cup \{ x \}$ is matching-independent, and since $x \in J$, the proof is complete.

□

To actually use the greedy approach to construct a maximum sized matching of minimum weight, we must determine a method that allows us to select the vertex x we wish to add to our matching-independent set. To do this, we must also keep track of the edges that are presently matching I into V_2. Otherwise, we would face the possibility of having to check all possible subsets of V_2 in a search for the matching. Since this is clearly an exponential process, the bookkeeping of the intermediate matchings is necessary. Applying our methods of finding alternating paths allows us to construct the maximum sized matching-independent set, and it is an exercise to show that the corresponding matching is of minimum cost.

Can we vary the assignment problem somewhat? For a suitability weight function w, we can change the function we are optimizing from

$$\sum_{e \in M} w(e) \text{ to } \min_{e \in M} \{ w(e) \}.$$

That is, suppose we try to maximize the minimum weight of an edge in the matching. This is the mathematical version of the old proverb that the strength of a chain equals the strength of its weakest link. This is known as the *bottleneck assignment problem.* It turns out that we can solve the bottleneck assignment problem by repeated applications of any algorithm for finding matchings in bipartite graphs. Suppose we begin with any matching M in the bipartite graph G. We can easily find the minimum weight of an edge in M, say b. We form a new graph G_b from G by removing all edges from G with weight b or less. If we now find a maximum matching in G_b, and if it is a perfect matching, then each of its edges must have weight greater than b, so we have improved the matching. If no such matching can be found, then the previous matching was the best. We continue this process until the matching which maximizes the minimum weight of an edge is found. Can you formally write an algorithm that solves the bottleneck assignment problem?

We conclude this section with a study of some interesting mathematical properties of marriage. For the remainder of this section we shall use some notions about matrices to study marriages. We take the marriage point of view because of the interesting and unusual manner the statements of our results will take. We begin with some ideas on matrices.

A matrix $D = (d_{i,j})$ is *doubly stochastic* if each $d_{i,j} \geq 0$ and the sum of the entries in any row or column equals 1. A *permutation matrix* is any matrix obtained from the identity matrix I by performing a permutation on the rows of I. A well-known result on doubly stochastic matrices from Birkhoff [4] and Von Neumann [15] states that any doubly stochastic $n \times n$ matrix D can be written as a combination of suitable permutation matrices. That is, there exist constants $c_1, c_2, \ldots, c_n$ and permutation matrices $P_1, \ldots, P_n$ such that $D = \sum_{i=1}^{n} c_i P_i$.

We can use matchings to indicate an algorithm for finding the constants and the decomposition of D into permutation matrices.

Suppose we model our doubly stochastic matrix D with a bipartite graph. Let vertices $r_1, r_2, \ldots, r_n$ represent the rows of D and let vertices $k_1, \ldots, k_n$ represent the columns. We draw an edge from r_i to k_j if, and only if, entry $d_{i,j}$ of D is nonzero. Then the permutation matrix P_1 represents the edges of a matching in this bipartite graph and the constant c_1 is the minimum weight of an edge in this matching. We can now write D as $D = c_1P_1 + R$, where the matrix R represents the remaining edges of our bipartite graph. The old edges were adjusted by subtracting c_1 from the weight of each edge of the matching and removing any edge with weight zero. We now repeat this process on R.

Suppose that at some stage we are unable to find a matching. Then by Hall's theorem there must exist some set A of vertices representing rows of D such that $|A| > |N(A)|$. That is, there are more rows than "neighboring" columns. Now, consider what this means in our matrix D. If each of these rows sums to 1 (counting the entries that were possibly removed prior to this), then the total value of the weights in these rows is $|A|$. But then this amount must also be distributed over $|N(A)|$ columns, which means some column must sum to more than 1, contradicting that D was doubly stochastic. Thus, we will be able to find a matching at each stage.

We now formally state the algorithm for finding this convex sum of permutation matrices.

Algorithm 7.2.2 Decomposing Doubly Stochastic Matrices.
Input: A doubly stochastic matrix D.
Output: A convex sum of permutation matrices $c_1P_1 + \cdots + c_nP_n$.

1. Set $t_1 \leftarrow 1$, $X \leftarrow D$ and $k \leftarrow 1$.
2. Having a doubly stochastic matrix X and nonnegative numbers $t_1, t_2, \ldots, t_k$ and permutation matrices $P_1, \ldots, P_{k-1}$ such that
   $D = t_1P_1 + \cdots + t_{k-1}P_{k-1} + t_k X$ and $\sum_{i=1}^{k} t_i = 1$.
   If X is a permutation matrix,
   then set $P_k \leftarrow X$ and halt;
   else use bipartite graphs to find a permutation matrix X^* such that $x_{ij}{}^* = 0$ whenever $x_{ij} = 0$.
3. The n entries $x_{i,j}$ of X for which $x_{ij}{}^* = 1$ are all positive entries. Let c be the least of these entries (note $c < 1$).
   Set $t \leftarrow t_k$, $t_k \leftarrow ct$, $t_{k+1} \leftarrow (1-c)t$ and $P_k \leftarrow X^*$. Now, replace X by $\frac{1}{1-c}(X - cP_k)$, set $k \leftarrow k + 1$ and go to step 2.

Using doubly stochastic matrices, we find that another unusual theorem about marriage is now possible. Suppose we consider a suitability matrix describing the marriage problem. That is, given a set of n men and another set of n women, let the matrix $S = (s_{i,j})$ be defined so that $s_{i,j}$ is a measure of the suitability or "happiness" of a marriage between man i and woman j. Our goal is to study the type of marriage that brings this collection of men and women the most "happiness." In particular, we will compare monogamy and polygamy. These relationships can be shown in a matrix $M = (m_{ij})$. Each row of the matrix M represents a man in our set of men and each column a woman. The entry $m_{i,j}$ in our marriage matrix M represents the fraction of time man i spends with woman j. Thus, monogamy would be a permutation matrix and polygamy a general doubly stochastic matrix. Our measure of the happiness of the present marriage relationship M will be $h(M) = \sum_{i,j} s_{i,j} m_{i,j}$. Our solution is then to find $\max_M h(M)$, where the maximum is taken over all doubly stochastic matrices M. But we note that

$$\begin{aligned} \max_M h(M) &= \max_{c_1, \ldots, c_n} h(c_1 P_1 + \cdots + c_n P_n) \\ &= \max_{c_1, \ldots, c_n} c_1 h(P_1) + \cdots + c_n h(P_n) \\ &= h(P_i) \quad \text{for some } i. \end{aligned}$$

(The above follows easily for the maximum $h(P_i)$). That is, the maximum corresponds to the matching represented by some permutation matrix. In other words, monogamy is the preferred mathematical state of marriage. We have just proven the following interesting marriage theorem.

Theorem 7.2.4 Among all forms of marriage, monogamy is optimal.

We conclude our study of marriage by considering its stability. Suppose we have a set of n men $m_1, \ldots, m_n$ and n women $w_1, \ldots, w_n$. Suppose, too, that man m_1 is married to woman w_1 and man m_2 to woman w_2. Further suppose that in reality m_2 prefers w_1 to his own wife and w_1 prefers m_2 to her own husband. It is easy to believe this is not a "stable" situation, in fact, we call such a pair of marriages *unstable.*

Let's construct two preference tables. In each table, the rows represent the men and the columns the women. The entries in any row of the first preference table are the integers 1 to n. This represents the order of preference of the women by the man corresponding to this row (with 1 being first choice). A similar description applies to the columns of the women's preference table.

Our problem, then, is given the two preference tables, can we find a stable set of marriages. That is, can we find a matching in which no pair of independent edges is unstable. We now describe an algorithm to produce our stable matching.

Algorithm 7.2.3 Stable Matching Algorithm.
Input: Given preference tables for the men and the women.
Output: A set of stable marriages

1. Each man proposes to his first choice.
2. The women with two or more proposals respond by rejecting all but the most favorable offer. However, no woman accepts a proposal.
3. The men that were rejected propose to their next choice. Those that were not rejected continue their offers.
4. We repeat step 3 until we reach a stage where no proposal is rejected.

Clearly, each woman can only reject a finite number (namely $n - 1$) of proposals, and so this process must eventually stop. We illustrate our algorithm on the following preference tables.

men	w_1	w_2	w_3	w_4
m_1	1	2	3	4
m_2	1	4	3	2
m_3	2	1	3	4
m_4	4	2	3	1

women	w_1	w_2	w_3	w_4
m_1	3	3	2	3
m_2	4	1	3	2
m_3	2	4	4	1
m_4	1	2	1	4

The set of proposals P_i appears below; starred proposals were rejected.

proposals	P_1	P_2	P_3	P_4	P_5	P_6
m_1	1	1	1	1^*	2^*	3
m_2	1^*	4	4	4	4	4
m_3	2	2	2^*	1	1	1
m_4	4	4^*	2	2	2	2

From this table we see that the final set of marriages is:

man m_1 with woman w_3
man m_2 with woman w_4
man m_3 with woman w_1
man m_4 with woman w_2.

It is easy to verify that this set of marriages is stable.

We now wish to prove we actually reach a stable matching. Suppose this were not the case; that is, suppose there was an unstable pair of marriages. Without loss of generality, let this pair be (m_1, w_1) and (m_2, w_2).

But if m_2 prefers w_1, he would have proposed to w_1 before he proposed to his present wife. Then w_1 would not have rejected m_2 if she actually preferred him over m_1. Hence, we could not have reached this unstable situation. Thus, the marriages cannot be unstable. We have now shown the following result, due originally to Gale and Shapely [10].

Theorem 7.2.5 Given n men and n women, there always exists a set of stable marriages.

Section 7.3 Factoring

We now wish to study matchings in a generalized setting. In addition, we want to consider relaxations of the concept of matchings. A perfect matching is often called a 1-factor, since the matching is a 1-regular spanning subgraph of the original graph. It is not a difficult leap to the idea of an r-factor, that is, an r-regular spanning subgraph of the original graph. We begin with a natural result.

Theorem 7.3.1 If G is a graph of order $2n$ and $\delta(G) \geq n$, then G contains a 1-factor.

Proof. This result follows as a consequence of Dirac's theorem (Corollary 5.2.1). □

We see that an algorithm for finding such a matching is apparent. Having obtained an r-matching, scan the remaining $(2n - 2r)$ vertices to see if any pair is joined by an edge. If this fails to be the case, choose any two of these vertices, say a and b, and scan the edges xy of the matching until one is found such that a is adjacent to x and b is adjacent to y. Then, replace xy by the edges ax and by and repeat this process.

The fundamental result on 1-factors is from Tutte [14]. The proof presented here is that of Anderson [1]. We denote the number of components of odd order in a graph G by $k_o(G)$.

Theorem 7.3.2 (Tutte [14]) A nontrivial graph G has a 1-factor if, and only if,

$k_o(G - S) \leq |S|$ for every proper subset S of $V(G)$.

Proof. Let F be a 1-factor of G and suppose there exists a proper subset S of $V(G)$ such that $k_o(G - S) > |S|$. For each of the odd components C of $G - S$, there must exist an edge in F that goes from C to S. But this implies that there is a vertex in S incident with at least two edges in F, which contradicts the definition of matching.

Conversely, note that $k_o(G - \phi) \leq |\varnothing| = 0$. Thus, G has only even components, and the order n of G must, then, be even. Also, observe that for every proper subset S of $V(G)$, the numbers $|S|$ and $k_o(G - S)$ are of the same parity.

Now, we proceed by induction on the order of G. If $n = 2$, then G must be K_2, and clearly G has a 1-factor. Next, assume that for all graphs H of even order less than n, the condition $k_o(H - S) \leq |S|$ for every proper nonempty subset S of vertices implies H has a 1-factor. Let G be a graph of even order n and assume that $k_o(G - S) \leq |S|$ for every proper subset S of $V(G)$. We now consider two cases.

Case 1. Suppose that $k_o(G - S) < |S|$ for all subsets S of $V(G)$ with $2 \leq |S| < n$. Since $k_o(G - S)$ and $|S|$ have the same parity, $k_o(G - S) \leq |S| - 2$. Let uv be an edge of G and consider $G - u - v$. Let S_1 be a proper subset of $V(G - u - v)$. Thus,

$$k_o(G - u - v - S_1) \leq |S_1|,$$

or else

$$k_o(G - u - v - S_1) > |S_1| = |S_1 \cup \{ u, v \}| - 2,$$

and, hence,

$$k_o(G - (S_1 \cup \{ u, v \})) \geq |S_1 \cup \{ u, v \}|,$$

which contradicts our assumptions. Then the matching obtained by applying the induction hypothesis, along with the edge uv, provides a 1-factor of G.

Case 2. Suppose that there exists some set S_2 such that $k_o(G - S_2) = |S_2|$. Among all such sets, let S be one of maximum cardinality. Further, let $k_o(G - S) = |S| = t$ and let $C_1, \ldots, C_t$ be the odd components of $G - S$. If E is an even component of $G - S$ and $x \in V(E)$, then $k_0(G - S - x)$ would equal $|S \cup \{ x \}|$ contradicting the fact that S was a set of maximum cardinality having this property. Thus, $G - S$ has no even components.

Let S_i $(i = 1, \ldots, t)$ denote those vertices of S with adjacencies in C_i. Each S_i is nonempty, or else some C_i would be an odd component of G. The union of any k of the sets $S_1, \ldots, S_t$ contains at least k vertices, or there exists an integer k such that the union U of some k of these sets contains less than k vertices. Thus, $k_o(G - U) > |U|$ which is a contradiction. Hence, by Hall's theorem (SDR), there exists a system of distinct representatives for the sets $S_1, \ldots, S_t$. This implies that in S there are distinct

vertices $v_1, \ldots, v_t$ and that in each C_i there is a vertex u_i such that $v_i u_i$ is an edge of G.

Let W be a proper subset of $C_i - u_i$. Since $C_i - u_i$ has even order, $k_o(C_i - u_i - W)$ and $|W|$ have the same parity. If

$$k_o(C_i - u_i - W) > |W|,$$

then it must be that

$$k_o(C_i - u_i - W) \geq |W| + 2.$$

Thus,

$$\begin{aligned} k_o(G - (S \cup W \cup \{ u_i \})) &= k_o(C_i - u_i - W) + k_o(G - S) - 1 \\ &\geq |S| + |W| + 1 \\ &= |S \cup W \cup \{ u_i \}| \end{aligned}$$

But this contradicts the maximality of S. Hence,

$$k_o(C_i - u_i - W) \leq |W|,$$

and so by induction, each $C_i - u_i$ has a 1-factor. These 1-factors, together with the edges $u_i v_i$, then form the desired 1-factor in G. □

Berge [3] noticed a useful related fact stemming from the proof of Tutte's theorem. This observation is often called the *Berge defect form* of Tutte's theorem. From Tutte's theorem, we see that a graph G of even order p contains a perfect matching unless there exists some set of r vertices whose removal leaves a graph with more than r odd components. However, because G has even order, this forces the existence of at least $r + 2$ odd components (see the proof). Further, the defect form also states that if G is a graph of odd order p, then G contains a maximum matching of size $\frac{1}{2}(p - 1)$ unless there is some set of r vertices whose removal leaves a graph with at least $r + 3$ odd components. This observation can be useful in dealing with graphs of odd order.

It is clear that every 1-regular graph contains (in fact, is) a 1-factor and that every 2-regular graph contains a 1-factor (in fact, is *1-factorable*, that is, its edge set can be decomposed into 1-factors) if, and only if, every component is an even cycle. The situation is not as simple for 3-regular graphs, however. Petersen [13] investigated 1-factors in 3-regular graphs and showed that they need not contain a 1-factor (see Figure 7.3.1). However, he was also able to show a situation in which such a graph would contain a 1-factor.

Theorem 7.3.3 (Petersen [13]). Every bridgeless 3-regular graph G can be expressed as the edge sum of a 1-factor and a 2-factor.

Proof. It suffices to show that such a graph contains a 1-factor since the remaining

edges form a 2-factor. Suppose the graph G fails to contain a 1-factor. Then by Tutte's theorem, there exists in G some proper nonempty set S of k vertices such that $n = k_o(G - S) > |S| = k$. Suppose that $C_1, \ldots, C_n$ are the odd components of $G - S$. There must exist an edge from each C_i to S $(1 \le i \le n)$, or else some C_i would be a 3-regular graph of odd order, which is impossible. Further, since G is bridgeless, there cannot be a single edge joining S to any C_i. If there were exactly two edges joining S to some C_i, then again C_i would contain an odd number of vertices of odd degree. Thus, at least three edges join any C_i to S. Thus, there are at least $3n$ edges joining S and the C_i $(1 \le i \le n)$. However, since each vertex of S has degree 3, there can be at most $3k$ edges into S. But since $3n > 3k$, a contradiction arises. Thus, no such set S can exist, and by Tutte's theorem, we see that G must contain a 1-factor. □

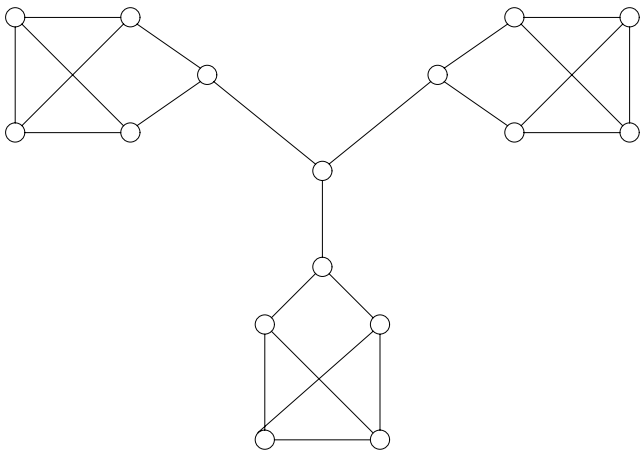


Figure 7.3.1. A 3-regular graph with no 1-factor.

Now that we know that every bridgeless 3-regular graph can be factored into a 1-factor and a 2-factor, it is natural to wonder if it can actually be 1-factored. Petersen also showed that this is not the case. His example, which has become perhaps the most famous of all graphs, is shown in Figure 7.3.2.

Petersen also characterized those graphs which are 2-factorable. It turns out that the obvious necessary condition that the graph be $2r$-regular for some $r \ge 1$ also suffices. The proof makes use of the fact that such graphs are eulerian.

Theorem 7.3.4 A nonempty graph G is 2-factorable if, and only if, G is $2r$-regular $(r \ge 1)$ for some integer r.

Proof. Clearly, if G is 2-factorable, then G is $2r$-regular for some $r \ge 1$.

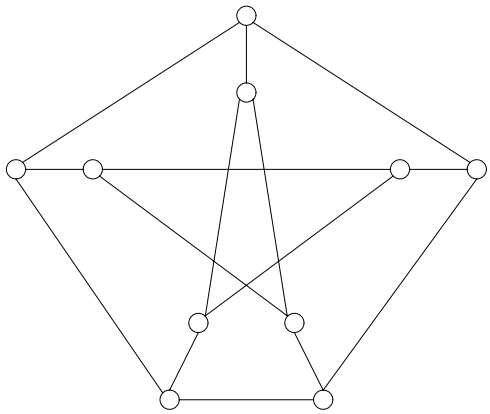


Figure 7.3.2. The Petersen Graph.

Conversely, let G be a $2r$-regular graph ($r \geq 1$). Without loss of generality we may assume G is connected, for otherwise we would simply consider each component separately. Thus, we see that G is eulerian with circuit C. Let $V(G) = \{ v_1, v_2, \ldots, v_p \}$ and define a bipartite graph

$$B = (V_1 \cup V_2, E)$$

from G as follows: Let

$$V_1 = \{ u_1, u_2, \ldots, u_p \}, \quad V_2 = \{ w_1, w_2, \ldots, w_p \} \quad \text{and}$$
$$E(B) = \{ u_i w_j \mid v_j \text{ immediately follows } v_i \text{ on } C \}.$$

The graph B is r-regular, and so by Corollary 7.1.1, B contains a perfect matching M_1. Then the graph $B - M_1$ is $r - 1$-regular and again by Corollary 7.1.1, $B - M$ contains a perfect matching M_2. Continuing in this manner, we see that $E(B)$ can be partitioned into matchings $M_1, M_2, \ldots, M_r$.

Corresponding to each matching M_k of B is a permutation π_k on the set of vertices defined by $\pi_k(v_i) = v_j$ if $u_i w_j \in E(M_k)$. We know that we can express π_k as the product of disjoint permutation cycles. Note that in this product, no permutation cycle is of length 1, for this would imply that $\pi_k(v_i) = v_i$. But this implies that $u_i w_i \in E(B)$, and, hence, that $v_i v_i$ is an edge of C, contradicting the fact G is a graph. Further note that there is no permutation cycle of length 2 in the product since this would imply that $\pi_k(v_i) = v_j$ and $\pi_k(v_j) = v_i$. But this means that $u_i w_j$ and $u_j w_i$ are edges of B and that v_j both precedes and follows v_i on C. But this contradicts the fact that C is a circuit and, hence, has no repeated edges. Thus, we are able to conclude that each permutation cycle in the product of disjoint permutation cycles representing π_k has length at least 3.

Each permutation cycle in π_k then gives rise to a cycle in G, and since the product of the permutation cycles is disjoint, the corresponding cycles span $V(G)$. But, these spanning cycles form a 2-factor of G. Further, since the matchings $M_1, M_2, \ldots, M_r$

partition the edges of G, the 2-factors that correspond to $\pi_1, \ldots, \pi_r$ are mutually edge disjoint. Thus, G is 2-factorable. □

We conclude this section by considering some special classes of graphs. The obvious starting point is the complete graphs. It turns out that we can produce very special 2-factors in K_{2p+1}. A 2-factorization of K_7 is shown in Figure 7.3.3.

Theorem 7.3.5 For every positive integer p, the graph K_{2p+1} can be 2-factored into p hamiltonian cycles.

Proof. The result is trivial when $p = 1$, so we can assume that $p \geq 2$. Let the vertices of K_{2p+1} be $v_0, \ldots, v_{2p}$. We arrange the vertices $v_1, \cdots, v_{2p}$ cyclically in a regular $2p$-gon and place v_0 in the center of the arrangement. We define the edges of the 2-factor F_i to consist of the edges v_0v_i, v_0v_{p+i} along with v_iv_{i+1} and all edges parallel to this edge, and $v_{i-1}v_{i+1}$ and all edges parallel to this edge. (All subscripts are expressed modulo $2p$). Then each F_i is a hamiltonian cycle, and K_{2p+1} is the edge sum of these 2-factors. □

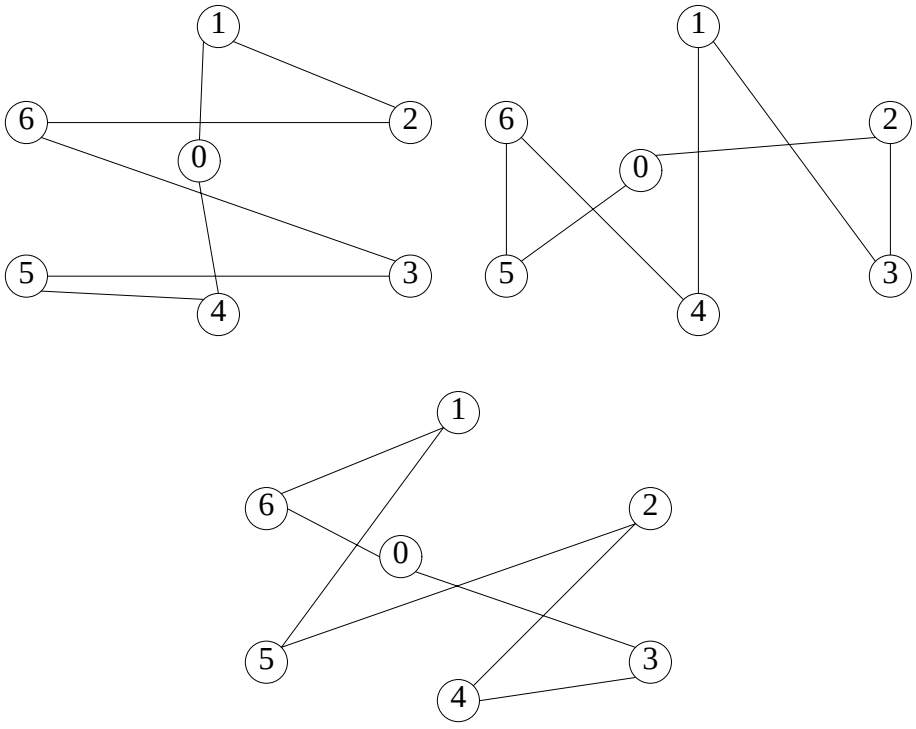


Figure 7.3.3. A 2-factorization of K_7.

Corollary 7.3.1 For every positive integer p, the graph K_{2p} can be factored into p

hamiltonian paths.

We conclude this section with another result on complete graphs. Its proof is left to the exercises.

Theorem 7.3.6 For every positive integer p, the graph K_{2p} is 1-factorable.

Section 7.4 Degrees and 2-Factors

In this section we wish to consider several results that appear similar to some of the theorems we saw earlier dealing with hamiltonian graphs. Since a hamiltonian cycle is a 2-factor, it is not surprising that there is a relationship between these hamiltonian results and theorems dealing with 2-factors. We begin with a very nice result due to El-Zahar [7].

Theorem 7.4.1 Let G be a graph of order n and let $n_1 \geq 3$ and $n_2 \geq 3$ be two integers such that $n = n_1 + n_2$. If $\delta(G) \geq \left\lceil \frac{n_1}{2} \right\rceil + \left\lceil \frac{n_2}{2} \right\rceil$, then G contains two disjoint cycles C_1 and C_2 of length n_1 and n_2, respectively.

El-Zahar's Theorem can be viewed as a generalization of Dirac's Theorem on hamiltonian graphs. Dirac's Theorem provides for a 2-factor that is one cycle while El-Zahar's Theorem uses a slightly stronger degree condition to provide for a 2-factor that is two cycles. A stronger look at Dirac's condition allows us to actually say much more. We begin with a lemma.

Lemma 7.4.1 Let G be a graph of order n with minimum degree $\delta(G) \geq n/2$. If G contains $k \geq 1$ vertex disjoint cycles $C_1, C_2, \ldots, C_k$ such that

$$\left| V(G) - \bigcup_{i=1}^{k} V(C_i) \right| \leq 2,$$

then G has a 2-factor with exactly k vertex disjoint cycles.

Proof. If $V(G) - \bigcup_{i=1}^{k} V(C_i) = \{ w \}$, then G contains the desired 2-factor since *deg* $w \geq n/2$ and hence w is adjacent to two consecutive vertices of a least one of the cycles.

Thus we may assume $V(G) - \bigcup_{i=1}^{k} V(C_i) = \{ u, v \}$. If one of u and v, say u, is adjacent to two consecutive vertices of one of the cycles, then, as before, we obtain the

desired 2-factor. Thus we may assume that $deg\ u = deg\ v = n/2$ and that each of u and v is adjacent to alternate vertices of each of the cycles and necessarily to each other. Let

$$C_1 : u_1, u_2, \ldots, u_t, u_1$$

be one such cycle. If u and v are adjacent to the same set of vertices of C_1, say $\{u_1, u_3, \ldots, u_{t-1}\}$. Then C_1 can be replaced by

$$u_1, u, v, u_3, u_4, \ldots, u_t, u_1$$

to obtain k vertex disjoint cycles containing all but one vertex of G. In this case, as we have seen, G has the desired 2-factor. On the other hand, if u is adjacent to $u_1, u_3, \ldots, u_{t-1}$ and v is adjacent to $u_2, u_4, \ldots, u_t$. Then we may replace C_1 with the cycle

$$u_1, u, u_3, u_2, v, u_4, u_5, \ldots, u_t, u_1$$

to complete the proof. □

Now, with the aid of the lemma, we can take the stronger look at Dirac's condition promised earlier. The result is from [5].

Theorem 7.4.2 Let k be a positive integer and let G be a graph of order $n \geq 4k$ with minimum degree $\delta(G) \geq n/2$. Then G has a 2-factor with exactly k vertex disjoint cycles.

Proof. The cases $k = 1, 2$ follow from Dirac's Theorem and El-Zahar's Theorem, respectively. Thus we may assume that $k > 2$. Since $\delta(G) \geq n/2 \geq 2k$ and $n \geq 4k$, G contains k vertex disjoint cycles $C_1, C_2, \ldots, C_k$ by Theorem 5.8.4. Let $X = V(G) - \bigcup_{i=1}^{k} V(C_i)$ and assume $X \neq \emptyset$.

If $\delta(\langle X \rangle) < |X|/2$, let $w \in X$ with $deg_{\langle X \rangle}(w) < |X|/2$. Then, since $\delta(G) \geq n/2$, it follows that w is adjacent to more than half of the vertices of some C_i, $1 \leq i \leq k$ and therefore adjacent to consecutive vertices of C_i. Therefore w can be added to C_i. Continue this process to obtain k vertex disjoint cycles $C'_1, C'_2, \ldots, C'_k$ such that either

$$V(G) = V(C'_1) \cup_k V(C'_2) \cup \cdots \cup V(C'_k) \quad \text{or}$$
$$X' = V(G) - \bigcup_{i=1} V(C'_i) \neq \emptyset \quad \text{and}$$
$$\delta(\langle X' \rangle) \geq |X'|/2.$$

In the first case we have the desired 2-factor. In the second case, either $\langle X' \rangle = K_2$ or $\langle X' \rangle$ is hamiltonian. If $\langle X' \rangle = K_2$, then by applying Lemma 7.4.1 we obtain the desired 2-factor. Thus we may assume that C'_{k+1} is a hamiltonian cycle of X'. Without loss of generality, assume that $|V(C'_1)| \leq |V(C'_i)|$ for $i = 2, 3, \ldots, k+1$, so that

$|V(C'_1)| \leq \frac{n}{k+1} \leq n/4.$

Since $\delta(G) \geq n/2$, the number of edges between $V(C'_1)$ and $V(G) - V(C'_1)$ is at least

$$|V(C'_1)|\,(n/2 - |V(C'_1)| + 1).$$

If between every three consecutive vertices of C'_1 and of C'_i $(2 \leq i \leq k+1)$ there are at most three edges, then the number of edges between $V(C'_1)$ and $V(G) - V(C'_1)$ is at most

$$(\frac{1}{3})\,(n - |V(C'_1)|)\,|V(C'_1)|.$$

This, however, implies that

$$(\frac{1}{3})\,(n - |V(C'_1)|)\,|V(C'_1)| \geq |V(C'_1)|\,(n/2 - |V(C'_1)| + 1),$$

so that

$$|V(C'_1)| \geq n/4 + 3/2,$$

contradicting the fact $|V(C'_1)| \leq n/4$. Thus, for some i with $2 \leq i \leq k+1$, three consecutive vertices of C'_1 have at least four adjacencies to three consecutive vertices of C'_i. In this case it is straightforward to verify that C'_1 and C'_i can be combined to form a cycle containing all but at most two of the vertices of C'_1 and C'_i. Then an application of Lemma 7.4.1 completes the proof. □

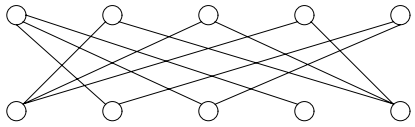
With slightly more effort it is possible to extend our generalizations to an Ore-like result concerning degree sums of nonadjacent vertices. The following is also from [5].

Theorem 7.4.3 Let G be a graph of order $n \geq 4k$ such that $deg\ x + deg\ y \geq n$ for each pair of nonadjacent vertices x, y in $V(G)$, then G has a 2-factor with exactly k vertex disjoint cycles.

Exercises

1. Show that the n-cube Q_n $(n \geq 2)$ has a perfect matching.
2. Show that Q_n is r-factorable if, and only if, $r \mid n$.
3. Characterize when the graph $K_{p_1, p_2, \ldots, p_n}$ has a perfect matching.
4. Determine the number of perfect matchings in the graphs $K_{p,p}$ and K_{2p}.

5. How many perfect matchings can exist in a tree?
6. Find a maximum matching and a minimum cover in the graph below using each of the indicated methods.
   a. Algorithm 7.2.1 and Theorem 7.2.1.
   b. A network model.



7. Use Dirac's theorem (Corollary 5.2.1) to show that if G has even order p and $\delta(G) \geq \frac{p}{2} + 1$, then G has a 3-factor.
8. Show that every doubly stochastic matrix is a square matrix.
9. Show that if $G = (X \cup Y, E)$ is a bipartite graph, then
$$\beta_1(G) = |X| - \max_{S \subseteq X} \{|S| - |N(S)|\}.$$
10. Use the previous exercise to show that if the (p, q) graph $G = (X \cup Y, E)$ is bipartite and $|X| = |Y| = n$ and $q > (k - 1)n$, then G has a matching of cardinality k.
11. [9] Suppose that G is a graph of order p with the property that for every pair of nonadjacent vertices x and y, $|N(x) \cup N(y)| \geq s$.
    a. Use Berge's defect form of Tutte's theorem to show that if $s > 2\left\lfloor \frac{p}{3} \right\rfloor - 2$ and p is odd and $p \geq 6$, then
$$\beta_1(G) = \frac{1}{2}(p - 1).$$
    b. Find a graph of order 5 for which the conditions of part (a) fail to ensure $\beta_1(G) = \frac{1}{2}(p - 1)$.
    c. Use Tutte's theorem to show that if $s > \frac{2}{3}(p - 1) - 1$ and p is even and G is connected, then $\beta_1(G) = \frac{p}{2}$.
12. Use Tutte's theorem to prove Hall's theorem.
13. Use König's theorem to prove Hall's theorem.

14. Prove Corollary 7.3.1.

15. Prove Theorem 7.3.6.

16. Can K_{2n} can be factored into $n - 1$ hamiltonian paths and one 1-factor?

17. Let G be a (p, q) graph of even order p with $\delta(G) < \frac{p}{2}$. Show that if

$$q > \binom{\delta(G)}{2} + \binom{p - 2\delta(G) - 1}{2} + \delta(G)(p - \delta(G)),$$

then G has a perfect matching.

18. Four men and four women apply to a computer dating service. The computer evaluates the unsuitability of each man for each woman as a percentage (see the table below). Find the best possible dates for each woman for this Friday night.

	M_1	M_2	M_3	M_4
W_1	60	35	30	65
W_2	30	10	55	30
W_3	40	60	15	35
W_4	25	15	40	40

19. Consider the table used for the last exercise as representing the weights assigned to a bipartite graph and solve the bottleneck assignment problem for this graph.

20. The math department at your college has six professors that must be assigned to teach each of five different classes. The department did an examination of the suitability of each professor for each class and the unsuitability table is shown below. What is the optimal teaching assignment that can be made if no professor is assigned more than one class?

	P_1	P_2	P_3	P_4	P_5	P_6
C_1	75	25	55	25	50	35
C_2	60	30	45	35	45	20
C_3	55	25	50	15	50	30
C_4	40	35	40	45	35	25
C_5	50	20	45	30	40	45

(Hint: Add a dummy class that each professor is equally suited to teach.)

21. Does the previous problem make sense as a bottleneck assignment problem? If so, solve it.

22. Consider the doubly stochastic matrix below. Use Algorithm 7.2.2 to decompose this matrix into permutation matrices.

$$\begin{vmatrix} 0.3 & 0.3 & 0.0 & 0.3 & 0.1 \\ 0.1 & 0.5 & 0.2 & 0.1 & 0.1 \\ 0.2 & 0.0 & 0.3 & 0.5 & 0.0 \\ 0.0 & 0.2 & 0.5 & 0.0 & 0.3 \\ 0.4 & 0.0 & 0.0 & 0.1 & 0.5 \end{vmatrix}$$

23. Consider the table of the previous problem as the weights assigned to the edges of a bipartite graph. Interpret your solution in relation to the last problem on this graph.

24. Explain why the adjustment process allows us to complete the hungarian algorithm applied to an unsuitability matrix.

25. A *decomposition* of G is a collection $\{ H_i \}$ of subgraphs of G such that $H_i = < E_i >$ for some subset E_i of $E(G)$ and where the sets $\{ E_i \}$ partition $E(G)$. Prove that the complete graph K_p can be decomposed as a collection of 3-cycles if, and only if, $p \geq 3$, p is odd and 3 divides $\binom{n}{2}$.

26. Find a decomposition of K_5 as 5-cycles.

27. Find a decomposition of K_{10} as paths of length 5.

28. Prove that for each integer $n \geq 1$, the graph K_{2n+1} can be decomposed as a collection of stars $K_{1,\,n}$ and that the graph K_{2n} can be decomposed as a collection of stars $K_{1,\,n}$.

29. By an ascending subgraph decomposition of a graph G of size $\binom{n+1}{2}$ we mean an edge decomposition into subgraphs $G_1, \ldots, G_n$ with the properties that $|E(G_i)| = i$ and $G_1 \leq G_2 \leq \cdots \leq G_n$, that is, each G_i is isomorphic to a subgraph of G_{i+1}. Show that K_m has an ascending subgraph decomposition as stars and also an ascending subgraph decomposition as paths.

30. Find an example that shows that the $n \geq 4k$ condition in Theorem 7.4.2 cannot be reduced.

31. Find an example to show that the degree condition in El-Zahar's Theorem is sharp.

32. Prove Theorem 7.4.1.

References

1. Anderson, I., Perfect Matchings of a Graph. *J. Combinatorial Theory,* 10B(1971), 183 – 186.
2. Berge, C., Two Theorems in Graph Theory. *Proc. Nat. Acad. Sci. USA,* 43(1957), 842 – 844.
3. Berge, C., Sur le Couplage Maximum d'un Graphe. *C. R. Acad. Sci. Paris,* 247(1958), 258 – 259.
4. Birkhoff, G., Tres Observaciones Sobre el Algebra Lineal. *Universidad Nacionale Tucumáń Revista,* 5(1946), 147 – 151.
5. Brandt, S., Chen, G., Faudree, R.J., Gould, R.J., and Lesniak, L., On the Number of Cycles in a 2-Factor, (preprint).
6. Egerváry, J., Matrixok Kombinatorikus Tulajdonságairól. *Mathematika eś Fizikai Laṕok,* 38(1931), 16 – 28.
7. El-Zahar, M.H., On Circuits in Graphs, *Discrete Math.* 50(1984d), 227-230.
8. Faudree, R. J., Gould, R. J., Jacobson, M. S., and Schelp R. H., Extremal Problems Involving Neighborhood Unions, *J. Graph Theory*, (in press).
9. Ford, L. R., Jr. and Fulkerson, D. R., *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.
10. Gale, D. and Shapley, L.S., College Admissions and the Stability of Marriage, *The American Mathematical Monthly*, 69(1962), 9-14.
11. Hall, P., On Representatives of Subsets, *J. London Math. Soc.,* 10(1935), 26 – 30.
12. König, D., Graphs and Matrices (Hungarian). *Mat. Fig. Lapok,* 38(1931), 116 – 119.
13. Petersen, J., Die Theorie der Regulären Graphs. *Acta Math.,* 15(1891), 193 – 220.
14. Tutte, W. T., The Factorization of Linear Graphs. *J. London Math. Soc.,* 22(1947), 107 – 111.
15. Von Neumann, J., A Certain Zero-Sum Game Equivalent to the Optimal Assignment Problem. Contributions to the Theory of Games, Kuhn and Tucker, eds., *Annals of Mathematics Studies,* 28(1953), 206 – 212.

Chapter 8
Independence

Section 8.1 Vertex Independence and Coverings

Next, we consider a problem that strikes close to home for us all, final exams. At the end of each term, students are required to take final exams in each of their classes. Each exam is to be given once during some specified period, and the time allowed for each exam (no matter what the class) is the same. The question of interest is: What is the minimum number of examination periods needed to ensure there are no conflicts, that is, that no student has two exams during the same period. Of course, as you well know, this is a completely fictional problem since no school has ever tried to determine this number.

As usual, we desire a graph model for this problem. Thus, we seek a graph $G = (V, E)$ where each vertex of V represents an examination and $xy \in E$ if, and only if, there is some student that must take both examination x and examination y. Two examinations can be scheduled in the same period only if there is no edge between the corresponding vertices in our model. Thus, we seek sets of mutually nonadjacent vertices in G; that is, we seek independent sets of vertices. A solution to our problem is a partitioning of V into sets of mutually independent vertices where the number of such sets is a minimum. Vertices in the same set of this partition represent exams that can be scheduled during the same period without conflict. Thus, it is clear that we need to study independence if we are to solve this scheduling problem.

There are several ideas that are related to vertex independence that will be helpful. We have already studied one such idea, namely matchings. In our study, we sought independent sets of edges; now we seek independent sets of vertices. Another useful and related notion was also introduced in Chapter 7, the idea of a covering. In Chapter 7 we also saw that there is a relation between independent edges and coverings. In this section we wish to show a similar relationship between independent vertices and coverings. It is easy to see that given any independent set I in V, the vertices of $V - I$ form a covering of G. Conversely, if $V - I$ forms a covering, then $< I >$ must be empty; hence, I must be independent. Thus, we have shown the following useful result.

Proposition 8.1.1 In a graph $G = (V, E)$, a subset I of V is independent if, and only if, $V - I$ is a covering of G.

An independent set in a graph G is called a *maximum* independent set provided no other independent set in G has larger cardinality; it is called *maximal* if it is contained in no larger independent set. Recall that the number of vertices in a maximum independent set in G is called the *independence number* of G and is denoted $\beta(G)$. Analogously, the number of vertices in a minimum covering of a graph G is called the *covering number* of

G and is denoted by $\alpha(G)$. Natural analogs of the independence number and covering number also exist for edges. The *edge independence number*, denoted $\beta_1(G)$, is the size of a maximum matching in G, and the *edge covering number*, denoted $\alpha_1(G)$, is the minimum size of a set L of edges with the property that every vertex is an end vertex of some edge in L.

In Chapter 7, we proved a result from König and Egerváry (Theorem 7.1.3) that showed that in a bipartite graph, $\beta_1(G) = \alpha(G)$. Our next result, from Gallai [11], shows several other relations among these parameters.

Theorem 8.1.1 If G is a graph of order p with $\delta(G) > 0$, then

$$\alpha(G) + \beta(G) = p \quad \text{and}$$
$$\alpha_1(G) + \beta_1(G) = p.$$

Proof. In order to establish the first equality, let I be an independent set of vertices in G with $|I| = \beta(G)$. Since I is independent, $V - I$ is a cover of G. Therefore,

$$\alpha(G) \leq |V - I| \leq p - \beta(G).$$

If C is a set of vertices that covers $E(G)$ and $|C| = \alpha(G)$, then $V - C$ is an independent set of vertices, and so

$$\beta(G) \geq |V - C| \geq p - \alpha(G).$$

But these two inequalities together show that $\alpha(G) + \beta(G) = p$.

The proof of the second equality is left as an exercise. □

With the aid of the last result, we can now prove a theorem that looks very much like the König-Egerváry theorem.

Theorem 8.1.2 If G is a bipartite graph with $\delta(G) > 0$, then $\beta(G) = \alpha_1(G)$.

Proof. Let G be a bipartite graph with $\delta(G) > 0$. By Gallai's theorem (Theorem 8.1.1),

$$\alpha(G) + \beta(G) = \alpha_1(G) + \beta_1(G).$$

However, by the König-Egerváry theorem, $\beta_1(G) = \alpha(G)$; thus, $\beta(G) = \alpha_1(G)$. □

Section 8.2 Vertex Colorings

Recall that we wish to partition the vertices of a graph into independent sets in such a way that we minimize the number of these sets. This problem is usually described in a more visual manner. We say that an assignment of colors to the vertices of a graph G (one color per vertex) so that adjacent vertices are assigned different colors is a (legal) *coloring* of G. Colorings always exist since we can assign each vertex a different color if necessary. In a given coloring of a graph G, the set of all those vertices assigned the same color is called a *color class.* Clearly, a coloring of G produces a partition of $V(G)$ into color classes, and each of the color classes is an independent set of vertices. A coloring that uses n colors is called an *n-coloring,* and a graph whose vertices can be colored with n or fewer colors is called *n-colorable.* The minimum number of colors in a coloring of G, where the minimum is taken over all colorings of G, is called the *chromatic number* of G and is denoted by $\chi(G)$. If G is a graph for which $\chi(G) = n$, then we say G is *n –chromatic.*

Note that in most cases, there is nothing unique about colorings. By this we mean that color classes may easily vary according to the particular coloring at hand. For example, suppose we consider the two colorings of C_7 shown in Figure 8.2.1. We obtain very different color classes for each of these colorings. The first coloring has color classes

$$\{ v_1, v_3, v_7 \} \quad , \quad \{ v_2, v_4, v_6 \} \quad \text{and} \quad \{ v_5 \}.$$

The second coloring yields color classes of

$$\{ v_1, v_4 \}, \quad \{ v_2, v_5, v_7 \} \quad \text{and} \quad \{ v_3, v_6 \}.$$

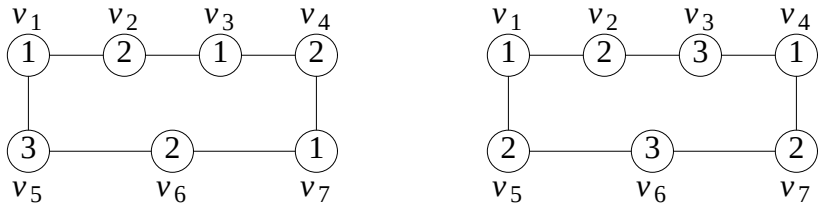


Figure 8.2.1. Different color classes formed from different colorings.

For several well-known classes of graphs, chromatic numbers are readily determined. For example,

$$\chi(C_{2p}) = 2, \quad \chi(C_{2p+1}) = 3, \quad \text{and} \quad \chi(K_p) = p.$$

Further, it is also easy to show that $\chi(K_{p_1, p_2, \ldots, p_n}) = n$. In general, if G is a k-partite graph, then $\chi(G) \leq k$. Thus, if G is a 2-chromatic graph, then G must be bipartite.

Despite the fact that a great deal of effort has gone into the study of colorings, there is no known general formula or method for finding the chromatic number of a graph. Thus, we can only hope for general bounds or formulas in certain special cases. In the study of graph colorings, certain types of graphs are often helpful. A graph G is *critically n-chromatic* or simply, *n-critical,* (if the context of coloring is clear), if $\chi(G) = n$ and $\chi(G - x) = n - 1$ for every vertex x in G. Similarly, we say G is *minimally n-chromatic,* or simply *n-minimal,* if $\chi(G) = n$ and $\chi(G - e) = n - 1$ for every edge e in G. Dirac [7] began the investigation of critically n-chromatic graphs. Several of his results are structural in nature and will be useful in our study of chromatic numbers of graphs. Our goal is to use this structural information to establish the sharpest possible general upper bound on the chromatic number of a graph. We begin with a look at degrees in critical graphs.

Theorem 8.2.1 If G is a critically n-chromatic graph, then $\delta(G) \geq n - 1$.

Proof. Suppose that this is not the case; that is, let G be a critically n-chromatic graph with $\delta(G) < n - 1$. We will produce a coloring of G using fewer than n colors. To do this, let v be a vertex of degree $\delta(G)$. Since G is critically n-chromatic, $G - v$ is $(n - 1)$-colorable. Color the vertices of $G - v$ with $n - 1$ colors, and let $V_1, V_2, \ldots, V_{n-1}$ be the corresponding color classes. Since $\deg v = \delta(G) < n - 1$, there must exist a color class V_i with the property that v is nonadjacent with every vertex in V_i. Thus, v can be assigned color i, producing an $n - 1$ coloring of G and the desired contradiction. □

There are several useful facts that follow directly from Theorem 8.2.1 These are stated next, with the proofs left for the exercises.

Corollary 8.2.1

1. Every n-chromatic graph has at least n vertices of degree at least $n - 1$.
2. For any graph G, $\chi(G) \leq \Delta(G) + 1$.

We have established an upper bound on $\chi(G)$, but with a little more effort, we can improve upon this bound. Thus, we continue our investigation of the structure of critically n-chromatic graphs by examining the nature of vertex cut sets in critical graphs. In order to do this, we adopt the following notation. Let S be a vertex cut set in a connected graph G. Let the components of $G - S$ have vertex sets $V_1, V_2, \ldots, V_t$. Then the subgraphs $G_i = \langle V_i \cup S \rangle$ are called the *S-components* of G. Moreover, we say that colorings of $G_1, G_2, \ldots, G_t$ *agree on* S if each vertex of S is assigned the same color in each of the colorings of the G_i $(i = 1, \ldots, t)$.

Theorem 8.2.2 If G is a critically n-chromatic graph ($n \geq 4$), then no vertex cut set

induces a complete graph and, hence, G must be 2-connected.

Proof. We proceed by contradiction. Let G be a critically n-chromatic graph and suppose that G has a vertex cut set S that induces a complete graph. Denote the S-components of G by $G_1, G_2, \ldots, G_k$. Since G is critically n-chromatic, each G_i is $(n-1)$-colorable. Furthermore, since $< S >$ is complete, the vertices of S must receive different colors in any $(n-1)$-coloring of some G_i. By permuting colors in $G_2, \ldots, G_k$ we see that there are $(n-1)$-colorings of $G_1, G_2, \ldots, G_k$ that agree on S. These colorings together produce an $(n-1)$-coloring of G, which contradicts the fact that G is n-critical. Therefore, no vertex cut set of G induces a complete graph.

Now, suppose that $\{ v \}$ is a vertex cut set of G. Then clearly $< v >$ is a complete graph. But we just showed that no critical graph could have a cut set that induced a complete graph and, hence, $\{ v \}$ cannot be a cut set. That is, G must be 2-connected. □

We continue the investigation of critical graphs with a look at their edge connectivity. The next result is again from the work of Dirac [7].

Theorem 8.2.3 Every critically n-chromatic graph $(n \geq 2)$ is $(n-1)$-edge connected.

Proof. Suppose that G is a critically n-chromatic graph $(n \geq 2)$. If $n = 2$, then G is K_2, while if $n = 3$, then G is an odd cycle. Thus, G is $1-$edge or $2-$edge connected, respectively.

Now, we assume that $n \geq 4$ and that G is not $(n-1)$-edge connected. Thus, there must exist a partition of $V(G)$ into subsets W_1 and W_2 such that there are fewer than $n-1$ edges joining W_1 and W_2. Call the set of these edges E_W. Since G is critically n-chromatic, we know that $< W_1 >$ and $< W_2 >$ are both $(n-1)$-colorable. Let them both be colored with $(n-1)$ colors. If the edges in E_W are all incident to vertices assigned different colors, then we have an $(n-1)$-coloring of G, a contradiction. Suppose this does not happen. Our strategy now is to permute colors so that the edges of E_W do have end vertices assigned different colors, producing a contradiction.

Let $V_1, \ldots, V_k$ be the color classes of $< W_1 >$ with at least one edge to $< W_2 >$. Further suppose that there are q_i edges from V_i to $< W_2 >$. Thus, we see from our assumptions that $\sum_{i=1}^{k} q_i \leq n-2$.

We now try to permute colors to obtain the desired coloring. If each vertex $v_1 \in V_1$ is adjacent only with vertices of W_2 with different colors, then we do nothing. If however, there is some v_1 that is adjacent to some vertex of W_2 of the same color, then in $< W_1 >$ we permute the colors so that no vertex of V_1 is adjacent to a vertex of W_2 having the same color. This is possible since the vertices of V_1 may be assigned any one of at least $n-1-q_1$ (> 0) colors.

Now, with this new coloring, if each vertex $v_2 \in V_2$ is adjacent only to vertices in W_2 assigned different colors, then again we do nothing. But, if some vertex $v_2 \in V_2$ is adjacent to a vertex of W_2 assigned the same color, then in W_1 we again permute the $n - 1$ colors, leaving the color assigned to V_1 fixed, until no vertex in $V_1 \cup V_2$ is adjacent to a vertex in W_2 having the same color. This is possible since the vertices of V_2 can be assigned any of $(n - 1) - (q_2 + q_{1)}$ colors and this value is greater than zero. Continuing this process, we arrive at an $(n - 1)$-coloring of G and the desired contradiction. □

Analogous to the idea of a critical graph is the edge concept of a minimally n-chromatic graph. Since every connected, minimally n-chromatic graph is critically n-chromatic, there is a strong relationship between these two ideas. This relationship will aid us in our study of the structure of critical graphs. Theorems 8.2.3 and 8.2.1 have the following immediate Corollary.

Corollary 8.2.2

1. If G is a connected, n-minimal graph $(n \geq 2)$, then G is $(n - 1)$-edge connected.
2. If G is n-critical or connected and n-minimal, then $\delta(G) \geq n - 1$.

We are now ready to complete our study of the structure of critical and minimal graphs. If an n –critical graph G has a two vertex cut set $\{ u, v \}$, then we know u and v cannot be adjacent. We say that an $S = \{ u, v \}$-component H of G is *color-unique* if every $(n - 1)$-coloring of H assigns the same color to both u and v and that it is *color-distinct* if every $(n - 1)$-coloring of H assigns different colors to u and v. The following result is again from Dirac [8].

Theorem 8.2.4 Let G be an n-critical graph with a two vertex cut set $S = \{ u, v \}$. Then:

1. $G = H_1 \cup H_2$, where H_1 is a color-unique S-component and H_2 is a color-distinct S-component.
2. Both $H_1 + uv$ and the graph obtained from H_2 by identifying u and v are n-critical.

Proof. Let G be an n-critical graph with a two vertex cut set $S = \{ u, v \}$. Then, since G is critical, each S-component of G is $(n - 1)$-colorable. There exists no $(n - 1)$-colorings of these S-components which all agree on S or else there would be an $(n - 1)$-coloring of G. Thus, there are two S-components, say H_1 and H_2, such that no $(n - 1)$-coloring of H_1 agrees with any $(n - 1)$-coloring of H_2. Clearly, then, one component must be color-unique and the other color-distinct. Further, if there were more S-components, then some two would agree, and then deleting one of these would

contradict the fact G is n-critical. Without loss of generality, say H_1 is color-unique. Since H_1 and H_2 are of different types, the subgraph $H_1 \cup H_2$ is not $(n-1)$-colorable. Thus, since G is n-critical, we must have that $G = H_1 \cup H_2$.

A proof of (2) is left to the exercises. □

Corollary 8.2.3 Let G be an n-critical graph with a two vertex cut set $\{ u, v \}$. Then

$$deg\ u + deg\ v \geq 3n - 5.$$

The proof of the corollary is also left to the exercises.

We conclude this section with the most fundamental result dealing with vertex colorings. This theorem is from Brooks [5], and it provides the general upper bound on the chromatic number we have been seeking.

Theorem 8.2.5 If G is a connected graph that is neither an odd cycle nor a complete graph, then $\chi(G) \leq \Delta(G)$.

Proof. Let G be a connected n-chromatic graph which is neither an odd cycle nor a complete graph. Without loss of generality, we may assume that G is n-critical. By Theorem 8.2.2, G is 2−connected. Further, since 1−critical and 2−critical graphs are complete and 3−critical graphs are odd cycles (see the exercises), we must have that $n \geq 4$.

If G has a 2-vertex cut set, say $\{ u, v \}$, then Corollary 8.2.3 implies that

$$2\Delta(G) \geq deg\ u + deg\ v \geq 3n - 5 \geq 2n - 1.$$

But this implies that $\chi(G) = n \leq \Delta(G)$ since since $2\Delta(G)$ is even.

Now assume that G is a 3−connected graph. Since G is not complete, there are three vertices, u, v, and w, in G such that both uv and vw are edges of G but uw is not an edge of G. Let $u = v_1$ and $w = v_2$ and let $v_3, \ldots, v_p = v$ be an ordering of the vertices of $G - \{ u, w \}$ with the property that each v_i is adjacent to some v_j where $j > i$. This ordering can be accomplished by arranging the vertices in nonincreasing order of their distances from v in $G - \{ u, w \}$.

We wish to color the vertices of G using at most $\Delta(G)$ colors. To do this, assign both v_1 and v_2 color 1. Then, successively color $v_3, v_4, \ldots, v_p$ with the smallest available color in the numerically ordered colors $1, 2, \ldots, \Delta(G)$. From our construction of the ordering on the vertices, each vertex v_i $(i \geq 3)$, except v_p, is adjacent to some vertex v_j where $j > i$. Therefore, v_i is adjacent to at most $\Delta(G) - 1$ vertices preceding it in the vertex ordering. Thus, when v_i is to be colored (in its turn according to our listing), there will be a color available to assign to it. Also, since v_p is adjacent to two vertices colored 1, then it will also have a color available when it is time for it to be colored. Thus, we

have colored the vertices of G with at most $\Delta(G)$ colors, completing the proof. □

Can you find examples of graphs for which the bound in Brooks's theorem is sharp? Can you find examples for which the bound from Brooks's theorem is arbitrarily bad, that is, for which the difference between the actual chromatic number and the bound grows larger as the order grows larger?

Section 8.3 Approximate Coloring Algorithms

The general question of determining the chromatic number of a graph is another NP-complete problem (see [12]). Some exhaustive search algorithms have been developed (for example, see [6] or [4]). These algorithms will find the chromatic number of very small graphs effectively. However, in most practical cases, we are interested in finding the chromatic number of large graphs, certainly of order 100 or more and often of order 1000 or more. Thus, we again turn to a variety of heuristics to help us develop approximation algorithms providing good bounds.

The first (and usually worst) heuristic one thinks of involves the greedy approach. The idea is simply to color the vertices, one by one, as they are encountered using any available color, that is, any color not already assigned to a neighboring vertex. Unfortunately, as you might imagine, this algorithm can be very bad. In fact, this approach can provide a very poor bound on the chromatic number.

Manvel [17] described several heuristics typical of those motivating most approximation algorithms. We state these ideas for later reference.

1. A vertex of high degree is harder to color than a vertex of low degree.
2. Vertices with the same neighborhood should be colored alike.
3. Coloring many vertices with the same color is a good idea.

The algorithms we are about to examine are motivated by some combination of these ideas. We can also classify them as falling into two fundamental categories. The first group of algorithms can be thought of as being in the category of sequentially based methods, where the order in which the vertices will be colored is decided before we begin to color them. This technique is essentially the technique we used in the proof of Brooks's theorem. Given any ordering of the vertices of a graph, sequential coloring algorithms usually try to assign the minimum color possible to the next vertex. That is, if we are to color vertex v, then having ordered the colors numerically, we assign v the smallest color according to this ordering that does not appear in $N(V)$. We formally state the generic sequential coloring algorithm next.

Algorithm 8.3.1 Generic Sequential Coloring Algorithm.

Input: Any ordering of the vertices of a graph G.
Output: A coloring of the vertices.
Method: Use the minimum available color.

1. Assign color 1 to vertex v_1.
2. If $H_{i-1} = < v_1, \ldots, v_{i-1} >$ has been colored with j colors, then assign v_i color k, where $k \leq j + 1$ is the minimum available color (according to some numerical ordering of the colors, say $1, 2, \ldots, n$).

Heuristic 1 is the prime motivation for the *largest first* heuristic, which orders the vertices in descending order based on their degrees (hence, it is a sequential coloring algorithm). The vertex of highest degree is colored first, the vertex of next highest degree second, and so forth in a greedy manner. In each case, the color selected for the vertex is the smallest possible legal color. Thus, a fast and simple method for coloring vertices is available. In fact, the largest first heuristic generally provides a reasonable bound on the chromatic number of small-order graphs.

Using the largest first heuristic and the sequential coloring algorithm, Welsh and Powell [22] obtained the following max-min result.

Theorem 8.3.1 Let G be a graph with $V(G) = \{ v_1, \ldots, v_n \}$ and where $deg\ v_i \geq deg\ v_{i+1}$ for $i = 1, \ldots, n - 1$. Then

$$\chi(G) \leq \max_i \min \{ i, deg\ v_i + 1 \}.$$

Many variations on heuristic 1 are also possible. A reversal in strategy provides us with a somewhat more effective use of the same basic idea. This is known as the *smallest last* algorithm of Matula, Marble and Isaacson [18]. In this algorithm we again want to determine an ordering of the vertices, this time based on recursively examining the vertices of smallest degree and removing them from the graph. The order of removal of the vertices is the reverse of the order in which they are to be colored. Thus, we select the vertex of lowest degree in the graph and remove it from the graph, effectively placing it as the last vertex on the list to be colored. In the subgraph that remains, we repeat the process, again selecting the vertex of smallest degree and removing it (hence, it will be the next to last vertex colored). We continue in this manner until all vertices have been ordered. Next, we again sequentially color this listing in a greedy manner. You should note that this is not necessarily the same as selecting the vertices of minimum degree in the graph itself; hence, this is potentially a different ordering from the largest first ordering.

The fundamental difference between the smallest last and largest first heuristics is their views of the graph. The ordering in the smallest last method is based on the degrees in the subgraphs obtained by removing the vertices of smallest degree, rather than simply on the degrees in the graph itself. Thus, the ordering of vertices obtained via each of

these methods may well be different. Tests by Brelaz [4] show that, typically, the smallest last method is somewhat better than the largest first method.

Example 8.3.1. Suppose we apply each of our heuristics to the graph of Figure 8.3.1.

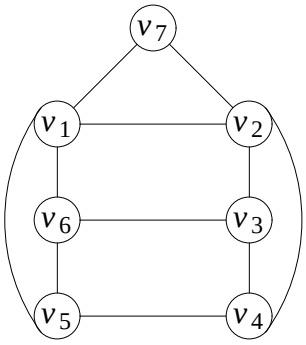


Figure 8.3.1. A graph to test coloring heuristics.

Given the arbitrary ordering of $v_1, v_2, v_4, v_3, v_5, v_6, v_7$, the generic sequential coloring algorithm provides the following color assignments:

$$v_1 \leftarrow 1, \quad v_2 \leftarrow 2, \quad v_4 \leftarrow 1,$$
$$v_3 \leftarrow 3, \quad v_5 \leftarrow 2, \quad v_6 \leftarrow 4, \quad v_7 \leftarrow 3.$$

Hence, this arbitrary ordering provides a bound of 4 on the chromatic number of the graph.

If we use the largest first ordering of $v_1, v_2, v_3, v_6, v_5, v_4, v_7$, we obtain the following coloring:

$$v_1 \leftarrow 1, \quad v_2 \leftarrow 2, \quad v_3 \leftarrow 1, \quad v_6 \leftarrow 2,$$
$$v_5 \leftarrow 3, \quad v_4 \leftarrow 4, \quad v_7 \leftarrow 3.$$

Thus, we again obtain a bound of 4 on the chromatic number of the graph.

Finally, a smallest last ordering of $v_1, v_5, v_6, v_4, v_3, v_2, v_7$ provides the following coloring:

$$v_1 \leftarrow 1, \quad v_5 \leftarrow 2, \quad v_6 \leftarrow 3, \quad v_4 \leftarrow 1,$$
$$v_3 \leftarrow 2, \quad v_2 \leftarrow 3, \quad v_7 \leftarrow 2.$$

Hence, this ordering provides a bound of 3, and since the graph contains a K_3, we see this is the value of the chromatic number. □

Note that the sequential ordering used in each of the above algorithms need not be unique for the particular graph and that a different ordering used in the same algorithm can provide a different value. Can you find a legal largest first ordering that also provides a bound of 3 for the chromatic number of this graph?

In each of the previous examples, we have implicitly broken ties by a random selection of vertices, since the only information we were using was the degree of the vertex itself. Certainly, more involved heuristic tests can be applied. For example, suppose we try a two-step approach to constructing our vertex ordering. That is, let's include a bit more information than merely the degrees of the vertices. Suppose that we include the sum of the degrees of all neighbors of a vertex as well. The idea is that a vertex of high degree whose neighbors together also have high degree sum would possibly present us with problems later. Certainly, you can devise other such tests for deciding on the vertex ordering and any of these tests can also be used to break ties.

One of the best known two-step heuristics for fairly small graphs (up to about 100 vertices) is from Brelaz [4]. It is motivated by a combination of heuristics 1 and 2. Define the *color-degree* of a vertex v to be the number of colors used to color the vertices adjacent to v. A sequential order of the vertices is then decided primarily by the color-degree, with ties being broken by selecting the vertex with largest degree in the uncolored subgraph.

Algorithm 8.3.2 Brelaz Color-Degree Algorithm.
Input: A graph G.
Output: An approximate coloring of the vertices of G.
Method: Break ties based on the smallest color-degree.

1. Order the vertices in decreasing order of degrees.
2. Color a vertex of largest degree with color 1.
3. Select a vertex with maximum color-degree. If there is a tie, choose any of these vertices of largest degree in the uncolored subgraph.
4. Color the vertex selected in step 3 with the least possible color.
5. If all vertices are colored,
   then stop;
   else go to step 3.

We can prove that there is an instance when this algorithm provides the chromatic number exactly.

Theorem 8.3.2 If G is a 2-connected bipartite graph of order at least 3, then the coloring obtained from Algorithm 8.3.2 determines the chromatic number for G.

Proof. Let G be a 2-connected bipartite graph of order at least 3 and suppose that G has been colored by Algorithm 8.3.2. Assume that vertex x has color-degree 2. In this case, assume that it has two neighbors with different colors. Now, using these two colors, construct color alternating paths from these vertices. Since G is finite, a cycle must be

formed. Since G is bipartite, this cycle must be even, and the neighbors of x must have the same color, contradicting our assumption. □

Example 8.3.2. Suppose we perform Algorithm 8.3.2 on the graph of our previous example. Initially, all vertices have color-degree zero; hence, we first select the vertex of highest degree, say v_1, and we assign v_1 the color 1. Since v_2, v_6 and v_7 now have color-degree 1, select v_2 since it has the largest uncolored degree (in this case 3). Then v_2 is assigned the color 2. Now, v_7 is the only vertex with color-degree 2, so it is selected and assigned color 3. All the remaining vertices have color-degree 1 and uncolored degree 2, so we randomly select v_3 and assign it color 1. Next, v_4 has color-degree 2, and it is then assigned color 3. This is followed by assigning v_5 color 2 and v_6 color 3. Thus, the bound obtained from this algorithm is 3, which we already know is the chromatic number of the graph. □

Brelaz [4] conducted some tests on his algorithm and compared the results with the smallest last heuristic, among others. Without increasing the level of sophistication of these algorithms, he found that Algorithm 8.3.2 was generally the best. However, since we are never, or at least rarely, content, let's increase the level of sophistication somewhat.

Suppose that a graph G is colored with k colors. Let $J_1, \ldots, J_k$ be the color classes determined by this coloring. Then, if we consider the graph induced by the union of any two of these color classes, say $< J_i \cup J_j >$, we see that it may not be connected. We term any component of $< J_i \cup J_j >$ an *i-j component.* If the sets J_i and J_j are interchanged, that is, the vertices in J_i are recolored j and the vertices of J_j are recolored i, then we say we have performed an *i-j interchange.* Clearly, the graph G is still k-colored. At times though, we can gain some flexibility after performing an interchange. The following algorithm introduces interchanges into our sequentially based methods.

Algorithm 8.3.3 Interchange Coloring Algorithm.
Input: A sequential ordering on the vertices of G.
Output: A coloring of G.
Method: We try to perform interchanges before using additional colors.

1. Assign v_1 the color 1.
2. If $H_{i-1} = < v_1, \ldots, v_{i-1} >$ has been colored with j colors, and if m is the least color not occurring on a neighbor of v_i in H_{i-1}, then
   a. if $m \leq j$, then assign v_i color m;
   b. if $m = j + 1$, then let C^1 be the set of colors that occur on exactly one vertex in $N_{H_{i-1}}(v_i)$.
   If some distinct pair $b, c \in C^1$ has a b, c-component of H_{i-1} with only one

neighbor of v_i in H_i, then perform a $b-c$ –interchange on one such component of H_{i-1}. Now, color v_i with the available color, producing a j coloring of H_i. If no such interchange is possible, color v_i with color $j+1$, and, hence, $j+1$ coloring H_i.

This color interchange approach can be applied in combination with any of the sequential ordering algorithms we have seen. Brelaz's [4] testing showed that improvements could be made when such an enhancement was incorporated into any of these algorithms.

In order to motivate the second category of coloring algorithms, we need to speculate about the reasons for our failure to be able to color graphs exactly. One reason might be the difference in the way in which the chromatic number seems to be determined in large graphs as opposed to small graphs. There appear to be two different lower bounds that drive up the chromatic number. It is certainly the case that $\chi(G) \geq \omega(G)$ (the *clique number*, that is, the order of the largest complete subgraph of G). It seems that for small graphs, $\chi(G)$ and $\omega(G)$ tend to be very close. On the other hand, no color class can contain more than $\beta(G)$ vertices, so it is also clear that $\chi(G) \geq \frac{|V(G)|}{\beta(G)}$. For small graphs, this lower bound tends to be much less than $\chi(G)$, while for large graphs, this lower bound seems to be much better than $\omega(G)$.

Matula has calculated the expected value of $\beta(G)$ for random graphs with edge probability 0.5. His estimates seem to predict the value of $\beta(G)$ very well and suggest the following approach. Based on the order and edge density of G, locate an independent set with the expected number of vertices. Now, delete this set and in the graph that remains, repeat this process. Continue until all vertices are colored.

Johri and Matula [14] have produced a variety of algorithms based on this approach. We shall briefly describe their attack. In small graphs we can carry out an exhaustive search for the desired independent sets and perform the algorithm basically as described. However, in large graphs this is not practical. Thus, they turned to a two-step approach. The first step is to find an independent set of vertices, say I_1, that is fairly large with respect to the the desired size. That is, I_1 is within some tolerance t_1 of the expected value of $\beta(G)$. The second step is to search in the remaining collection of vertices with no adjacencies to I_1 for the largest expected independent set, say I_2. The only catch here is that we do not want to drastically change the edge density by removing independent sets carelessly; hence, I_2 is selected to cover as many edges as possible. Then, $I_1 \cup I_2$ is deleted from the graph and is used as the next color class.

It is clear that this general method is very flexible and offers a great many variations for experimentation. Thus, with more complex testing and conditions, progressively slower versions can be created that color with progressively fewer colors.

Johri and Matula tried several variations of this algorithm. The first, called GE1, used random selection of the vertices in step 1 to find I_1. Their second algorithm, GE2, tried to increase the size of $I_1 \cup I_2$ at each stage by sampling various sets. If $I_1 \cup I_2$ was not as large as the desired independent set, then the step was repeated in an effort to find better candidates. After a certain number of failures, the expectation would be readjusted downward and the search would be continued. Their third algorithm, GE3, selects vertices to add to I_1 by taking those whose neighbors have the largest average degree. This algorithm also reverts to a simple exhaustive search when 80 or fewer vertices remain to be colored.

The following table is extracted from their work. It is based on tests they performed on ten random graphs of order 1000 with edge probability 0.5. The time is in CPU seconds on a CDC 6600.

Algorithm Tested	Average No. of Colors	Average Time
Largest First	122.7	101.3
Smallest Last	124.3	126.7
Brelaz Algorithm	115.8	111.7
GE1	105.2	432.8
GE2	100.1	1128.2
GE3	95.9	3212.2

Section 8.4 Edge Colorings

A natural analog to coloring the vertices of a graph is coloring the edges. We define the *edge chromatic number,* sometimes called the *chromatic index,* to be the least number of colors needed to color the edges of a graph G so that no two adjacent edges are assigned the same color. Denote the edge chromatic number of the graph G as $\chi_1(G)$. An immediate observation is that $\chi_1(G) = \chi(L(G))$. Another easy observation is that if G contains a vertex of degree k, then $\chi_1(G) \geq k$. Some examples of edge chromatic numbers of special classes of graphs are also easy to see. For instance,

$$\chi_1(C_p) = \begin{cases} 2 & \text{if } p \text{ is even,} \\ 3 & \text{if } p \text{ is odd.} \end{cases}$$

$$\chi_1(K_p) = \begin{cases} p-1 & \text{if } p \text{ is even,} \\ p & \text{if } p \text{ is odd.} \end{cases}$$

It turns out that we can bound the edge chromatic number fairly tightly for graphs and somewhat less effectively, but still reasonably, for multigraphs. We now turn our attention to developing these bounds. The bound for graphs is from Vizing [20].

Theorem 8.4.1 If G is a graph, then $\Delta(G) \leq \chi_1(G) \leq \Delta(G) + 1$.

Proof. It is clear that for any graph G, $\chi_1(G) \geq \Delta(G)$. Thus, we need only show that the upper bound also holds. To do this, we use induction on the number of edges in G. The result is clear if G has only one edge; thus, assume that all but one edge of G has been colored using at most $\Delta(G) + 1$ colors. Say the remaining uncolored edge is $e_1 = vw_1$. There must be at least one color unused at v and at least one color unused at w_1. If the same color is unused at both vertices, then we color e_1 with this color and we have the desired edge coloring of G. If this is not the case, then let c_0 be an unused color at v and let c_1 ($\neq c_0$) be an unused color at w_1. We now consider a three-step algorithm that will complete the proof.

Step 1. Let $e_2 = vw_2$ be the edge incident to v which has been assigned the color c_1. Since c_1 was not used at w_1, we know that such an edge must exist. Remove this color from e_2 and assign it instead to e_1. We may also assume that v, w_1 and w_2 all belong to the same component induced by the edges colored c_0 and c_1 or else we could interchange the colors of the edges in the component containing w_2 without changing the color of e_1. But if that were the case, we could color e_2 with c_0 and obtain a proper coloring of G. Let $P(c_0, c_1)$ be the bicolored path joining w_1 and w_2 in this component (see Figure 8.4.1).

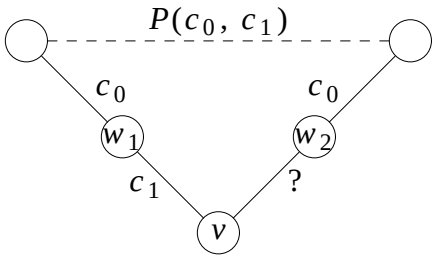


Figure 8.4.1. The configuration of step 1.

Step 2. Let c_2 ($\neq c_1$) be any unused color at w_2. We may assume that c_2 is used at v or else we could complete the proof by coloring e_2 with c_2. Thus, let $e_3 = vw_3$ be the edge incident to v with color c_2. Then, we can remove color c_2 from e_3 and assign it to e_2. By the argument used in step 1, we may assume that v, w_2 and w_3 all belong to the same two-color component of G induced by c_0 and c_2. Let $P(c_0, c_2)$ be the bicolored path joining w_2 and w_3 in this component (see Figure 8.4.2).

Step 3. If we repeat the procedure of step 2, we eventually reach a vertex w_k that is adjacent to v, but the edge vw_k is uncolored and some color c_i ($i < k - 1$) is unused at w_k. Again, we may assume that v, w_i and w_{i+1} all belong to the same two-color component H of G obtained by using c_0 and c_i. Since c_0 is missing at v and c_i is missing at w_{i+1}, then H must be a path from v to w_{i+1} that passes through w_i and consists entirely of edges alternately colored c_0 and c_i. This path does not contain w_k

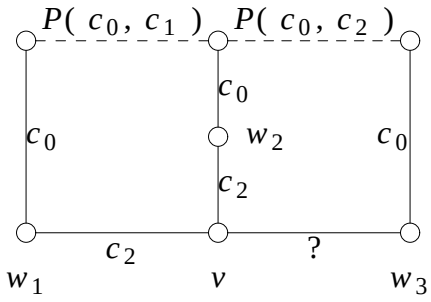


Figure 8.4.2. Colorings in step 2.

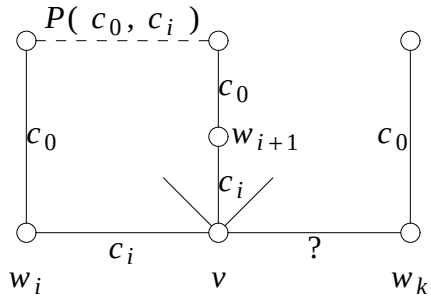


Figure 8.4.3. Step 3.

since c_i does not appear at w_k (see Figure 8.4.3). Thus, if H_k is the two-color component of G obtained by using c_0 and c_i and containing the vertex w_k, then H_k and H must be disjoint. We can then interchange the colors of the edges in H_k and then color vw_k with c_0. This completes the proof. □

Our next two results provide upper bounds on the edge chromatic number of a multigraph. The first is from Vizing [21] and the second from Shannon [19]. Vizing's bound is sometimes better than the bound of Shannon. To state Vizing's result, let the *maximum multiplicity* $m(G)$ be defined as the maximum number of edges joining any pair of vertices in a multigraph G. Vizing's bound for multigraphs can now be stated. It can also be viewed as a generalization of his bound for graphs since for graphs $m(G) = 1$.

Theorem 8.4.2 If G is a multigraph, then $\Delta(G) \leq \chi_1(G) \leq \Delta(G) + m(G)$.

Example 8.4.1. The upper bound in Theorem 8.4.2 is sharp. This can be seen from the multigraph of Figure 8.4.4. It has maximum degree $2x$ and multiplicity x. Since any two edges are adjacent, we see that χ_1 must be $3x = q$.

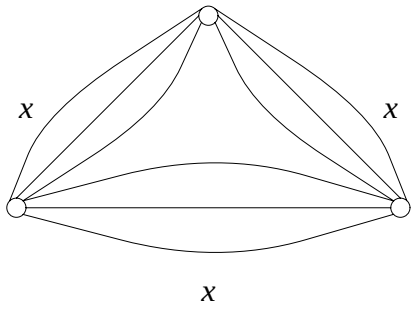


Figure 8.4.4. Sharpness example for Theorem 8.4.2.

We now present Shannon's [19] bound for multigraphs. We shall use Vizing's generalization to aid in our proof.

Theorem 8.4.3 If G is a multigraph, then $\chi_1(G) \leq \frac{3}{2}\,\Delta(G)$.

Proof. Let G be a multigraph with $\chi_1(G) = k$, where $k > \frac{3}{2}\,\Delta(G)$. By removing sufficient edges from G, we can obtain a minimal multigraph M from G. By Theorem 8.4.2, we know that $\chi_1(M) \leq \Delta(M) + m(M)$. Thus, there must be vertices v and w that are joined by at least $k - \Delta(M) > \Delta(M)/2$ edges.

Now, color all the edges of M except one of the edges between v and w. Since M is minimal, this can be done using only $k - 1$ colors. The number of colors unused at v or w (or both) cannot exceed

$$(k - 1) - (\Delta(M) - 1) \leq m(M)$$

since $k \leq \Delta(M) + m(M)$. But, the number of colors unused at v (or w) is also at least

$$(k - 1) - (\Delta(M) - 1) = k - \Delta(M) > \Delta/2.$$

Then, we see that the number of colors unused at both v and w is at least

$$2(k - \Delta(M)) > 0.$$

By assigning one of these unused colors to the uncolored edge from v to w we obtain a coloring of M with only $k - 1$ colors. But this contradicts the fact that $\chi_1(M) = k$, and the result is proved. □

Vizing's theorem (8.4.1) has set off a rather extensive study attempting to classify graphs according to their edge chromatic number. A graph G is said to be of *class 1* if $\chi_1(G) = \Delta(G)$ and of *class 2* otherwise. From the examples we have seen, we know that K_{2n} is of class 1 and that K_{2n+1} is of class 2. However, the general problem of deciding which graphs are class 1 and which are class 2 (sometimes called the

classification problem) remains unsolved. Evidence exists to show that class 2 graphs are fairly rare. In fact, Erdös and Wilson [9] have shown that almost all graphs are class 1; that is, if $Pr(n)$ is the probability that a random graph of order n is class 1, then $Pr(n) \to 1$ as $n \to \infty$. However, it also seems natural to expect that the more edges a graph contains, the more likely it is to be in class 2. The following result from Beineke and Wilson [2] confirms this idea.

Theorem 8.4.4 Let G be a (p, q) graph. If $q > \Delta(G)\ \beta_1(G)$, then G is of class 2.

Proof. If G is class 1, then any $\Delta(G)-$coloring of the edges of G partitions the edges into $\Delta(G)$ independent sets. Since the number of edges in each such set cannot exceed $\beta_1(G)$, then $q \leq \Delta(G)\beta_1(G)$, a contradiction. □

Further results about the classification problem will be explored in the exercises.

Section 8.5 The Four Color Theorem

The idea of coloring can be traced to Francis Guthrie. In 1852, while he was a student of Augustus De Morgan, Guthrie asked De Morgan to verify the "fact" that any map (consider it a map of countries if you wish) drawn in the plane could be colored with at most four colors, so that adjacent (that is, sharing a boundary) countries received different colors. De Morgan responded by saying he did not know that this was a "fact," and he proceeded to ask other mathematicians (like Hamilton) about this problem. Both Guthrie and De Morgan believed this statement was indeed a fact, yet neither could verify it.

If we consider a map drawn in the plane and insert a vertex in each country and join two vertices by an edge if the corresponding countries share a common boundary, then we have created a graph model of the map, and this model is easily seen to be planar. We created this model in much the same way as the geometric dual of a graph and, in fact, the graph model can be thought of as the dual of the map. The problem of coloring the countries of this map can then be stated as a graph-coloring problem. The "fact" that De Morgan and Guthrie could not prove can now be stated.

The Four Color Conjecture Every planar graph can be colored with four or fewer colors.

The four color conjecture was to become one of the most famous of all mathematical problems. It is sometimes called the four color disease. By this we mean that many good mathematicians spent a great deal of time (in fact, for some a lifetime) working on this problem without complete success. This problem has generated a strange history filled

with attempts at proofs, publication of incorrect proofs and, in general, a great deal of unrewarded efforts.

The first and most famous attempt at a proof was provided by Alfred Bay Kempe [15]. His proof appeared in 1880 (it was announced in 1879), and for ten years the problem was believed to be settled. Then in 1890, Heawood [13] discovered an error in Kempe's proof. Heawood was able to modify Kempe's argument to produce the following result.

Theorem 8.5.1 Every planar graph is 5-colorable.

Proof. We proceed by induction on the order of the graph. Let G be a graph of order p. If $p \leq 5$, the result is clear. Assume that $p \geq 6$ and inductively assume that all planar graphs of order $p - 1$ are $5-$colorable. By Corollary 6.1.5, we know that G contains a vertex v of degree at most 5. Then, by our assumptions, $G - v$ is planar, has order $p - 1$ and is $5-$colorable. Consider a $5-$coloring of $G - v$. If this coloring does not use all 5 colors on the vertices in $N(v)$, then we can assign v one of the missing colors and obtain a $5-$coloring of G. Thus, suppose all five colors are used in $N(v)$ (hence, $deg\ v = 5$).

Without loss of generality, we can assume the vertices adjacent to v in G are v_i and that each has color i, $i = 1, 2, 3, 4, 5$. Further assume that these vertices are arranged cyclically about v. Consider any two colors assigned to nonconsecutive vertices, say 1 and 3, and let H be the subgraph of $G - v$ induced by the vertices colored 1 and 3. If the vertices v_1 and v_3 belong to different components of H, then by interchanging colors in one of these components, we can free a color to use for v.

Thus, we suppose that v_1 and v_3 belong to the same component of H. Thus, there exists a path P from v_1 to v_3 that has its vertices alternately colored 1 and 3. The path P, along with the path v_1, v, v_3, produces a cycle that completely encloses v_2 or both v_4 and v_5. Thus, there exists no path alternately colored 2 and 4 joining v_2 and v_4 in G. Let H_1 be the subgraph of G induced by those vertices colored 2 and 4. Interchanging the colors in the component of H_1 containing v_2 frees the color 2 for use on v, producing the desired $5-$coloring of G. □

In the years that followed Heawood's work, a great deal of time and effort went into the study of graph colorings. Finally, in 1976, Appel and Haken [1], with the computer aid of Koch, verified the four color conjecture. Their general strategy was very similar to Kempe's original idea. However, their proof is very long and has many cases, and it required nearly 1200 hours of computer time to check that these cases all worked. The somewhat amazing fact here is that they were able to build a theory that reduced the infinitely many possible structures to a finite collection of cases, regardless of the number of these cases.

Theorem 8.5.2 Every planar graph is 4-colorable.

Section 8.6 Chromatic Polynomials

In an attempt to study the four color conjecture, Birkhoff [3] found that studying the number of colorings could be helpful. Two colorings of G are regarded as distinct provided some vertex is assigned different colors in the two colorings. Suppose we denote by $c_k(G)$ the number of distinct k colorings of G. By our definition, $c_k(G) > 0$ if, and only if, G is k colorable.

Example 8.6.1. It is easy to see that K_3 has six colorings. First, color one vertex and note that there are two colorings possible on the remaining two vertices. These colorings are obtained by interchanging the remaining two colors on these two vertices. Two of the six colorings are shown in Figure 8.6.1; the rest come about by permuting the roles of the colors.

Figure 8.6.1. Two distinct colorings of K_3.

It is straightforward to see that if $G = K_n$ $(n \leq k)$, then there are k choices for coloring the first vertex, $k - 1$ choices for coloring the second vertex, and so forth. Hence,

$$c_k(K_n) = k(k-1)\cdots(k-n+1).$$

Also, if G is empty, then any vertex can be assigned any one of the k colors and so

$$c_k(G) = k^n.$$

In general, we can determine a recurrence relation for $c_k(G)$ that is similar to our formula for the number of spanning trees of G. We let G/e denote the simple graph (with loops or multiple edges removed) obtained by identifying the end vertices of the edge e.

Theorem 8.6.1 If G is a graph, then $c_k(G) = c_k(G - e) - c_k(G / e)$ for any edge e of G.

Proof. Let $e = uv$. For each k-coloring of $G - e$ that assigns the same color to u and v, there corresponds a k-coloring of G/e in which the vertex of G/e formed by identifying u and v is assigned the common color of u and v. Therefore, $c_k(G/e)$ is just the number of

k-colorings of $G - e$ in which u and v have the same color.

Since each k-coloring of $G - e$ that assigns different colors to u and v is also a legal k-coloring of G, and conversely, then $c_k(G)$ is the number of k-colorings in $G - e$ in which u and v are assigned different colors. Thus,

$$c_k(G - e) = c_k(G) + c_k(G/e),$$

and the result follows. □

This recurrence relation will allow us to describe chromatic polynomials well enough to actually justify calling them polynomials. The following corollary is again from Birkhoff [3].

Corollary 8.6.1 For any graph G of order n, $c_k(G)$ is a polynomial in k of degree n. Further, this polynomial has integer coefficients, leading term k^n, constant term 0 and the coefficients alternate in sign.

Proof. We proceed by induction on the size of G. If G is empty, then we already know that $c_k(G) = k^n$. Thus, suppose the result holds for all graphs with fewer than q edges and let G be a graph with q edges, $(q \geq 1)$. Let e be an arbitrary edge of G. Then both $G - e$ and G/e have size $q - 1$, and from the induction hypothesis, we know that there are nonnegative integers $r_1, \ldots, r_{n-1}$ and $s_1, \ldots, s_{n-2}$ such that

$$c_k(G - e) = \sum_{i=1}^{n-1} (-1)^{n-i} r_i k^i + k^n \quad \text{and}$$

$$c_k(G/e) = \sum_{i=1}^{n-2} (-1)^{n-i-1} s_i k^i + k^{n-1}.$$

Now, by Theorem 8.6.1, we see that

$$\begin{aligned} c_k(G) &= c_k(G - e) - c_k(G/e) \\ &= \sum_{i=1}^{n-2} (-1)^{n-i}(r_i + s_i)k^i - (r_{n-1} + 1)k^{n-1} + k^n. \end{aligned}$$

But then G also satisfies the conditions of the corollary, and so the result follows. □

We now have a means of calculating the chromatic polynomial of a graph using our recursive formula. In fact, there are two possible approaches. We can either use the recursion formula as a difference of terms, which allows us to reduce from our graph G to empty graphs or we can view our graph as the graph $G - e$ and use the sum formula to reduce to complete graphs. In the following example, we take both approaches for the path P_3.

Example 8.6.2. Given the graph P_3, we first use the difference formula:

$$c_k(P_3) = c_k(P_3 - e) - c_k(G/e) = c_k(P_3 - e) - c_k(K_2)$$

$$= c_k(3K_1) - c_k(2K_1) - (c_k(2K_1) - c_k(K_1))$$

$$= c_k(3K_1) - 2c_k(2K_1) + c_k(K_1) = k^3 - 2k^2 + k.$$

If $k = 2$, then $c_2(P_3) = 2$. This is easily verified while if $k = 3$, then $c_3(K_3) = 12$. In Figure 8.6.2, four of the twelve colorings are shown. The rest can be obtained by permuting colors.

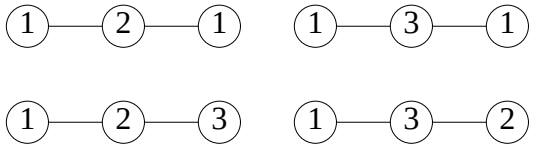


Figure 8.6.2. Four colorings of P_3.

Now, to apply the recursion as a sum formula, we view P_3 as $G - e$ and, thus, we compute

$$c_k(G - e) = c_k(G) + c_k(G/e) = c_k(K_3) + c_k(K_2)$$

$$= k(k-1)(k-2) + k(k-1) = k^3 - 2k^2 + k.$$

Thus, either method effectively produces the chromatic polynomial for P_3. □

Section 8.7 Perfect Graphs

Recall that a complete subgraph is called a *clique*, and that the maximum order of a clique of G is called the *clique number* of G and is denoted $\omega(G)$. Clearly, $\omega(G) \leq \chi(G)$. One might consider the class of graphs in which $\omega(G) = \chi(G)$, but this class has too little structure to be of much use. Berge introduced a related class of graphs, in which there is enough structure to gain valuable information. In fact, for this class the independence number and chromatic number are computable in polynomial time. A graph G is *perfect* if G and each of its induced subgraphs have the property that their chromatic number equals their clique number. The graph of Figure 8.7.1 is perfect. Its clique number is easily seen to be 3, and a 3-coloring of its vertices is shown. It is straightforward to convince oneself that this graph is perfect.

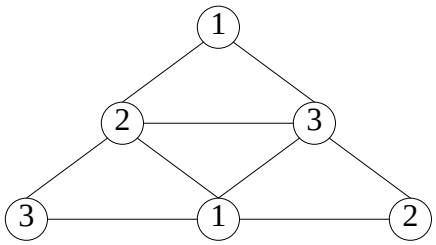


Figure 8.7.1. A perfect graph.

It is also easy to see that every bipartite graph is perfect. It requires a little more effort to see that the complement of a bipartite graph is also perfect.

Theorem 8.7.1 The complement $\overline{G}$ of any bipartite graph G is perfect.

Proof. Let G be a bipartite graph. Then each induced subgraph of $\overline{G}$ has the form $\overline{H}$, where H is an induced subgraph of G. If H has no isolated vertices, then we know that $\alpha_1(H) = \beta(H)$ (see Theorem 8.1.2). Since it is clear that $\beta(H) = \omega(\overline{H})$, we need only show that $\chi(\overline{H}) = \alpha_1(H)$ in order to establish the fact that $\overline{G}$ is perfect.

Clearly, the chromatic number of $\overline{H}$ equals the minimum number of elements in a partition of $V(H)$ such that each element of the partition induces a complete subgraph in H. Since H contains no triangles, each such complete subgraph has order 1 or 2. It follows, then, that such a partition contains $\alpha_1(H)$ elements and, thus, $\chi(\overline{H}) = \alpha_1(H)$. Since a similar argument can be applied if H has isolated vertices, the proof is complete. □

There are many other classes of graphs that are perfect. An interesting class is obtained in the next result.

Theorem 8.7.2 If a graph G is P_4-free, then G is perfect.

Proof. We proceed by induction on the order p of G. The result is clear if $p = 1$. Assume the result holds for all graphs of order less than p, ($p \geq 2$) and let G be a graph of order p that is P_4-free. From exercise 28 in this chapter, for every nontrivial subset S of $V(G)$, either $< S >_G$ or $< S >_{\overline{G}}$ is disconnected. In particular, this implies that either G or $\overline{G}$ is disconnected.

Suppose that G is disconnected and say $C_1, C_2, \ldots, C_k$ are the components of G ($k \geq 2$). Since G is P_4-free, so are each of its components. Further, since each component has order less than p, by the inductive hypothesis, each component is perfect. But then

$$\chi(G) = \max_i \chi(C_i)$$

and

$$\omega(G) = \max_i \omega(C_i).$$

Thus, $\chi(G) = \omega(G)$. Since the same agrument applies to any induced subgraph of G, we see in this case that G is perfect.

Now, assume that $\overline{G}$ is disconnected. Let $B_1, \ldots, B_t$ be the components of $\overline{G}$. Each $\overline{B}_i$ is a subgraph of G of order less than p; hence, by the inductive hypothesis

$$\chi(\overline{B}_i) = \omega(\overline{B}_i) \quad \text{for } i = 1, 2, \ldots, t.$$

Furthermore, G is the join of the $\overline{B}_i$s. Thus,

$$\chi(G) = \sum_{i=1}^{t} \chi(\overline{B}_i)$$

and

$$\omega(G) = \sum_{i=1}^{t} \omega(\overline{B}_i) \quad \text{(see exercise 25 in this chapter).}$$

Thus, $\chi(G) = \omega(G)$. Since the same argument applies to any induced subgraph of G, we see G is perfect. □

Berge conjectured that the complement of any perfect graph was also perfect. This result was proved independently by Fulkerson [10] and Lovász [16].

Theorem 8.7.3 (The Perfect Graph Theorem) The complement of a perfect graph is perfect.

The odd cycle C_{2k+1} (for $k \geq 2$) is not a perfect graph, since $\omega(C_{2k+1}) = 2$ and $\chi(C_{2k+1}) = 3$. However, every proper subgraph of C_{2k+1} is perfect. Thus, in a sense, C_{2k+1} is minimally imperfect! Such graphs have come to be called *p-critical.* The same fact holds for the complement of C_{2k+1}. To date, these are the only two known graphs with this property. This lead Berge to the following conjecture, which can be stated in several equivalent ways.

The Strong Perfect Graph Conjecture

1. A graph G is perfect if, and only if, neither G nor $\overline{G}$ contains as an induced subgraph an odd cycle of length at least 5.
2. A graph G is perfect if and only if in G and $\overline{G}$ every odd cycle of length at least 5 has a chord.

3. The only p-critical graphs are C_{2k+1} and $\overline{C}_{2k+1}$.

Exercises

1. Show that $\chi_1(K_{m,n}) = \max\{ m, n \}$.
2. Show that $\chi(K_{2n}) = \Delta(K_{2n})$ and that $\chi(K_{2n+1}) = \Delta(K_{2n+1}) + 1$.
3. Show that if G is a bipartite graph, then $\chi_1(G) = \Delta(G)$.
4. Prove that if G is a graph of order p with $\delta(G) > 0$, then $\alpha_1(G) + \beta_1(G) = p$.
5. Prove that $\chi(K_{p_1, \ldots, p_n}) = n$.
6. Prove that if G is k-partite, then $\chi(G) \leq k$.
7. Prove Corollary 8.2.1.
8. Prove Corollary 8.2.2.
9. Prove Corollary 8.2.3.
10. Prove Theorem 8.3.1.
11. Show that every k-chromatic graph is a subgraph of some complete k-partite graph.
12. Determine the n-critical graphs for $n = 1, 2, 3$.
13. Show that a critically n-chromatic graph need not be $(n - 1)$-connected.
14. Characterize graphs whose line graphs are 2-colorable.
15. Show that for every graph G, $\chi(G) \leq 1 + \max \delta(H)$ where the maximum is taken over all induced subgraphs H of G.
16. If $m(G)$ denotes the length of a longest path in G, prove that $\chi(G) \leq 1 + m(G)$.
17. Find a largest first ordering of the vertices of the graph in Example 8.3.1 that produces a sequential coloring using three colors.
18. Show that every regular graph of odd order is class 2.
19. Show that if H is a regular graph of odd order and if G is any graph obtained from H by deleting at most $\frac{1}{2}\delta(G) - 1$ edges, then G is of class 2.
20. Show that if H is a regular graph of even order and if G is any graph obtained from H by subdividing any edge of H, then G is class 2.
21. Show that if G is any graph obtained from an odd cycle C_{2k+1} by adding no more than $2k - 2$ independent edges, then G is class 2.

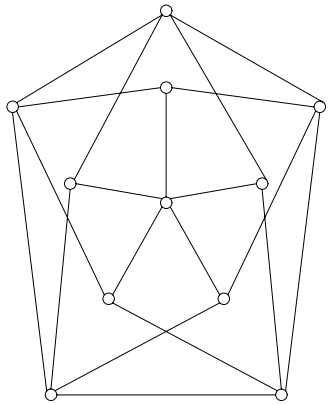
22. Show that if G is a regular graph containing a cut vertex, then G is of class 2.

23. Show that there are no regular $\delta(G)$-minimal graphs with $\delta(G) \geq 3$.

24. Show that every bipartite graph is perfect.

25. Let $G_1, G_2, \ldots, G_k$ be pairwise disjoint graphs. Also let $G = G_1 + G_2 + \cdots + G_k$. Prove that $\chi(G) = \sum \chi(G_i)$ and that $\omega(G) = \sum_{i=1}^{k} \omega(G_i)$.

26. Use the largest first, smallest last and color-degree algorithms to bound the chromatic number of each of the following graphs.
    a. $K_{1,3}$
    b. $K_4 - e$
    c. The Petersen graph
    d. The Grötsch graph shown below



27. Find the chromatic polynomial for $K_{1,3}$ and for $K_4 - e$. How many 5-colorings are there for each of these graphs?

28. Let G be a graph. For every nontrivial subset S of $V(G)$, either $< S >_G$ or $< S >_{\overline{G}}$ is disconnected if, and only if, G is P_4-free.

References

1. Appel, K., Haken, W., and Koch, J., Every Planar Map is Four Colorable. *Illinois J. Math.*, 21(1977), 429 – 567.

2. Beineke, L. W., and Wilson, R. J., On the Edge Chromatic Number of a Graph. *Discrete Math.,* 5(1973), 15 – 20.

3. Birkhoff, G. D., A Determinant Formula for the Number of Ways of Coloring a Map. *Ann. of Math.,* 14(1912), 42 – 46.

4. Brelaz, D., New Methods to Color the Vertices of a Graph. *Comm. ACM,* 22(1979), 251 – 256.

5. Brooks, R. L., On Coloring the Nodes of a Network. *Proc. Cambridge Philos. Soc.,* 37(1941), 194 – 197.

6. Christofides, N., An Algorithm for the Chromatic Number of a Graph. *The Computer Journal,* 14(1971), 38 – 39.

7. Dirac, G. A., A Property of 4-Chromatic Graphs and Some Remarks on Critical Graphs. *J. London Math. Soc.,* 27(1952), 85 – 92.

8. Dirac, G. A., The Structure of *k*-Chromatic Graphs. *Fund. Math.,* 40(1953), 42 – 50.

9. Erdös, P., and Wilson, R. J., On the Chromatic Index of Almost All Graphs. *J. Combinatorial Theory B,* 26(1977), 255 – 257.

10. Fulkerson, D. R., Blocking and Anti-Blocking Pairs of Polyhedra. *Math. Programming,* 1(1971), 168 – 194.

11. Gallai, T., Über Extreme Punkt und Kantenmengen. *Ann. Univ. Sci. Budapest, Eötvös Sect. Math.,* 2(1959), 133 – 138.

12. Garey, M. R., and Johnson, D. S., *Computers and Intractability.* Freeman Publishing, San Francisco (1979).

13. Heawood, P. J., Map-Color Theorem. *Quart. J. Math.,* 24(1890), 332 – 339.

14. Johri, A., and Matula, D. W., Probabilistic Bounds and Heuristic Algorithms for Coloring Large Random Graphs. (in press).

15. Kempe, A. B., On the Geographical Problem of the Four Colors. *Amer. J. Math.,* 2(1879), 193 – 200.

16. Lovász, L., Normal Hypergraphs and the Perfect Graph Conjecture. *Discrete Math.,* 2(1972), 253 – 267.

17. Manvel, B., Extremely Greedy Coloring Algorithms. *Graphs and Applications,* ed. Harary, F., and Maybee, J., John Wiley and Sons, Inc., New York (1985), 257 – 270.

18. Matula, D. W., Marble, G., and Isaacson, J. D., Graph Coloring Algorithms. *Graph Theory and Computing,* ed. Read, R., Academic Press, New York, (1972), 109 – 122.

19. Shannon, C. E., A Theorem on Coloring the Lines of a Network. *J. Math. Phys.*, 28(1949), 148 – 151.

20. Vizing, V. G., On an Estimate of the Chromatic Class of a *p*-Graph. (Russian), *Diskret. Analiz,* 3(1964), 25 – 30.

21. Vizing, V. G., The Chromatic Class of a Multigraph. *Cybernetics*, 3(1965), 32 – 41.

22. Welsh, D. J. A., and Powell, M. B., An Upper Bound to the Chromatic Number of a Graph and its Application to Time-Table Problems. *Comp. J.*, 10(1967), 85 – 86.

Chapter 10
Extremal Theory

Section 10.0 Introduction

We now begin a study of one of the most elegant and deeply developed areas in all of graph theory, extremal graph theory. We have often dealt with extremal questions. For example, earlier we tried to determine the minimum number of edges e so that every graph of order n with at least e edges contained a cycle. This was one of the first problems we attempted when studying trees. Other extremal problems we have investigated include ramsey theory, finding bounds on the cardinality of neighborhood unions that ensured hamiltonian cycles and degree requirements that ensured hamiltonian properties.

Extremal graph theory, in its most general form, concerns any problem which attempts to determine the relation between graph invariants (such as order, size or minimum degree) and a graph property (like being hamiltonian, containing a perfect matching or containing a particular subgraph G_1). Typically, given a graph property P, an invariant i and a class of graphs $\hat{H}$, one tries to determine the least value m such that every graph G in $\hat{H}$ with $i(G) > m$ has property P.

We shall limit our investigation to that question generally credited with starting extremal theory and to the beginnings of the research that sprang from this question. This study is rich in counting techniques and estimations. We shall use elementary results about convex functions to obtain some bounds. All the necessary results on convex functions are contained in the appendix or the exercises.

We shall limit our investigation to a particular type of extremal problem whose initiation is generally credited to Turán [19]. In this problem, we ask the following: Given a graph G, determine the maximum number of edges, $ex(n; G)$ in a graph of order n that does not contain G as a subgraph. A graph E of order n with $ex(n; G)$ edges and not containing G as a subgraph is called an *extremal graph* for this problem. The complete solution of any such problem ordinarily requires two things. First, we must produce an extremal graph on n vertices and $ex(n; G)$ edges that does not contain any G. Second, we must show that any graph on n vertices and with at least $ex(n; G) + 1$ edges must contain a G.

The investigation of this extremal problem will eventually lead us to the study of the structure of extremal graphs. A rather beautiful theory has been developed that essentially tells us that the exact structure of the forbidden graphs themselves is not really as important as their chromatic number.

Section 10.1 Complete Subgraphs

We begin with Turán's original problem: What is the maximum number of edges q in a graph of order n that does not contain the complete graph K_p? We begin by producing the extremal graph for the Turán problem. This graph is easy to describe. For the forbidden graph K_{p+1} (with chromatic number $p + 1$), we begin with the complete p-partite graph $K_{n_1, n_2, \ldots, n_p}$ where $n = \sum n_i$. It is easy to show that among all such graphs, the one with the maximum number of edges is that graph with partite sets as nearly equal as possible. In fact, among all graphs on n vertices with chromatic number p, it has the maximum number of edges. Thus, if $n = kp + r$, $0 \le r < p$, then $p - r$ of the partite sets contain k vertices and the remaining r of the partite sets contain $k + 1$ vertices. We denote this graph as $T_{n, p}$, and call it the *Turán graph.* We further note that

$$|E(T_{n,p})| = \binom{n}{2} - \frac{k(n - p + r)}{2}.$$

We are now ready to state Turán's theorem.

Theorem 10.1.1 Among the graphs of order n which do not contain K_p, there exists exactly one with the maximum number of edges, namely $T_{n,p-1}$.

We will present (or at least sketch) two proofs to Turán's theorem, showing two common and useful techniques in extremal theory. The first technique is called "chopping" and resembles Turán's original proof. The strategy is to chop off a "useful" subgraph and work around this structure to complete the proof, carefully avoiding the "chopped" graph. For convenience and to maintain a notation common in extremal theory, we denote by G^n a graph of order n.

Turán's Chopping Proof. We proceed by induction on n, the order of the extremal graph under construction. The anchor is trivial so assume the result holds for orders less than n and suppose the extremal graph G^n is $K_p -$ free. Since G is extremal, it follows that $H = K_{p-1} \subseteq G^n$ and define q_1, q_2, q_3 as follows:

$$\begin{aligned} q_1 &= |E(H)| = \binom{p-1}{2}, \\ q_2 &= \text{no. of edges between } H \text{ and } V - H \le (n - p + 1)(p - 2) \\ q_3 &= |E(V - H)| \le |E(T_{n-p+1,p-1})| \end{aligned}$$

(the bound in the third expression follows from the inductive assumption).

It is clear that $|E(G^n)| = q_1 + q_2 + q_3$, and by summing the bounds given on q_1, q_2 and q_3 we see that

$$|E(G^n)| \le |E(T_{n,p-1})|.$$

It remains to show that if equality holds, then $G^n = T_{n,p-1}$. Clearly, $q_2 = (n - p + 1)(p - 2)$. This determines a partition of $V(G^n)$ into $p - 1$ classes, defined according to their $p - 2$ adjacencies in H. These classes are clearly independent, so G^n is a complete $(p - 1)$ partite graph defined by these classes, that is, $G = T_{n,\,p-1}$. □

The second proof technique, known as *symmetrization,* has become a powerful tool in extremal theory. The process of symmetrization proceeds as follows: Given nonadjacent vertices v and u, we delete all the edges incident to the vertex u and make u adjacent to all vertices in $N(v)$. The vertex u is then said to be *symmetric to v.* We can see that under certain conditions symmetrization can be useful in extremal problems. First, it is clear that no K_p is formed during this process, since only the vertices of $N(v)$ have new adjacencies and u and v are not adjacent. Thus, if a K_p now exists, it must have existed prior to symmetrization. Second, if $deg\ u < deg\ v$, then we have increased the number of edges in the graph without producing the forbidden K_p. We now sketch a second proof of Turán's theorem using symmetrization from Zykov [21].

Sketch of Zykov's Proof. We assume the anchor and inductive steps have been performed and consider the extremal graph G^n which is K_p-free. Let v have maximum degree in G^n and symmetrize all of $V - N(v)$ to v. Let S_1 denote these vertices along with v. Clearly, S_1 is an independent set of vertices. Further, since v had maximum degree, our new graph has at least as many edges as G^n. Now, repeat this process on $< G^n - S_1 >$, forming the set S_2. Continue the procedure, forming the sets $S_3, \ldots, S_d$. As we noted earlier, since G^n was K_p-free, this new graph formed by symmetrization is also K_p-free. (That is, $d \le p - 1$). Thus, any K_p-free graph can be transformed into a d-partite $(d \le p - 1)$ graph. Further, to maximize the number of edges in such a graph, standard convexity arguments (as noted before) imply that the graph is actually $T_{n,p-1}$. □

The following is immediate from Turán's Theorem, but was originally proven by Mantel in 1906 [13]. We provide an independent proof of this result because the technique is different and very interesting.

Corollary 10.1.1 If G^n is K_3-free, then $|E(G^n)| \le \frac{n^2}{4}$.

Proof: Suppose G is as described and number the vertices of G from 1 to n. Assign vertex i a weight of $w_i \ge 0$ such that $\sum_{i=1}^{n} w_i = 1$. Our goal is to maximize

$$S = \sum_{ij \in E(G)} w_i w_j$$

(where the sum is taken over all edges in G). Suppose vertices u and v are not adjacent in G. Let the neighbors of u have total weight x and let the neighbors of v have total weight y, where we assume without loss of generality that $x \geq y$.

Since

$$(w_u + \varepsilon)x + (w_v - \varepsilon)y \geq w_u x + w_v y$$

we do not decrease the value of S if we shift some weight from the vertex v to the vertex u. It follows that S is maximal if all the weight is concentrated on some complete subgraph of G, in fact, on one edge. But then $S \leq \frac{1}{4}$ (applying standard convexity). On the other hand, taking all $w_i = n^{-1}$, we see that $S \geq n^{-2}|E|$ But then these two inequalities imply that $|E| \leq \frac{n^2}{4}$. □

We now state an extension of Turán's theorem from Erdös [5] which can be used to provide yet another proof of Turán's theorem.

Theorem 10.1.2 Let G^n be a K_p-free graph with degree sequence $d_1 \geq d_2 \geq \cdots \geq d_n$. Then there exists a $(p-1)$ chromatic graph H^n which is K_p-free with degrees $s_1 \geq s_2 \geq \cdots \geq s_n$ and such that $s_i \geq d_i$ for every i.

We continue our investigation of complete subgraphs with a theorem from Dirac [4] that shows that we actually get more than a K_p once we have more than the extremal number of edges. To this end, we say that a graph H is *saturated* (in particular, we say a graph H is *G-saturated)* if H does not contain G and if the addition of any edge to H results in a graph that does contain G. This idea is similar to the technique we have often used in assuming maximal counterexamples. For example, the proof of Ore's theorem (Theorem 5.1.1) used this approach.

Theorem 10.1.3 If $n \geq r+1$, then every $(n, ex(n, K_r)+1)$–graph G contains a $K_{r+1} - e$.

Proof. We proceed by induction on n, the order of G. For $n = r+1$, it is clear that having one more than the extremal number of edges forces $G = K_{r+1} - e$, and so we have the anchor step.

Now, we assume the result holds on all such graphs of order less than n and consider an $(n, ex(n; K_r)+1)$ graph G. Let x have minimum degree $\delta(G)$. Then it is easily seen that $\delta(G) \leq \delta(T_{n,r})$, and so $|E(G-x)| \geq ex(n-1; K_r)+1$. Hence, by induction we see that $G - x$ contains $K_{r+1} - e$, and the result holds. □

For completeness, we now state the following corollary to Turán's theorem.

Corollary 10.1.2 (Zarankiewicz [20]) If G^n is K_r-free, then

$$\delta(G^n) \leq (1 - \frac{1}{r-1})n = \frac{r-2}{r-1}n.$$

With considerable effort, one can improve upon the above corollary; however, we shall simply state this improvement.

Theorem 10.1.4 ([1]). If $\chi(G^n) \geq r$ and G^n is K_r-free, then

$$\delta(G^n) \leq \frac{(3r-7)}{(3r-4)} n.$$

We continue our investigation of complete subgraphs by counting triangles. Turán's theorem tells us when we can be sure one triangle exists, but our goal is to establish bounds on the number of triangles that exist in general. In first attacking this problem, we will find it useful to change the setting and sum the number of triangles that must be contained in a graph and its complement. Let $k_r(G)$ equal the number of $K_r's$ contained in the graph G. Independent work of several people, including Goodman [11], Moon and Moser [14] and Lovász [12] all lead to the following result.

Theorem 10.1.5 Given an (n, q)−graph G with $(n, \overline{q})$ complement $\overline{G}$,

$$k_3(G) + k_3(\overline{G}) = \binom{n}{3} - (n-2)q + \sum_{i=1}^{n} \binom{deg\ v_i}{2}$$

$$= \binom{n}{3} - (n-2)\overline{q} + \sum_{i=1}^{n} \binom{n-1-deg\ v_i}{2}.$$

Proof. Consider the degree sequence of G. There are $\sum_{i=1}^{n} \binom{deg\ v_i}{2}$ pairs of adjacent edges of G and $\sum_{i=1}^{n} \binom{n-1-deg\ v_i}{2}$ pairs of adjacent edges in $\overline{G}$. The sum of these two numbers can be counted in another way as well. Each of the triangles in G and $\overline{G}$ contains three pairs of adjacent edges, and each of the remaining

$$L = \binom{n}{3} - k_3(G) - k_3(\overline{G})$$

triples of vertices contains exactly one such pair. Hence,

$$\sum_{i=1}^{n} \binom{deg\ v_i}{2} + \sum_{i=1}^{n} \binom{n-1-deg\ v_i}{2} = 3k_3(G) + 3k_3(\overline{G}) + L.$$

Solving for our desired sum yields,

$$k_3(G) + k_3(\overline{G}) = \frac{1}{2}\left[\sum_{i=1}^{n} \binom{deg\ v_i}{2} + \sum_{i=1}^{n} \binom{n-1-deg\ v_i}{2} - \binom{n}{3}\right].$$

But note that

$$\sum_{i=1}^{n} \binom{n-1-deg\ v_i}{2} = \sum_{i=1}^{n} \frac{(n-1-deg\ v_i)(n-2-deg\ v_i)}{2}$$

$$= \sum_{i=1}^{n} \left\lfloor \binom{n-1}{2} - (n-2)deg\ v_i + \binom{deg\ v_i}{2} \right\rfloor.$$

Now, substituting and rearranging terms completes the result. □

Corollary 10.1.3 The graphs G^n and $\overline{G}^n$ contain a total of at least

$$\frac{n(n-1)(n-5)}{24} \quad \text{triangles.}$$

Theorem 10.1.6 An (n, q)–graph contains at least $\frac{q}{3n}(4q - n^2)$ triangles.

Proof. Suppose that $uv \in E$. Then there are at least $deg\ u + deg\ v - n$ vertices adjacent to both u and v. Thus, we see that

$$k_3(G) \geq \frac{1}{3} \sum_{uv \in E} (deg\ u + deg\ v - n).$$

But since each $deg\ u$ term appears $deg\ u$ times in this sum, we have that

$$k_3(G) \geq \frac{1}{3} \sum_{u \in V} deg^2\ u - nq.$$

So by the Cauchy inequality (see the appendix),

$$k_3(G) \geq \frac{1}{3}\left\{\frac{(2q)^2}{n} - nq\right\} = \frac{q}{3n}[4q - n^2]. \quad \square$$

The next result is due to Rademacher (see [9]) and extends Mantel's Theorem.

Theorem 10.1.7 For every even n, a graph on n vertices with $\frac{n^2}{4} + 1$ edges contains

at least $\frac{n}{2}$ triangles. Furthermore, this result is best possible.

The graph $K_{n/2,n/2} + e$ shows that Rademacher's Theorem is best possible. Our next result was originally conjectured by Nordhaus and Stewart [15]. The result is due to Bollobás [2].

Theorem 10.1.8 If G is a graph on n vertices and $\frac{n^2}{4} \leq |E(G)| \leq \frac{n^2}{3}$ edges then G contains at least $\frac{n}{9}(4|E(G)| - n^2)$ triangles.

Comparing these theorems over the range of possible values for q we see that Rademacher's Theorem is most acurate for $q = \frac{n^2}{4} + 1$ edges; the bounds of Moon and Moser and of Bollobás are equal when $q = \frac{n^2}{3}$; finally, Moon and Moser's Theorem yields the exact number of triangles when $q = \binom{n}{2}$.

We complete this section with a result due to Erdös [8]. We begin with a sequence of lemmas.

Lemma 10.1.1 Every $(n, ex(n-1, K_3) + 2)$–graph G which contains an odd cycle, contains a triangle.

Proof. Let G be as described and let C: $u_1, u_2, \ldots, u_{2k+1}$ be the vertices of a shortest odd cycle in G. We can assume that $3 < 2k + 1 \leq n$. Now $< u_1, u_2, \ldots, u_{2k+1} >$ can have no other edges, for otherwise a shorter odd cycle would be formed. Let $v_1, v_2, \ldots, v_{n-2k-1}$ be the other vertices of G. Any v_i $(1 \leq i \leq n - 2k - 1)$ can be adjacent to at most two u_j $(1 \leq j \leq 2k + 1)$, for otherwise an odd cycle shorter than C would be formed. Finally, Turá n's Theorem implies $< v_1, \ldots, v_{n-2k-1} >$ can have at most $ex(n - 2k - 1, K_3)$ edges. Thus, the number of edges in G is at most

$$2k + 1 + 2(n - 2k - 1) + ex(n - 2k - 1, K_3) \leq ex(n - 1, K_3) + 1,$$

contradicting our assumptions. □

Lemma 10.1.2 There exists a constant $c_2 > 0$ such that every $(n, ex(n, K_3) + 1)$

graph G contains at least $\lfloor c_2 n \rfloor$ triangles having a common edge (u, v).

Proof. Let $T = \{ (u_i, v_i, w_i) \mid 1 \le i \le r \}$ be a maximal system of disjoint triangles in G. Thus, in $G - T$ the remaining $n - 3r$ vertices contain no triangles and therefore have at most $ex(n - 3r, K_3)$ edges.

Denote by $G(i)$ the graph obtained from G by deleting the first $i - 1$ triangles of T. Further, let $deg_i u_i$, $deg_i v_i$ and $deg_i w_i$ be the degrees of u_i, v_i and w_i in $G(i)$.

We now show that for some i $(1 \le i \le r)$ we must have

$$deg_i u_i + deg_i v_i + deg_i w_i > n(1 + 9c_2) - 3i, \tag{1}$$

for if this failed to hold for any i, then the number of edges in G would be at most

$$\sum_{i=1}^{r} [\, n(1 + 9c_2) - 3i \,] + ex(n - 3r, K_3) < ex(n, K_3)$$

by a simple calculation for sufficiently small c_2. But this contradicts the fact G contains at least $ex(n, K_3) + 1$ edges. Thus, (1) holds for say $i = i_0$. Then a simple calculation shows that there are at least $3\lfloor c_2 n \rfloor$ vertices of $G(i_0)$ which are adjacent to more than one of the vertices $u_{i_0}, v_{i_0}, w_{i_0}$. Therefore, there are at least $\lfloor c_2 n \rfloor$ vertices adjacent to the same pair, which completes our proof. □

Lemma 10.1.3 Let $\delta > 0$ be a fixed number. Consider any (n, q)-graph G with $q > ex(n, K_3) - \frac{n}{2}(1 - \delta)$, $n > n_0(\delta)$, which contains a triangle. Then G contains an edge (u, v) and $r = \lfloor c_3 n \rfloor + 1$ $(c_3 = c_3(\delta))$ vertices w_i $(i = 1, 2, \ldots, r)$ so that all the triangles (u, v, w_i) $(i = 1, 2, \ldots, r)$ are in G.

Proof. By assumption, G contains a triangle (u, v, w). Assume first that

$$deg\ u + deg\ v + deg\ w > n(1 + 9c_3) + 9. \tag{2}$$

Then the result follows from Lemma 2.

If (2) fails to hold, then $G - u - v - w$ has $n - 3$ vertices and at least $q - n(1 + 9c_3) - 9$ edges. But if $c_3 < \frac{\delta}{18}$, then for $n > n_0$,

$$\begin{aligned} q - n(1 + 9c_3) - 9 &> ex(n, K_3) - \frac{n}{2}(1 - \delta) - n(1 + 9c_3) - 9 \\ &> ex(n - 3, K_3). \end{aligned}$$

But then by Lemma 10.1.2, $G - u - v - w$ contains the desired configuration of triangles, which completes the proof. □

We are finally ready to present our goal, a theorem due to Erdös [8].

Theorem 10.1.9 There exists a constant $c_1 > 0$ such that for n sufficiently large and $t < c_1 n/2$, if a graph G on n vertices contains at least $\lfloor \frac{n^2}{4} \rfloor + t$ edges, then G contains at least $t\lfloor \frac{n}{2} \rfloor$ triangles.

Proof. Suppose G is as above and $t < c_1 \frac{n}{2}$. We first assume that after the omission of any $r = \lfloor c_1 n/2c_3 \rfloor$ edges, the graph still contains a triangle. (Note: $c_3 = c_3(\delta)$, for $\delta = \frac{1}{4}$ in the last lemma.) For sufficiently small c_1, $\frac{c_1}{2c_3} < \frac{1}{4}$; thus it will be permissible to apply Lemma 10.1.3 during the omission of these edges.

By Lemma 10.1.3 (or Lemma 10.1.2) there exists an edge e_1 contained in $\lfloor c_3 n \rfloor + 1$ triangles of G. Again by Lemma 10.1.3 in $H_1 = G - e_1$, there exists an edge e_2 contained in at least $\lfloor c_3 n \rfloor + 1$ triangles of H_1. Suppose we have already chosen the edges $e_1, \ldots, e_r$ each of which is contained in at least $\lfloor c_3 n \rfloor + 1$ triangles. By our earlier assumption $H_r = G - e_1 - \cdots - e_r$ contains at least one triangle. But then by Lemma 10.1.3 there is an edge e_{r+1} in H_r which is contained in at least $\lfloor c_3 n \rfloor + 1$ triangles of H_r. These triangles incident on the edges $e_1, \ldots, e_{r+1}$ are clearly distinct, thus G contains at least

$$(r + 1)\{\lfloor c_3 n \rfloor + 1\} > c_1 \frac{n^2}{2} > t\frac{n}{2}$$

triangles, which completes the proof in this case.

Therefore, we may assume that there are $s \leq r < \frac{n}{4}$ edges $e_1, e_2, \ldots, e_s$ so that the graph $H = G - e_1 - e_2 - \cdots - e_s$ contains no triangles and we may assume s is the smallest integer with this property. By the fact that $s \leq r < \frac{n}{4}$, H has

$$ex(n, K_3) + t - s > ex(n, K_3) - \frac{n}{4} > ex(n - 1, K_3) + 1$$

edges. Thus, by Lemma 1, H must contain only even cycles.

By Theorem 10.1.1, $s \geq t$. Suppose $s = t$. Then H has $ex(n, K_3)$ edges and by Theorem 10.1.1, $H = T_{n,2}$. Clearly, the addition of any edge creates at least $\lfloor \frac{n}{2} \rfloor$ distinct triangles. A simple argument shows that the addition of every further edge introduces at least $\lfloor \frac{n}{2} \rfloor$ triangles and that these triangles are distinct. Thus, G contains at least $t\lfloor \frac{n}{2} \rfloor$ triangles and our result is shown in this case as well.

Finally assume $s = t + w$, $0 < w < \frac{n}{4}$ (since $s < n/4$). We also assume n is even, say $n = 2m$. Now since H contains only even cycles, it is a subgraph of a bipartite graph B whose vertices are say $\alpha_1, \ldots, \alpha_{m-u}$ and $\beta_1, \ldots, \beta_{m+u}$ (since H has more than $ex(2m, K_3) - \frac{m}{2}$ edges, we have $0 \leq u < (m/2)^{1/2}$).

Clearly, every one of the edges $e_1, \ldots, e_s$ join two of the α's or two of the β's, for otherwise for some e_i, the graph $G - e_1 - \cdots - e_{i-1} - e_{i+1} - \cdots - e_s$ would still have only even cycles and hence no triangles, which contradicts the minimum property of s.

By our assumption, H is a subgraph of B. Assume H is obtained from B by the omission of x edges. Then we clearly have

$$s = x + u^2 + t \quad (\text{or} \quad w = x + u^2),$$

and G is obtained from H by adding s edges $e_1, \ldots, e_s$ which are all of the form $(\alpha_{i_1}, \alpha_{i_2})$ or $(\beta_{i_1}, \beta_{i_2})$. Let $e_i = (\beta_{i_1}, \beta_{i_2})$ and let us estimate the number of triangles $(\beta_{i_1}, \beta_{i_2}, \alpha_j)$ in B. Clearly, at most x of the edges (β_{i_1}, α_j), (β_{i_2}, α_j) are not in B; thus $B + e_i$ contains at least $m - u - x$ triangles (if e_i connects two α's, then $B + e_i$ contains at least $m + u - x$ triangles). For different e_i's these triangles are clearly different; thus $G = H + e_1 + \cdots + e_s$ contains at least

$$(m - u - x)s = (m - u - x)(x + u^2 + t) \geq tm = t(n/2)$$

triangles. The above follows by simple computation from $s = u^2 + x + t < m/2$. The above equation completes the proof in the $n = 2m$ case. For $n = 2m + 1$ the proof is almost identical and hence we omit it here. This completes our proof. □

This result has been improved by Lovász and Simonovits (see [3]) who showed that the theorem holds for $c_1 = 1$.

Section 10.2 Cycles in Graphs

We now modify the forbidden subgraph in question and consider extremal results involving cycles of various sizes. Our first result is somewhat less specific than those we have seen thus far. Our concern is in finding a pair of vertex disjoint cycles of unspecified order. We define $s(n)$ to be the minimum number of edges so that every graph on n vertices contains two vertex disjoint cycles. The following result of Pósa [16] was presented in Chapter 5. We restate it here for completeness.

Theorem 10.2.1 For $n \geq 6$, $s(n) = 3n - 5$.

Despite finding $s(n)$ precisely, we find the previous result somewhat unsatisfying in that we do not have any information about the size of the cycles that must exist. We now turn to a line of investigation for cycles similar to the one taken for complete graphs. We begin with a bound on the size that will ensure that a graph contains a 4-cycle. Our strategy will be to count pairs of vertices that are dominated by a common vertex. Allowing only one such neighbor will prevent 4-cycles.

Theorem 10.2.2 Every graph on n vertices and $q > \frac{n}{4}(1 + \sqrt{4n-3})$ edges contains a 4-cycle.

Proof. Suppose G^n contains no 4-cycle. Our strategy will be to count pairs of vertices that are dominated by another vertex and in doing so, to create an upper bound on the number of edges in G^n.

For a fixed $z \in V$, there are $\binom{deg\ z}{2}$ pairs of vertices dominated by z. On the other hand, each pair x, y is counted at most once, since if it were counted twice, a 4-cycle would have to exist. Hence,

$$\sum_{z \in V} \binom{deg\ z}{2} \leq \binom{n}{2}.$$

Now, by Jensen's Inequality (see the appendix),

$$\binom{n}{2} \geq \sum_{z \in V} \binom{deg\ z}{2} \geq n\binom{2qn^{-1}}{2} = \frac{2q^2 - nq}{n}.$$

Thus,

$$q^2 - \frac{nq}{2} \leq \frac{n^3 - n^2}{4}$$

and hence,

$$q \leq \left(\frac{(4n^3 - 3n^2)}{16}\right)^{1/2} + \frac{n}{4}$$
$$= \left(\frac{n}{4}\right)(1 + \sqrt{4n - 3}).$$

Thus, the result follows. We note that we often say that the extremal value for C_4 is $O(n^{3/2})$; that is, q is bounded by a function that is on the order of $O(n^{3/2})$. □

In attempting to produce a sharp extremal example for the previous result, we come across our first difficulty. However, with the use of projective planes, we can produce a meaningful example. A *finite projective plane* is a special type of block design. That is, it is a family of v points and b subsets of these points such that each subset contains k of

the points and each pair of points occur together in exactly λ of the subsets. For finite projective planes, a more geometric view is taken and the parameters carry special restrictions. First of all, $k = n + 1$ and $\lambda = 1$. The sets are called *lines,* and the following four properties (defining properties of block designs) hold.

1. Any line is incident with $n + 1$ points.
2. Any point is incident with $n + 1$ lines.
3. Any pair of points are joined by exactly one line.
4. Any pair of lines intersect in exactly one point.

We can model projective planes as bipartite graphs as follows. Define the *point-line incidence graph G* of a projective plane of order *p*, where *p* is a prime as follows: The vertices of *G* correspond to the points and lines of the projective plane. A line is joined to each of the points that lie on the line. Thus, the point-line incidence graph is an $n + 1$ regular bipartite graph of order $2x = 2(n^2 + n + 1)$. Further, it can be shown (in the exercises) that this graph has no cycle smaller than a 6-cycle. In fact, it represents the $(n + 1)$-regular graphs of smallest order having this property. Further,

$$|E(G)| = x(n + 1) = \frac{x}{2}(1 + \sqrt{4x - 3}), \text{ where } x = \frac{|V(G)|}{2}.$$

For $n = 2$, the projective plane can be modeled as shown in Figure 10.2.1. Can you construct the point-line incidence graph for the projective plane of order 2?

We can find a natural setting in which the above examples do provide a set of extremal graphs. Suppose we consider a variation on the usual extremal question in which we consider only bipartite graphs rather than general graphs. It is natural to forbid only other bipartite graphs and, in particular, complete bipartite graphs. With this in mind, we define $ex(n, m; K_{s,t})$ to be the maximum number of edges in a bipartite graph with partite sets of orders $n \geq s$ and $m \geq t$ that does not contain a $K_{s,t}$. In particular, then, we first investigate forbidden C_4s; that is, we seek the extremal value $ex(n, n; K_{2,2})$.

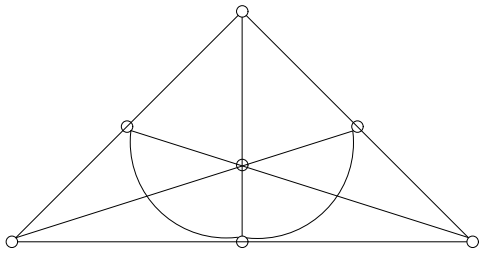


Figure 10.2.1. A model of the projective plane of order $p = 2$.

Theorem 10.2.3 For $n \geq 2$, $ex(n, n; K_{2,2}) \leq \frac{n}{2}(1 + \sqrt{4n - 3})$, and equality holds infinitely often.

Proof. Let $q = \frac{n}{2}(1 + \sqrt{4n - 3})$ and note that

$$(q - n)q = n^2(n - 1).$$

Now, suppose there is a bipartite graph $G_{n,n}$ of size greater than q that does not contain a $K_{2,2}$. Denote by $d_1, d_2, \ldots, d_n$ the degrees of the vertices in the first partite set V_1. Then,

$$\sum_{i=1}^{n} d_i = |E(G_{n,n})| = e > q$$

and

$$\binom{n}{2} \geq \sum_{i=1}^{n} \binom{d_i}{2} = \frac{1}{2}\sum_{i=1}^{n} {d_i}^2 - \frac{1}{2}\sum_{i=1}^{n} d_i$$

$$\geq \frac{1}{2n}e^2 - \frac{e}{2} > \frac{q(q-n)}{2n} = \binom{n}{2}.$$

Thus, we reach a contradiction and the bound now follows.

The previous example of the point-line incidence graph of the projective plane shows that equality holds infinitely often . □

We now consider the more general bipartite extremal problem. The following lemma will be useful in our computations. This approach follows Bollobás [3].

Lemma 10.2.1 Let m, n, s, t, r, k be integers with $2 \leq s \leq m$, $2 \leq t \leq n$, $k \geq 0$, $0 \leq r < m$ and $G_{m,n}$ of size $mx = km + r$ and with no $K_{s,t}$; then

$$m\binom{x}{t} \leq (m - r)\binom{k}{t} + r\binom{k+1}{t} \leq (s - 1)\binom{n}{t}.$$

Proof. Let $T \subseteq V_2$ with $|T| = t$ and say T is linked to $v \in V_1$ if $T \subseteq N(v)$. Then the number of t-sets linked to v is $\binom{deg\ v}{t}$. Since by assumption $G_{m,n}$ contains no $K_{s,t}$, each t-set in V_2 is linked to at most $s - 1$ vertices of V_1. Hence,

$$m\binom{x}{t} \leq \sum_{v \in V_1} \binom{deg\ v}{t} \leq (s-1)\binom{n}{t}.$$

Since $\sum_{v \in V_1} deg\ v = mx = km + r$, $0 \leq r < m$, and since $f(w) = \binom{w}{t}$ is convex, the inequality follows. □

Theorem 10.2.4 For $t \leq s$,

$$ex(m, n; K_{s,t}) \leq (s-1)^{1/t}\,(n - t + 1)m^{1-1/t} + (t-1)m.$$

Proof. Let $G_{m,n}$ be an extremal bipartite graph with $ex(m, n; K_{s,t}) = mx$ edges which is $K_{s,t}$-free. As $x < n$, the lemma implies that

$$(x - (t-1))^t \leq (s-1)(n - (t-1))^t m^{-1}.$$

Thus,

$$x - (t-1) \leq (s-1)^{1/t}\,(n - t + 1)\,m^{-1/t},$$

or

$$mx \leq (s-1)^{1/t}\,(n - t + 1)\,m^{1-1/t} + (t-1)m. \quad \square$$

Corollary 10.2.1 If $t \geq 2$ and c is a constant such that $c > (t-1)^{1/t}$ (t, c fixed), then for n sufficiently large,

$$ex(n, n; K_{t,t}) < cn^{2-1/t}.$$

We can use the bound in Theorem 10.2.4 to obtain a bound in general graphs.

Theorem 10.2.5 Let G be an (n, q)-graph that does not contain a $K_{s,t}$, $2 \leq s$, $2 \leq t$. Then

$$q \leq \frac{1}{2}\,ex(n, n\,;\, K_{s,t})$$

Proof. We construct a bipartite graph B from G as follows. Let V_1 and V_2 be copies of $V(G)$. For every edge xy in G, insert the edge from $x_1 \in V_1$ to $y_2 \in V_2$ (hence, two edges are placed into B). Then B has $2q$ edges and does not contain a $K_{s,t}$ and, hence, the result follows. □

Section 10.3 On the Structure of Extremal Graphs

In this section we begin an investigation of the structure of extremal graphs. After determining the extremal values of various forbidden graphs, it is natural to try to gain further information about the extremal graphs themselves. It is not surprising that a great deal can be said and that this information opens still other avenues of investigation. It should be noted that there is a fundamental difference between extremal problems in which one of the forbidden graphs is bipartite (called a *degenerate extremal problem*), and one where none of the graphs is bipartite. The reasons for this will become more apparent as we progress. For now, simply note that in the degenerate case $ex(n, H) = o(n^2)$, while in the nondegenerate case,

$$ex(n, H) \geq \left\lfloor \frac{n^2}{4} \right\rfloor.$$

The foundation for this section is primarily the work of Erdös and Stone [10]. Our goal is to show that for a class of graphs H, the extremal number $ex(n; H)$ depends only loosely on the graphs in H. That is, the exact structure of the forbidden subgraphs is not the critical issue, but rather the dominant feature is the minimum chromatic number of a graph in the class H. In what follows we use the notation $K_{(s)(t)}$ to mean the complete s-partite graph with t vertices in each partite set. We begin with two beautiful results from Erdös and Stone [10].

Theorem 10.3.1 Let $\varepsilon > 0$ and $k, t \geq 1$ be given. Then, for n sufficiently large, every graph of order n and with $\delta \geq (1 - \frac{1}{k} + \varepsilon)n$ contains $K_{(k+1)(t)}$.

Proof. (By induction on k). For $k = 1$, the statement claims that $\delta \geq \varepsilon n$ and, hence, G has at least $\frac{\varepsilon}{2}n^2$ edges. That G contains a $K_{t,t}$ follows from Corollary 10.2.1 and Theorem 10.2.5.

Now let $k \geq 2$ and $s = \lfloor \frac{1}{\varepsilon} t \rfloor$. If n is sufficiently large, then by our induction assumption, we can find a $K_{(k)(s)}$ in G. Let $Y = V(G) - K_{(k)(s)}$ and let X be those vertices of Y that are adjacent to at least t vertices in each of the partite sets of the $K_{(k)(s)}$. Then the number of missing edges between $Y - X$ and $K_{(k)(s)}$ is at least

$$\begin{aligned}(|Y| - |X|)(s - t) &\geq (|Y| - |X|)(1 - \varepsilon)s \\ &= (n - ks - |X|)\,(1 - \varepsilon)s.\end{aligned}$$

Also, the number of edges missing from any vertex in $K_{(k)(s)}$ is at most $(\frac{1}{k} - \varepsilon)n$. Thus, the number of edges missing from the vertices in $K_{(k)(s)}$ is at most

$$ks(\frac{1}{k} - \varepsilon)n = (1 - k\,\varepsilon)sn.$$

Thus, the preceding two inequalities imply that

$$(n - ks - |X|)(1 - \varepsilon)s \leq (1 - k\,\varepsilon)sn,$$

and solving we see that $|X| \geq \frac{\varepsilon(k-1)}{(1-\varepsilon)}\ n - ks.$

Since $k \geq 2$ and $\varepsilon > 0$, we see that $|X|$ grows large as n grows large. Then, if

$$|X| > \binom{s}{t}^k (t - 1)$$

we can select t vertices that will form the final partite set we desire. □

Theorem 10.3.2 Let G be a graph of order n with at least $(1 - \frac{1}{k} + \varepsilon)\ \frac{n^2}{2}$ edges. Then for n sufficiently large, G contains a $K_{(k+1)(t)}$.

Proof. Remove a vertex of degree less than $(1 - \frac{1}{k} + \frac{\varepsilon}{2})\ |V(G)|$ if any exist. Now, in the graph that remains, repeat this process and continue to repeat this process as often as possible. Suppose that at some point in this process we are unable to continue; that is, suppose we are left with a graph H in which all vertices have degree at least $(1 - \frac{1}{k} + \frac{\varepsilon}{2})\ |V(H)|$. Let $|V(H)| = N$; then if N is sufficiently large, the result will follow from our last theorem. Then, all that remains is for us to show that N cannot be "too small." That is, we wish to show that N is bounded below by a function that grows as n grows.

In the construction of H, the number of edges we removed is at most

$$\sum_{j=N+1}^{n} j(1 - \frac{1}{k} + \frac{\varepsilon}{2}) = \left(\binom{n+1}{2} - \binom{N+1}{2}\right)(1 - \frac{1}{k} + \frac{\varepsilon}{2})$$
$$\leq \left(\binom{n}{2} - \binom{N}{2}\right)(1 - \frac{1}{k} + \frac{\varepsilon}{2}) + (n - N).$$

The graph H has at most $\binom{N}{2}$ edges, and, thus,

$$(1 - \frac{1}{k} + \varepsilon) \binom{n}{2} \leq |E(G)|$$
$$\leq (1 - \frac{1}{k} + \frac{\varepsilon}{2}) [\binom{n}{2} - \binom{N}{2}]$$
$$+ (n - N) + \binom{N}{2}.$$

Thus,

$$\frac{\varepsilon}{2} \binom{n}{2} \leq (\frac{1}{k} - \frac{\varepsilon}{2}) \binom{N}{2} + (n - N).$$

Hence, we see that N grows large if n grows large.

Finally, to see that the process of removing vertices of small degree must stop, suppose that it does not stop and examine the sum on the number of edges removed (as we did above). In this case we would have at most $(1 - \frac{1}{k} + \frac{\varepsilon}{2}) \frac{n^2}{2}$ edges in G, a contradiction. Hence, the process must stop and the result is proved . □

The next, somewhat surprising result has been the goal of our work in this section. It tells us that the forbidden subgraph's structure is only somewhat responsible for the extremal number. That is, the exact structure of the graph is not as important as the chromatic number. The significance of the next result has lead to the following definition: Given a family of graphs F, the *subchromatic number* is defined to be

$$\Psi(F) = \min \{ \chi(G) : G \in F \} - 1.$$

The following result of Erdös and Simonovits [9] is an easy consequence of the Erdös - Stone Theorems.

Theorem 10.3.3 If F is a family of graphs with $\psi(F) = p$, then

$$ex(n, F) = (1 - \frac{1}{p}) \binom{n}{2} + o(n^2).$$

Proof. Since each $G \in F$ is not p-colorable, we see that G is not a subgraph of $T_{n,p}$. Hence,

$$ex(n, F) \geq |E(T_{n,p})| = (1 - \frac{1}{p}) \frac{n^2}{2} + O(n).$$

On the other hand, there is some $G_0 \in F$ with $\chi(G_0) = p + 1$ and say $|V(G_0)| = m$. Now the Erdös - Stone Theorem (10.3.2) asserts that

$$ex(n, K_{(p+1)(m)}) = (1 - \frac{1}{p}) \binom{n}{2} + o(n^2).$$

Since G_0 is a subgraph of $K_{(p+1)(m)}$, we have that

$$ex(n, F) \leq ex(n, K_{(p+1)(m)}) \leq (1 - \frac{1}{p} + o(1)) \binom{n}{2}.\square$$

The following is an immediate corollary.

Corollary 10.3.1

$$\lim_{n \to \infty} \frac{ex(n; G)}{n^2} = \frac{1}{2} (1 - \frac{1}{\chi(G) - 1}).$$

The structure of extremal graphs is fairly stable, in the sense that graphs that are nearly extremal (that is, do not contain the forbidden graph or graphs but have nearly as many edges as the extremal graphs) have a structure that is close to that of extremal graphs. That is, we need not make a great many changes in the edge set of a nearly extremal graph to obtain an extremal graph. This idea is expressed in our next result, the combined efforts of Erdös [6], [7] and Simonovits [18].

Theorem 10.3.5 (The First Stability Theorem) Let F be a family of forbidden graphs with subchromatic number p. For every $\varepsilon > 0$, there exists a $\delta > 0$ and an n_ε such that if G^n is F-free and if, for $n > n_\varepsilon$,

$$|E(G^n)| > ex(n; F) - \delta n^2,$$

then G^n can be obtained from $T_{n,p}$ by changing at most εn^2 edges.

The name "first stability theorem" clearly implies that there are others. Unfortunately, these results are beyond the scope of this text, but the interested reader is advised to see [17] and [3]. Our next theorem can be proven using the first stability theorem (10.3.5) and is due to the combined work of Erdös and Simonovits [6], [7] and [18].

For our next result we need the following idea. Consider a partition of the vertex set of G^n as say $S_1, \ldots, S_p$ and the p-partite graph $K_{s_1, \ldots, s_p}$ corresponding to this partition of $V(G^n)$, where $s_i = |S_i|$ An edge vw is called an *extra edge* if it is not in $K_{s_1, \ldots, s_p}$ but is in G^n (similarly, an edge is missing if it is in $K_{s_1, \ldots, s_p}$ but not in

G^n). For a given p, the partition $S_1, \ldots, S_p$ is *optimal* if the number of missing edges is minimum. Finally, for a given vertex v, let $b(v)$ denote the number of extra edges at v.

Theorem 10.3.6 (The Asymptotic Structure Theorem) Let F be a family of forbidden subgraphs with $\Psi(F) = p$. If S^n is any extremal graph for F, then it can be obtained from $T_{n,p}$ by deleting and adding at most $o(n^2)$ edges. Furthermore, if F is a finite family, then

$$\frac{\delta(S^n)}{n} = 1 - \frac{1}{p} + o(1).$$

Sketch of Proof. The first part of the theorem follows from Theorems 10.3.3 and the First Stability Theorem.

For the second part, consider an optimal partition $R_1, R_2, \ldots, R_p$ of $V(S^n)$ and assume R_1 has minimum order. Then, $| R_1 | \leq \frac{n}{p}$ and by the First Stability Theorem

$$\sum_{v \in R_1} b(v) = o(n^2).$$

If r denotes the maximum order of a graph in F, take r vertices $v_1, \ldots, v_r$ with $\sum_{i=1}^{r} b(v_i)$ minimum. Clearly for some $c > 0$, $| R_1 | > cn$. Thus,

$$\sum_{i=1}^{r} b(v_i) \leq \frac{r}{\lceil R_1 \rceil} \sum_{v \in R_1} b(v) = o(n).$$

Now apply symmetrization in a slightly modified form. For an arbitrary vertex v in S^n, delete all incident edges and join v to all vertices adjacent to each of $v_1, \ldots, v_r$. The resulting graph S^* contains no member of F. Further, $| E(S^n) | \leq | E(S^*) |$. Hence,

$$deg\ v = \left| \bigcap_{i=1}^{r} N(v_i) \right| \geq \left| \bigcup_{j=2}^{r} R_j \right| - \sum_{i=1}^{r} b(v_i) \geq n - \frac{n}{p} - o(n)$$

and the result follows. □

We next present an easy but useful result on the behavior of $ex(n; F)$.

Theorem 10.3.7 For every family F, $\dfrac{ex(n; F)}{\binom{n}{2}}$ is decreasing as $n \to \infty$.

Proof. For a fixed extremal graph H^m, take all $\binom{m}{n}$ subgraphs of order n, say

$G_1, \ldots, G_t$. Each edge of H^m is in $\binom{m-2}{n-2}$ of the G_is and, thus,

$$\binom{m-2}{n-2} | E(H^m) | \leq \sum_{i<t} | E(G_i) | \leq \binom{m}{n} | E(H^n) |$$

But this implies that

$$\frac{| E(H^m) |}{\binom{m}{2}} \leq \frac{| E(H^n) |}{\binom{n}{2}}. \quad \square$$

We finish our study of the structure of extremal graphs by trying to determine when the Turán graph is the extremal graph for a family of graphs F. We will see that $T_{n,p}$ is fundamental to extremal graphs.

To prove the next result, we use the technique of progressive induction. Essentially, the technique is as follows. For a given problem you are able to prove the inductive step under the assumptions of the inductive hypothesis. However, you are unable to prove the anchor step. (This could be because the anchor step is not true for small values.) It also appears that the proof of the anchor step is as difficult as a direct proof of the result. Thus, to establish the result, we define a function, say D, used to measure the "distance between our knowledge and the conjecture." We then attempt to show that the value of this measure must approach zero. We now use progressive induction to establish our next result. Once again this result is due to Simonovits [18].

Theorem 10.3.4 A family F has $T_{n,p}$ as an extremal graph (for n sufficiently large) if, and only if, some $G \in F$ has an edge e such that $p = \chi(G - e) = \Psi(F)$. Furthermore, if $T_{n,p}$ is extremal for F for infinitely many values of n, then it is the only extremal graph (again, provided n is sufficiently large).

Proof. One direction is easy, for if $\chi(G - e) \geq p + 1$ for every graph $G \in F$ and for every edge e of G, then the addition of one edge to $T_{n,p}$ cannot produce a graph that contains one of the forbidden graphs of F. If it did, that graph would necessarily have chromatic number p after the deletion of some edge. Thus, $T_{n,p}$ is not extremal for the family F.

For the other direction, we assume that $\chi(G) = p + 1$ but that $\chi(G - e) = p$. Further suppose that $\{E^n\}$ is a sequence of extremal graphs for the family F. Since $T_{n,p}$ contains none of the forbidden subgraphs, $|E(E^n)| \geq |E(T_{n,p})|$. We now define our measure of the "distance between our knowledge and the conjecture." Let

$$D(n) = |E(E^n)| - |E(T_{n,p})|.$$

In order to accomplish our goal we shall prove the following statement.

(*) For $n > n_0$, either $T_{n,p}$ is the only extremal graph or there is an $n' < n$ such that $D(n') > D(n)$ and $n' \to \infty$ as $n \to \infty$.

The implications of statement (*) are that for $n > n_0$, $T_{n,p}$ is the only extremal graph for the family F. In fact, for $n < n_0$, $D(n)$ is bounded by some constant c. Using (*), then, $D(n) \leq c$ for every n.

If we define N_i such that if $n > N_i$, then $n' > N_{i-1}$ and if $N_0 = n_0$, it is then easy to show by induction on i that for $n > N_i$, either $D(n) = 0$ or $D(n) \leq c - i$. Hence, for $n_1 = N_{k+1}$ (since $D(n) \geq 0$) $T_{n,p}$ is the only extremal graph.

To prove (*), we choose an arbitrary sequence of extremal graphs $\{E^n\}$ and applying Theorem 10.3.2 and the fact that

$$|E(E^n)| \geq |E(T_{n,p})|,$$

we know that $T_{pt,p} \subseteq E^n$. Similarly, we know that $T_{pt,p} \subseteq T_{n,p}$.

Let $S = E^n - T_{pt,p}$ and $T = T_{n,p} - T_{pt,p}$. Also, denote by e_S the number of edges from $T_{pt,p}$ to S and e_T the number of edges between $T_{pt,p}$ and T. It is easily seen that $e_T = (n - pt)(p - 1)t$.

Since E^n is extremal, $T_{pt,p}$ is an induced subgraph of E^n and each vertex of S is adjacent to at most $(p - 1)t$ vertices of $T_{pt,p}$. Therefore,

$$\begin{aligned} |E(E^n)| &= |E(T_{pt,p})| + e_s + |E(S)| \\ &\leq |E(T_{pt,p)}| + (n - pt)(p - 1)\ t + |E(S)|. \end{aligned}$$

Using the above we see that

$$\begin{aligned} D(n - pt) - D(n) &= [|E(S)| - |E(T)|] - [|E(E^n)| - |E(T_{n,p})|] \\ &= [|E(T_{n,p})| - |E(T)|] - [|E(E^n)| - |E(S)|] \\ &\geq e_T - e_S \\ &\geq 0. \end{aligned}$$

The only thing left for us to do is to check that if $D(n) = D(n - pt)$, then $E^n = T_{n,p}$. Clearly, each vertex of S is joined to exactly $p - 1$ classes of $T_{pt,p}$. We partition the vertices of E^n into p sets $A_1, \ldots, A_p$ by placing in A_j all vertices of E^n that are not adjacent to vertices in the jth partite set of $T_{pt,p}$. The vertices of A_j are clearly independent; otherwise, some $H \in F$ must be in E^n. Thus, E^n must be a p-colorable graph. Recall that it has at least as many edges as $T_{n,p}$. Therefore, we have that $E^n = T_{n,p}$, and the proof is complete. □

Exercises

1. Complete the arithmetic on $q_1 + q_2 + q_3$ in the proof of Theorem 10.1.1 that establishes the bound on $\lfloor E(G^n) \rfloor$.
2. Prove Theorem 10.1.2.
3. Provide a graph theoretic proof of the fact that the maximum number of edges in a complete d-partite graph of order n ($d \leq p - 1$) actually occurs in the graph $T_{n,\,p-1}$ (see Zykov's proof).
4. Prove Mantel's Theorem directly, without using Turan's theorem or proof. (Hint: Use a counting argument initially similar to that of Theorem 10.1.5).
5. Verify Corollary 10.1.2.
6. If G is a k-regular graph of order n then
$$k_3(G) + k_3(\overline{G}) = \binom{n}{3} - \frac{nk(n-k-1)}{2}.$$
7. Prove Corollary 10.1.3.
8. Let G be a graph of order n and let N_k denote the number of K_ks in G. Prove that
$$\frac{N_{k+1}}{N_k} \geq \frac{1}{k^2-1}\left(k^2 \frac{N_k}{N_{k+1}} - n\right).$$
9. Let G be a graph on mk vertices and more than $\binom{k}{2}m^2$ edges. Prove that G contains a K_{k+1}.
10. (Stronger version of Theorem 10.3.2) Let $k \geq 2$ be an integer and let $0 < \varepsilon < \frac{1}{2}(r-1)$. Then there exists a $d = d(\varepsilon, r) > 0$ such that if n is sufficiently large and
$$q > \left\{\frac{k-2}{k-1} + \varepsilon\right\}\frac{n^2}{2}$$
then every graph of order n and size q contains a $K_{k(t)}$ with $t \geq \lfloor d \log n \rfloor$. Hint: use induction, Theorem 10.2.4 with $s = 2$ and
$$\varepsilon_k = \tfrac{1}{2}\left\{\frac{k-2}{k-1} - \frac{k-3}{k-2}\right\} = \{2(k-1)(k-2)\}^{-1} > 0.$$
11. If $|E(G)| = \left(1 - \frac{1}{r}\right)\frac{n^2}{2}$, then $N_k \geq \binom{r}{k}\left(\frac{n}{k}\right)^k$, ($k \leq r + 1$, r real).
12. A strongly connected tournament on n vertices contains at least $\binom{n-1}{2}$ cycles.

13. A tournament of order n contains at least one and at most $n!/2^{\frac{n}{2}}$ hamiltonian paths.

14. Every tournament T of order n contains a transitive tournament of order at least $[\log_2 n] + 1$.

15. If T is a strong tournament, then the number of transitive subtournaments of order k ($k \geq 3$) is at most
$$\binom{n}{k} - \binom{n-2}{k-2}.$$

16. Suppose G is a (p, q) graph with at most $q = |E(T_{r,p})|$ but G is not $T_{r,p}$. Then G contains a subgraph H of order $r + p$ and size
$$\binom{r+p}{2} - p + 1.$$
That is, there are less than p edges missing from H.

17. Show that if G is a graph of order p with
$$\delta(G) \geq \left\lfloor \frac{(r-2)p}{(r-1)} \right\rfloor + 1,$$
then G contains a K_r.

18. Suppose that G has order $p \geq r + 1$ and size $q = ex(p; K_r) + 1$.
(a) Show that G contains two copies of K_r with $r - 1$ vertices in common.
(b) Show that for every k, $r \leq k \leq p$, G has a subgraph with k vertices and at least $ex(k, K_{r-1}) + 1$ edges.

19. Show that every graph of order $p \geq 5$ and size $q \geq \left\lfloor \frac{p^2}{4} \right\rfloor + 2$ contains two triangles with exactly one vertex in common.

20. Let $0 < a < a + \varepsilon < 1$. Then every graph G of order $n \geq 2a/\varepsilon$ and size at least $\frac{1}{2}(a + \varepsilon)n^2$ contains a subgraph H with $|V(H)| = h \geq (\frac{\varepsilon}{2})n$ and minimum degree at least ah.

21. Let $r \geq 2$, $1 \leq t \leq q$ and $N = n - (r-1)q \geq 1$. Let G be a graph of order n that contains a $K_{(r-1)(q)}$, but does not contain a $K_{r(t)}$. Then G has at most $e = ((r-1)q + t)N + 2qN^{1-\frac{1}{t}}$ edges of the form xy, where $x \in K_{(r-1)(q)}$ and $y \in G - K_{(r-1)(q)}$.

22. Let $F = K_{r(t)}$, where $r \geq 2$ and $t \geq 1$. Then the maximal size of a graph of order n without F is

$$ex(n; F) = \tfrac{1}{2}\{\frac{(r-2)}{(r-1)} + o(1)\}n^2.$$

23. Let $F_1, F_2, \ldots, F_j$ be non-empty graphs. Denote by r the minimum of the chromatic numbers of the F_i. Then the maximal size of a graph of order n not containing any of the F_i is

$$ex(n; F_1, \ldots, F_j) = \tfrac{1}{2}\{\frac{(r-2)}{(r-1)} + o(1)\}n^2.$$

References

1. Andrásfai, B., Erdös, P., and Sós, V. T., On the Connection Between Chromatic Number, Maximal Clique and Minimal Degree of a Graph. *Discrete Math.*, 8(1974), 205 – 218.
2. Bollobás, B., On Complete Subgraphs of Different Orders, *Math. Proc. Cambridge Phil. Soc.* 79(1976), 19 – 24.
3. Bollobás, B., *Extremal Graph Theory.* Academic Press, London (1978).
4. Dirac, G. A., Extensions of Turán's Theorem on Graphs. *Acta Math. Acad. Sci. Hungar.*, 14(1963), 417 – 422.
5. Erdös, P., On the Number of Complete Subgraphs and Circuits Contained in Graphs. *Casopis Pest. Mat.*, 94(1969), 290 – 296.
6. Erdös, P., Some Recent Results on Extremal Problems in Graph Theory. *Theory of Graphs*, ed. by Rosenstiehl, Gordon and Breach, New York (1967), 117 – 123.
7. Erdös, P., On Some New Inequalities Concerning Extremal Properties of Graphs. *Theory of Graphs*, ed. by Erdös and Katona, Academic Press, New York (1968), 77 – 81.
8. Erdös, P., On a Theorem of Rademacher-Turán, *Illinois J. Math.* 6(1962), 122 – 127.
9. Erdös, P., and Simonovits, M., A Limit Theorem in Graph Theory. *Studia Sci. Math. Hungar.* 1(1966), 51 – 57.
10. Erdös, P., and Stone, A. M., On the Structure of Linear Graphs. *Bull. Amer. Math. Soc.*, 52(1946), 1087 – 1091.
11. Goodman, A. W., On Sets of Acquaintances and Strangers at a Party. *Amer. Math. Monthly*, 66(1959), 778 – 783.
12. Lovász, L., On the Sieve Formula (in Hungarian). *Mat. Lapok*, 23(1972), 53 – 69.

13. Mantel, W., *Wiskundige Opgaven* 10(1906), 60 – 61.

14. Moon, J.W., and Moser, L., On a Problem of Turán. *Magyar Tud. Akad. Mat. Kutató Int. Közl.,* 7(1962), 283 – 286.

15. Nordhaus, E.A. and Stewart, B. M., Triangles in an Ordinary Graph, *Canad. J. Math.* 15(1963), 33 – 41.

16. Pósa, L. (See Erdös, P., Extremal Problems in Graph Theory. *A Seminar in Graph Theory*. Holt, Rinehart, and Winston, New York (1967).

17. Simonovits, M., Extremal Graph Theory. *Selected Topics in Graph Theory 2,* ed. by Beineke and Wilson, Academic Press, London (1983).

18. Simonovits, M., A Method for Solving Extremal Problems in Graph Theory. *Theory of Graphs,* ed. by Erdös and Katona, Academic Press, New York (1968), 279 – 319.

19. Turán, P., On an Extremal Problem in Graph Theory. *Mat. Fiz. Lapok*, 48(1941), 436 – 452.

20. Zarankiewicz, K., On a Problem of Turán Concerning Graphs. *Fund. Math.*, 41(1954), 137 – 145.

21. Zykov, A. A., On Some Properties of Linear Complexes. *Mat. Sbornik N. S.*, 24(66)(1949), 163 – 188.