Graph Theory: Penn State Math 485 Lecture Notes

Version 0.9.8

Christopher Griffin

🄯 2011

Licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License

With Contributions By:

Suraj Shekhar

Contents

List of Figures

Using These Notes

Stop! This is a set of lecture notes. It is not a book. Go away and come back when you have a real textbook on Graph Theory. Okay, do you have a book? Alright, let's move on then. This is a set of lecture notes for Math 485–Penn State's undergraduate Graph Theory course. Since I use these notes while I teach, there may be typographical errors that I noticed in class, but did not fix in the notes. If you see a typo, send me an e-mail and I'll add an acknowledgement. There may be many typos, that's why you should have a real textbook.

The lecture notes are loosely based on Gross and Yellen's *Graph Theory and It's Applications*, Bollobás' *Graph Theory*, Diestel's *Graph Theory*, Wolsey and Nemhauser's *Integer and Combinatorial Optimization*, Korte and Vygen's *Combinatorial Optimization* and several other books that are cited in these notes. All of the books mentioned are good books (some great). The problem is, they are either too complex for an introductory undergraduate course, have odd notation, do not focus enough on applications or focus too much on applications.

This set of notes correct some of the problems I mention by presenting the material in a format for that can be used easily in an undergraduate mathematics class. Many of the proofs in this set of notes are adapted from the textbooks with some minor additions. One thing that is included in these notes is a treatment of graph duality theorems from the perspective linear optimization. This is not covered in most graph theory books, while graph theoretic principles are not covered in many linear or combinatorial optimization books. I should note, Bondy and Murty discuss Linear Programming in their book *Graph Theory*, but it is clear they are not experts in optimization and their treatment is somewhat non sequitur, which is a shame. The best book on the topic of combinatorial optimization is by far Korte and Vygen's, who do cover linear programming in their latest edition. *Note:* Penn State has an expert in graph coloring problems, so there is no section on coloring in these notes, because I invited a guest lecturer who was the expert. I may add a section on graph coloring eventually.

In order to use these notes successfully, you should have taken a course in combinatorial proof (Math 311W at Penn State) and ideally matrix algebra (Math 220 at Penn State), though courses in Linear Programming (Math 484 at Penn State) wouldn't hurt. I review a substantial amount of the material you will need, but it's always good to have covered prerequisites before you get to a class. That being said, I hope you enjoy using these notes!

CHAPTER 1

Preface and Introduction to Graph Theory

1. Some History of Graph Theory and Its Branches

Graph Theory began with Leonhard Euler in his study of the Bridges of Königsburg problem. The city of Königsburg exists as a collection of islands connected by bridges as shown in Figure 1.1. The problem Euler wanted to analyze was: Is it possible to go from

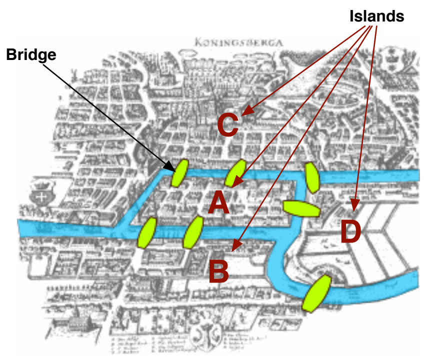


Figure 1.1. The city of Königsburg is built on a river and consists of four islands, which can be reached by means of seven bridges. The question Euler was interested in answering is: Is it possible to go from island to island traversing each bridge only once? (Picture courtesy of Wikipedia and Wikimedia Commons: http://en.wikipedia.org/wiki/File:Konigsberg_bridges.png)

island to island traversing each bridge only once? This was assuming that there was no trickery such as using a boat. Euler analyzed the problem by simplifying the representation. Assume that we treat each island as a dot and each bridge as a line (or curve) connecting these dots. The resulting representation is shown in Figure 1.2. We call the resulting structure a *graph*. We will formally define this in the next section, but for simplicity we can think of a graph as any number (usually finite) of dots connected by any number (also usually finite) of lines. The dots are called *vertices* or *nodes* and the lines are called *edges* or *links*. Two vertices are called *adjacent* if they are connected by an edge. Again, we'll formally define everything in the next section.

Note this representation dramatically simplifies the analysis of the problem in so far as we can now focus only on the structural properties of this graph. It's easy to see (from Figure 1.2) that each vertex has an odd number of edges attached to it. More importantly, since we are trying to traverse islands without ever recrossing the same bridge, when we enter an island (say C) we will use one of the three edges. Unless this is our final destination, we must use *another* edge to leave C, assuming we have not crossed all the bridges yet, we know we *must* leave C. That means that the third edge that touches C *must* be used to return

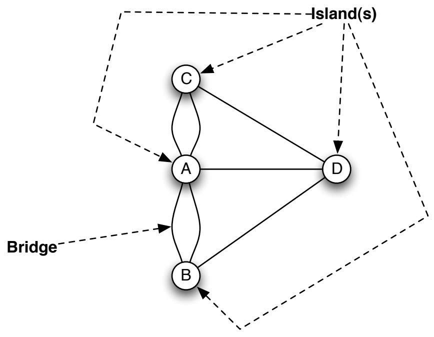


Figure 1.2. Representing each island as a dot and each bridge as a line or curve connecting the dots simplifies the visual representation of the seven Königsburg Bridges.

to C a final time. Alternatively, we could *start* at Island C and then return once and never come back. Put simply, our trip around the bridges of Königsburg had better start *or* end at Island C. But Islands (vertices) B and D *also* have this property. We can't start and end our travels over the bridges on Islands C, B and D simultaneously, therefore, no such walk around the islands in which we cross each bridge precisely once is possible.

Exercise 1. Since Euler's work two of the seven bridges in Königsburg have been destroyed (during World War II). Another two were replaced by major highways, but they are still (for all intents and purposes) bridges. The remaining three are still intact. Figure 1.3. Construct a graph representation of the new bridges of Königsburg and determine whether it

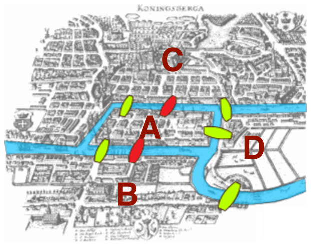


Figure 1.3. During World War II two of the seven original Königsburg bridges were destroyed. Later two more were made into modern highways (but they are still bridges). Is it now possible to go from island to island traversing each bridge only once? (Picture courtesy of Wikipedia and Wikimedia Commons: http://en.wikipedia.org/wiki/File:Konigsberg_bridges_presentstatus.png)

is possible to visit the bridges traversing each bridge exactly once. If so, find such a sequence of edges. [Hint: It might help to label the edges in your graph. You do not have to begin and end on the same island.]

Since Euler solved this very first problem in Graph Theory, the field has exploded, becoming one of the most important areas of applied mathematics we currently study. Generally speaking, Graph Theory is a branch of Combinatorics but it is closely connected to Applied Mathematics, Optimization Theory and Computer Science. At Penn State (for example) if you want to start a bar fight between Math and Computer Science (and possibly Electrical Engineering) you might claim that Graph Theory belongs (rightfully) in the Math

Department. (This is only funny because there is a strong group of graph theorists in our Computer Science Department.) In reality, Graph Theory is cross-disciplinary between Math, Computer Science, Electrical Engineering and Operations Research[1]. Here are some of the subjects within Graph Theory that are of interest to people in these disciplines:

(1) Optimization Problems on Graphs: Problems of optimization on graphs generally treat a graph structure like a road network and attempt to maximize flow along that network while minimizing costs. There are many classical optimization problems associated to graphs and this field is sometimes considered a sub-discipline within Combinatorial Optimization.
(2) Topological Graph Theory: Asks questions about methods of embedding graphs into topological spaces (like $\mathbb{R}^2$ or on the surface of a torus) so that certain properties are maintained. For example, the question of planarity asks: Can a graph be drawn on the plane in such a way so that *no two edge* cross. Clearly, the bridges of Königsburg graph had that property, but not all graphs do.
(3) Graph Coloring: A question related both to optimization and to planarity asks how many colors does it take to color each vertex (or edge) of a graph so that no two adjacent vertices have the same color. Attempting to obtain a coloring of a graph has several applications to scheduling and computer science.
(4) Analytic Graph Theory: Is the study of randomness and probability applied to graphs. Random graph theory is a subset of this study. In it, we assume that a graph is drawn from a probability distribution that returns graphs and we study the properties that certain distributions of graphs have.
(5) Algebraic Graph Theory: Is the application of abstract algebra (sometimes associated with matrix groups) to graph theory. Many interesting results can be proved about graphs when using matrices and other algebraic properties.

Obviously this is not a complete list of all the various problems and applications of Graph Theory. However, this is a list of some of the things we may touch on in the class. The textbook [**GY05**] is a good place to start on some of these topics. Another good source is [**BM08**], which I used for some of these notes. [**Bol01**] and [**Bol00**] are classics by one of the absolute masters of the field Bollobás and Diestel's [**Die10**] book is a pleasant read (it actually used to be much shorter). For the combinatorial optimization element of graph theory, turn to Nemhauser and Wolsey [**WN99**] as well as the second part of Bazarra et al.'s Linear Programming and Network Flows [**BJS04**]. Another reasonable book is [**PS98**], though it's a bit older, it's much less expensive than the others. In that same theme, [**Tru94**] and [**Cha84**] are also inexpensive little introductions to Graph Theory that are not as comprehensive as Gross and Yellen or Bondy and Murty, but they are nice to have in one's library for easier reading. In particular, [**Cha84**] spends a lot of time on applications rather than theory.

2. A Little Note on Network Science

If this were a real book, I'd never be able to add this section, but since these are lecture notes (and supposed to be educational) it's worth talking for a minute about *Network Science*. Network Science is one of these *interdisciplinary* terms that seems to be appearing everywhere

[1]See my note on *Network Science* below.

and it seems to be used by anyone who is not a formal mathematician or computer scientist to describe his/her work in some application of graph theory. [**New03**, **NBW06**] are good places to look to see what was popular in the field a few years ago. There are two opinions on Network Science that I've heard so far:

(1) this work is all so brilliant, new and exciting and will change the world *or*
(2) this is as old as the hills and is just a group of physicists reinterpreting classical results in graph theory and mixing in econometrics-style experiments.

Reality, I hope, is somewhere in between. There is a certain amount of redundancy from older work going on in *Network Science*. For example, Simon [**Sim55**] presaged and surpassed some of the work in the seminal Network Science literature [**AB00**, **AB02**, **BAJB00**, **BAJ99**] and Alderson et al. [**Ald08**] do correctly point out that there is a misinterpretation behind the mechanisms of network formation, especially man-made networks. On the other hand, some questions being asked by the Network Science community are new, useful and highly interdisciplinary such as detecting membership or multiple memberships in online communities (see e.g., [**FCF11**]) or understanding the spread of pathogens on specialized networks [**PSV01**, **GB06**]. It will be interesting to see where this interdisciplinary research goes in the long term. Hopefully, the results will justify the hype currently surrounding the discipline. Ideally these notes will help you decide what is really novel and exciting and what is just over-hyped nonsense.

CHAPTER 2

Some Definitions and Theorems

1. Graphs, Multi-Graphs, Simple Graphs

DEFINITION 2.1 (Graph). A graph is a tuple $G = (V, E)$ where V is a (finite) set of vertices and E is a finite collection of edges. The set E contains elements from the union of the one and two element subsets of V. That is, each edge is either a one or two element subset of V.

DEFINITION 2.2 (Self-Loop). If $G = (V, E)$ is a graph and $v \in V$ and $e = \{v\}$, then edge e is called a *self-loop*. That is, any edge that is a single element subset of V is called a self-loop.

DEFINITION 2.3 (Vertex Adjacency). Let $G = (V, E)$ be a graph. Two vertices v_1 and v_2 are said to be *adjacent* if there exists an edge $e \in E$ so that $e = \{v_1, v_2\}$. A vertex v is self-adjacent if $e = \{v\}$ is an element of E.

DEFINITION 2.4 (Edge Adjacency). Let $G = (V, E)$ be a graph. Two edges e_1 and e_2 are said to be *adjacent* if there exists a vertex v so that v is an element of both v_1 and v_2 (as sets). An edge e is said to be *adjacent* to a vertex v if v is an element of e as a set.

DEFINITION 2.5 (Neighborhood). Let $G = (V, E)$ be a graph and let $v \in V$. The *neighbors* of v are the set of vertices that are adjacent to v. Formally:

$$N(v) = \{u \in V : \exists e \in E\,(e = \{u, v\} \text{ or } u = v \text{ and } e = \{v\})\} \tag{2.1}$$

In some texts, $N(v)$ is called the *open neighborhood* of v while $N[v] = N(v) \cup \{v\}$ is called the *closed neighborhood* of v. This notation is somewhat rare in practice. When v is an element of more than one graph, we write $N_G(v)$ as the neighborhood of v in graph G.

REMARK 2.6. Expression 2.1 is read

> $N(v)$ is the set of vertices u in (the set) V such that there exists an edge e in (the set) E so that $e = \{u, v\}$ or $u = v$ and $e = \{v\}$.

The logical expression $\exists x\,(R(x))$ is always read in this way; that is, there exists x so that some statement $R(x)$ holds. Similarly, the logical expression $\forall y\,(R(y))$ is read:

> For all y the statement $R(y)$ holds.

Admittedly this sort of thing is very pedantic, but logical notation can help immensely in simplifying complex mathematical expressions[1].

[1] When I was in graduate school, I always found Real Analysis to be somewhat mysterious until I got used to all the ϵ's and δ's. Then I took a bunch of logic courses and learned to manipulate complex logical expressions, how they were classified and how mathematics could be built up out of Set Theory. Suddenly, Real Analysis (as I understood it) became very easy. It was all about manipulating logical sentences about those ϵ's and δ's and determining when certain logical statements were equivalent. The moral of the story: if you want to learn mathematics, take a course or two in logic.

Example 2.7. Consider the set of vertices $V = \{1, 2, 3, 4\}$. The set of edges

$$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}\}$$

Then the graph $G = (V, E)$ has four vertices and four edges. It is usually easier to represent this graphically (as we did for the Bridges of Königsberg Problem). See Figure 2.1 for the visual representation of G. These visualizations are constructed by representing each

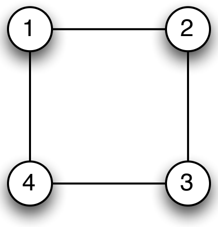


Figure 2.1. It is easier for explanation to represent a graph by a diagram in which vertices are represented by points (or squares, circles, triangles etc.) and edges are represented by lines connecting vertices.

vertex as a point (or square, circle, triangle etc.) and each edge as a line connecting the vertex representations that make up the edge. That is, let $v_1, v_2 \in V$. Then there is a line connecting the points for v_1 and v_2 if and only if $\{v_1, v_2\} \in E$.

In this example, the neighborhood of Vertex 1 is Vertices 2 and 4 and Vertex 1 is adjacent to these vertices and has degree 2.

Definition 2.8 (Degree). Let $G = (V, E)$ be a graph and let $v \in V$. The *degree* of v, written $\deg(v)$ is the number of non-self-loop edges adjacent to v plus two times the number of self-loops defined at v. More formally:

$$\deg(v) = |\{e \in E : \exists u \in V(e = \{u, v\})\}| + 2\,|\{e \in E : e = \{v\}\}|$$

Here if S is a set, then $|S|$ is the cardinality of that set.

Example 2.9. If we replace the edge set in Example 2.7 with:

$$E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}, \{1\}\}$$

then the visual representation of the graph includes a loop that starts and ends at Vertex 1. This is illustrated in Figure 2.2. In this example the degree of Vertex 1 is now 4. We obtain this by counting the number of non self-loop edges adjacent to Vertex 1 (there are 2) and adding two times the number of self-loops at Vertex 1 (there is 1) to obtain $2 + 2 \times 1 = 4$.

Definition 2.10 (MultiGraph). A graph $G = (V, E)$ is a *multigraph* if there are two edges e_1 and e_2 in E so that e_1 and e_2 are equal as sets. That is, there are two vertices v_1 and v_2 in V so that $e_1 = e_2 = \{v_1, v_2\}$.

Remark 2.11. Note in the definition of graph (Definition 2.1) we were very careful to specify that E is a *collection* of one and two element subsets of V rather than to say that E was, itself, a set. This allows us to have duplicate edges in the edge set and thus to define multigraphs. In Computer Science a set that may have duplicate entries is sometimes called a *multiset*. A multigraph is a graph in which E is a multiset.

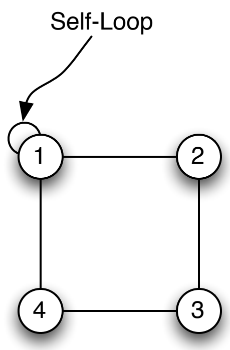


Figure 2.2. A self-loop is an edge in a graph G that contains exactly one vertex. That is, an edge that is a one element subset of the vertex set. Self-loops are illustrated by loops at the vertex in question.

EXAMPLE 2.12. Consider the graph associated with the Bridges of Königsburg Problem (see Figure 2.3). The vertex set is $V = \{A, B, C, D\}$. The edge collection is:

$$E = \{\{A, B\}, \{A, B\}, \{A, C\}, \{A, C\}, \{A, D\}, \{B, D\}, \{C, D\}\}$$

This multigraph occurs because there are two bridges connecting island A with island B and two bridges connecting island A with island C. If two vertices are connected by two (or more) edges, then the edges are simply represented as parallel lines (or arcs) connecting the vertices.

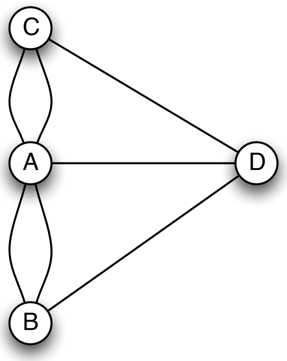


Figure 2.3. A multigraph is a graph in which a pair of nodes can have more than one edge connecting them. When this occurs, the for a graph $G = (V, E)$, the element E is a collection or *multiset* rather than a set. This is because there are duplicate elements (edges) in the structure.

REMARK 2.13. Let $G = (V, E)$ be a graph. There are two degree values that are of interest in graph theory: the largest and smallest vertex degrees usually denoted $\Delta(G)$ and $\delta(G)$. That is:

$$\Delta(G) = \max_{v \in V} \deg(v) \tag{2.2}$$

$$\delta(G) = \min_{v \in V} \deg(v) \tag{2.3}$$

Remark 2.14. Despite our initial investigation of The Bridges of Königsburg Problem as a mechanism for beginning our investigation of graph theory, most of graph theory is not concerned with graphs containing either self-loops *or* multigraphs.

Definition 2.15 (Simple Graph). A graph $G = (V, E)$ is a simple graph if G has no edges that are self-loops and if E is a *subset* of two element subsets of V; i.e., G is not a multi-graph.

Remark 2.16. In light of Remark 2.14, we will assume that every graph we discuss in these notes is a *simple* graph and we will use the term *graph* to mean *simple graph.* When a particular result holds in a more general setting, we will state it explicitly.

Exercise 2. Consider the new Bridges of Königsburg Problem from Exercise 1. Is the graph representation of this problem a simple graph? Could a self-loop exist in a graph derived from a Bridges of Königsburg type problem? If so, what would it mean? If not, why?

Exercise 3. Prove that for simple graphs the degree of a vertex is simply the cardinality of its (open) neighborhood.

2. Directed Graphs

Definition 2.17 (Directed Graph). A directed graph (digraph) is a tuple $G = (V, E)$ where V is a (finite) set of vertices and E is a collection of elements contained in $V \times V$. That is, E is a collection of ordered pairs of vertices. The edges in E are called *directed edges* to distinguish them from those edges in Definition 2.1

Definition 2.18 (Source / Destination). Let $G = (V, E)$ be a directed graph. The *source* (or *tail*) of the (directed) edge $e = (v_1, v_2)$ is v_1 while the *destination* (or *sink* or *head*) of the edge is v_2.

Remark 2.19. A directed graph (digraph) differs from a graph only insofar as we replace the concept of an edge as a set with the idea that an edge as an ordered pair in which the ordering gives some notion of direction of flow. In the context of a digraph, a *self-loop* is an ordered pair with form (v, v). We can define a multi-digraph if we allow the set E to be a true collection (rather than a set) that contains multiple copies of an ordered pair.

Remark 2.20. It is worth noting that the ordered pair (v_1, v_2) is *distinct* from the pair (v_2, v_1). Thus if a digraph $G = (V, E)$ has both (v_1, v_2) and (v_2, v_1) in its edge set, it is *not* a multi-digraph.

Example 2.21. We can modify the figures in Example 2.7 to make it directed. Suppose we have the directed graph with vertex set $V = \{1, 2, 3, 4\}$ and edge set:

$$E = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$$

This digraph is visualized in Figure 2.4(a). In drawing a digraph, we simply append arrowheads to the destination associated with a directed edge.

We can likewise modify our self-loop example to make it directed. In this case, our edge set becomes:

$$E = \{(1, 2), (2, 3), (3, 4), (4, 1), (1, 1)\}$$

This is shown in Figure 2.4(b).

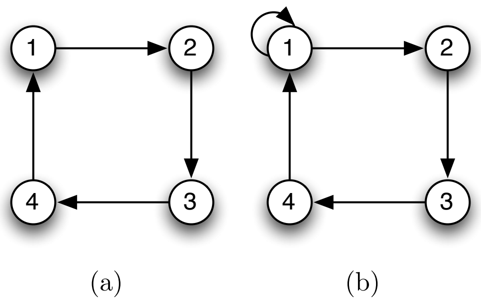


Figure 2.4. (a) A directed graph. (b) A directed graph with a self-loop. In a directed graph, edges are directed; that is they are ordered pairs of elements drawn from the vertex set. The ordering of the pair gives the direction of the edge.

Example 2.22. Consider the (simple) graph from Example 2.7. Suppose that the vertices represent islands (just as they did) in the Bridges of Königsburg Problem and the edges represent bridges. It is very easy to see that a tour of these islands exists in which we cross each bridge exactly once. (Such a tour might start at Island 1 then go to Island 2, then 3, then 4 and finally back to Island 1.)

Now suppose the islands pass an ordinance declaring all bridges are to be one-way. Then we might end up with a graphical representation like the one shown in Figure 2.4(a). If we now restrict the direction our island tours can take so that they obey the law, we see that our tour is still valid, but a tour starting at Island 1 and going to Island 4 would be illegal. Obviously, it is in the Islands' best interest to make sure that cars (or people) can both enter the island *and* leave the island. Thus, we might ask is there any way to directionalize these edges (i.e., make the bridges one-way) so that a tour is not possible *and* so that one can reach each island by car assuming that the given island is not the starting point?

It is easy to see the answer is *no*, but to prove it, consider the following: Island 1 has two adjacent edges. In order to be able to reach Island 1, one of these edges must have Island 1 as a source and the other edge must have Island 1 as a destination. This constrains the directions that the edges that are adjacent to Vertices 2 and 4 may take, which in turn completely specifies the directions for Vertex 3 (See Figure 2.5) We can now apply a symmetry argument. Each of those vertices are identical (in essence) in the way their adjacent edges look and behave. Thus, it is clear we cannot directionalize the edges (make the bridges one way) so that we both destroy our ability to tour the islands and cross each bridge only once **and** make it possible to enter and leave each island by a bridge. If two islands (say Islands 1 and 3) became xenophobic and made all its bridges one way so that it was only possible to leave the islands, then a tour of all the bridges would be impossible.

Definition 2.23 (Underlying Graph). If $G = (V, E)$ is a digraph, then the underlying graph of G is the (multi) graph (with self-loops) that results when each directed edge (v_1, v_2) is replaced by the set $\{v_1, v_2\}$ thus making the edge non-directional. Naturally if the directed edge is a directed self-loop (v, v) then it is replaced by the singleton set $\{v\}$.

Remark 2.24. Notions like edge and vertex adjacency and neighborhood can be extended to digraphs by simply defining them with respect to the underlying graph of a digraph. Thus the neighborhood of a vertex v in a digraph G is $N(v)$ computed in the underlying graph.

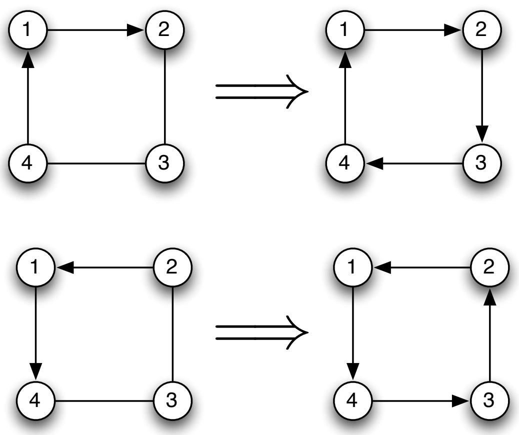


Figure 2.5. There are only two ways to directionalize the edges adjacent to Island 1 so that it's possible to both enter and leave the island. These choices then constrain the other edge directions, assuming that we wish to be able to enter and leave all the islands. The symmetry in the underlying graph ensures that it is not possible to **practically** make the bridges one way and destroy our ability to tour the islands.

Remark 2.25. It is possible to mix (undirected) edges and directed edges together into a very general definition of a graph with both undirected and directed edges. Situations requiring such a model *almost* never occur in modeling and when they do, the undirected edges with form $\{v_1, v_2\}$ are usually replaced with a pair of directed (v_1, v_2) and (v_2, v_1). Thus, for remainder of these notes, unless otherwise stated:

(1) When we say *graph* we will mean *simple graph* as in Remark 2.16. If we intend the result to apply to any graph we'll say a general graph.
(2) When we say *digraph* we will mean a directed graph $G = (V, E)$ in which every edge is a directed edge and the component E is a *set* and in which there are *no self-loops.*

Exercise 4. Suppose in the New Bridges of Königsburg (from Exercise 1) some of the bridges are to become *one way.* Find a way of replacing the edges in the graph you obtained in solving Exercise 1 with directed edges so that the graph becomes a digraph but so that it is still possible to tour all the islands without crossing the same bridge twice. Is it possible to directionalize the edges so that a tour in which each bridge is crossed once is not possible but it is still possible to enter and exit each island? If so, do it. If not, prove it is not possible. [Hint: In this case, enumeration is not that hard and its the most straight-forward. You can use symmetry to shorten your argument substantially.]

3. Elementary Graph Properties: Degrees and Degree Sequences

Definition 2.26 (Empty and Trivial Graphs). A graph $G = (V, E)$ in which $V = \emptyset$ is called the *empty* graph (or *null* graph). A graph in which $V = \{v\}$ and $E = \emptyset$ is called the *trivial* graph.

Definition 2.27 (Isolated Vertex). Let $G = (V, E)$ be a graph and let $v \in V$. If $\deg(v) = 0$ then v is said to be *isolated.*

Remark 2.28. Note that Definition 2.27 applies only when G is a simple graph. If G is a general graph (one with self-loops) then v is still isolated even when $\{v\} \in E$, that is there is a self-loop at vertex v and no other edges are adjacent to v. In this case, however, $\deg(v) = 2$.

Definition 2.29 (Degree Sequence). Let $G = (V, E)$ be a graph with $|V| = n$. The *degree sequence* of G is a tuple $\mathbf{d} \in \mathbb{Z}^n$ composed of the degrees of the vertices in V arranged in *decreasing order*.

Example 2.30. Consider the graph in Figure 2.6. The degrees for the vertices of this graph are:

(1) $\mathrm{v}_1 = 4$
(2) $\mathrm{v}_2 = 3$
(3) $\mathrm{v}_3 = 2$
(4) $\mathrm{v}_4 = 2$
(5) $\mathrm{v}_5 = 1$

This leads to the degree sequence: $\mathbf{d} = (4, 3, 2, 2, 1)$.

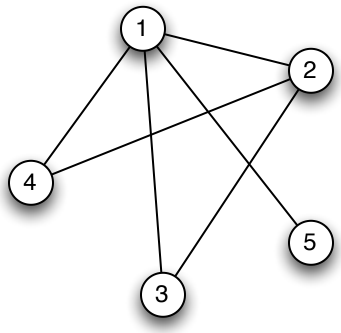


Figure 2.6. The graph above has a degree sequence $\mathbf{d} = (4, 3, 2, 2, 1)$. These are the degrees of the vertices in the graph arranged in increasing order.

Assumption 1 (Pigeonhole Principle). Suppose items may be classified according to m possible types and we are given $n > m$ items. Then there are at least two items with the same type.

Remark 2.31. The Pigeonhole Principle was originally formulated by thinking of placing $m+1$ pigeons into m pigeon holes. Clearly to place all the pigeons in the holes, one hole has two pigeons in it. The holes are the types (each whole is a different type) and the pigeons are the objects. Another good example deals with gloves: There are two types of gloves (left handed and right handed). If I hand you three gloves (the objects), then you either have two left-handed gloves or two right-handed gloves.

Theorem 2.32. *Let $G = (V, E)$ be a non-empty, non-trivial graph. Then G has at least one pair of vertices with equal degree.*

Proof. This proof uses the Pigeonhole Principal and is illustrated by the graph in Figure 2.6, where $\deg(v_3) = \deg(v_4)$. The types will be the possible vertex degree values and the objects will be the vertices.

Suppose $|V| = n$. Each vertex *could* have a degree between 0 and $n-1$ (for a total of n possible degrees), but the graph has a vertex of degree 0 if and only if it does not have a vertex of degree $n-1$. Therefore, there are only at most $n-1$ possible degree values depending on whether the graph has an isolated vertex or a vertex with degree $n-1$ (if it has neither, there are even fewer than $n-1$ possible degree values). Thus by the Pigeonhole Principal, at least two vertices must have the same degree. □

Theorem 2.33. *Let $G = (V, E)$ be a (general) graph then:*

$$2|E| = \sum_{v \in V} \deg(v) \tag{2.4}$$

Proof. Consider two vertices v_1 and v_2 in V. If $e = \{v_1, v_2\}$ then a $+1$ is contributed to $\sum_{v\in V} \deg(v)$ for both v_1 and v_2. Thus every non-self-loop edge contributes $+2$ to the vertex degree sum. On the other hand, if $e = \{v_1\}$ is a self-loop, then this edge contributes $+2$ to the degree of v_1. Therefore, each edge contributes exactly $+2$ to the vertex degree sum. Equation 2.4 follows immediately. □

Corollary 2.34. *Let $G = (V, E)$. Then there are an even number of vertices in V with odd degree.*

Exercise 5. Prove Corollary 2.34.

Definition 2.35 (Graphic Sequence). Let $\mathbf{d} = (d_1, \dots, d_n)$ be a tuple in $\mathbb{Z}^n$ with $d_1 \geq d_2 \geq \cdots \geq d_n$. Then $\mathbf{d}$ is *graphic* if there exists a graph G with degree sequence $\mathbf{d}$.

Corollary 2.36. *If $\mathbf{d}$ is graphic, then the sum of its elements is even.*

Exercise 6. Prove Corollary 2.36.

Lemma 2.37. *Let $\mathbf{d} = (d_1, \dots, d_n)$ be a graphic degree sequence. Then there exists a graph $G = (V, E)$ with degree sequence $\mathbf{d}$ so that if $V = \{v_1, \dots, v_n\}$ then:*

1. *$\deg(v_i) = d_i$ for $i = 1, \dots, n$ and*
2. *v_1 is adjacent to vertices $v_2, \dots, v_{d_1+1}$.*

Proof. The fact that $\mathbf{d}$ is graphic means there is at least one graph whose degree sequence is equal to $\mathbf{d}$. From among all those graphs, chose $G = (V, E)$ to maximize

$$r = |N(v_1) \cap \{v_2, \dots, v_{d_1+1}\}| \tag{2.5}$$

Recall that $N(v_1)$ is the neighborhood of v_1. Thus maximizing Expression 2.5 implies we are attempting to make sure that as many vertices in the set $\{v_2, \dots, v_{d_1+1}\}$ are adjacent to v_1 as possible.

If r $= d_1$, then the theorem is proved since v_1 is adjacent to $v_2, \dots, v_{d_1+1}$. Therefore we'll proceed by contradiction and assume $r < d_1$. We know the following things:

1. Since $\deg(v_1) = d_1$ there must be a vertex v_t with $t > d_1 + 1$ so that v_t is adjacent to v_1.
2. Moreover, there is a vertex v_s with $2 \leq s \leq d_1 + 1$ that is *not* adjacent to v_1.
3. By the ordering of V, $\deg(v_s) \geq \deg(v_t)$; that is $d_s \geq d_t$.

(4) Therefore, there is some vertex $v_k \in V$ so that v_s is adjacent to v_k but v_t is not because v_t is adjacent to v_1 and v_s is not and the degree of v_s is at least as large as the degree of v_t.

Let us create a new graph $G' = (V, E')$. The edge set E' is constructed from E by:

(1) Removing edge $\{v_1, v_t\}$.
(2) Removing edge $\{v_s, v_k\}$.
(3) Adding edge $\{v_1, v_s\}$.
(4) Adding edge $\{v_t, v_k\}$.

This is ilustrated in Figure 2.7. In this construction, the degrees of v_1, v_t, v_s and v_k are

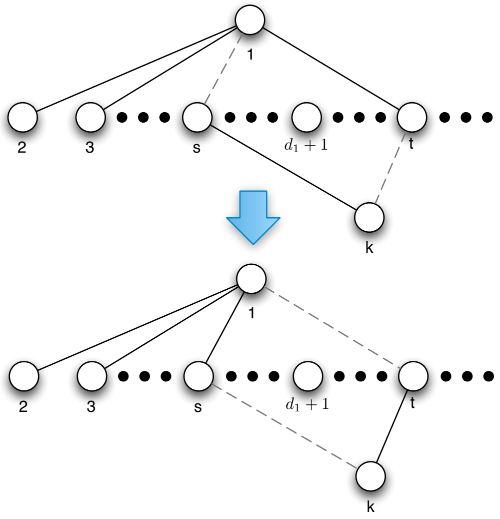


Figure 2.7. We construct a new graph G' from G that has a larger value r (See Expression 2.5) than our original graph G did. This contradicts our assumption that G was chosen to maximize r.

preserved. However, it is clear that in G':

$$r' = |N_{G'}(v_1) \cap \{v_2, \ldots, v_{d_1+1}\}|$$

and we have $r' > r$. This contradicts our initial choice of G and proves the theorem. $\square$

THEOREM 2.38 (Havel-Hakimi Theorem). *A degree sequence* $\mathbf{d} = (d_1, \ldots, d_n)$ *is graphic if and only if the sequence* $(d_2 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ *is graphic.*

PROOF. ($\Rightarrow$) Suppose that $\mathbf{d} = (d_1, \ldots, d_n)$ is graphic. Then by Lemma 2.37 there is a graph G with degree sequence d so that:

(1) $\deg(v_i) = d_i$ for $i = 1, \ldots, n$ and
(2) v_1 is adjacent to vertices $v_2, \ldots, v_{d_1+1}$.

If we remove vertex v_1 and all edges containing v_1 from this graph G to obtain G' then in G' for all $i = 2, \ldots d_1 + 1$ the degree of v_i is $d_i - 1$ while for $j = d_1 + 2, \ldots, n$ the degree of v_j is d_j because v_1 is not adjacent to $v_{d_1+2}, \ldots, v_n$ by choice of G. Thus G' has degree sequence $(d_2 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ and thus it is graphic.

($\Leftarrow$) Now suppose that $(d_2 - 1, \ldots, d_{d_1+1} - 1, d_{d_1+2}, \ldots, d_n)$ is graphic. Then there is some graph G that has this as its degree sequence. We can construct a new graph G' from G by adding a vertex v_1 to G and creating an edge from v_1 to each vertex v_2 through v_{d_1+1}. It is clear that the degree of v_1 is d_1, while the degrees of all other vertices v_i must be d_i and thus $\mathbf{d} = (d_1, \dots, d_n)$ is graphic because it is the degree sequence of G'. This completes the proof. □

Remark 2.39. Naturally one might have to rearrange the ordering of the degree sequence $(d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n)$ to ensure it is in descending order.

Example 2.40. Consider the degree sequence $\mathbf{d} = (5, 5, 4, 3, 2, 1)$. One might ask, is this degree sequence graphic. Note that $5 + 5 + 4 + 3 + 2 + 1 = 20$ so at least the necessary condition, that the degree sequence sum to an even number is satisfied. In this $\mathbf{d}$ we have $d_1 = 5$, $d_2 = 5$, $d_3 = 4$, $d_4 = 3$, $d_5 = 2$ and $d_6 = 1$.

Applying the Havel-Hakimi Theorem, we know that this degree sequence is graphic if and only if: $\mathbf{d}' = (4, 3, 2, 1, 0)$ is graphic. Note, that this is $(d_2 - 1, d_3 - 1, d_4 - 1, d_5 - 1, d_6 - 1)$ since $d_1 + 1 = 5 + 1 = 6$. Now, if $\mathbf{d}'$ where graphic, then we would have a graph with 5 vertices one of which has degree 4 and another that has degree 0 and no to vertices have the same degree. Applying either Theorem 2.32 (or its proof), we see this is not possible. Thus $\mathbf{d}'$ is not graphic and so $\mathbf{d}$ is not graphic.

Exercise 7. Develop a (recursive) algorithm based on Theorem 2.38 to determine whether a sequence is graphic. [Hint: See Page 10 of [GY05].]

Theorem 2.41 (Erdös-Gallai Theorem). *A degree sequence* $\mathbf{d} = (d_1, \dots, d_n)$ *is graphic if and only if its sum is even and for all* $1 \leq k \leq n - 1$*:*

$$\sum_{i=0}^{k} d_i \leq k(k+1) + \sum_{i=k+1}^{n-1} \min\{k+1, d_i\} \tag{2.6}$$

Exercise 8 (**Independent Project**). There are several proofs of Theorem 2.41, some short. Investigate them and reconstruct an annotated proof of the result. In addition investigate Berg's approach using flows [Ber73].

Remark 2.42. There has been a lot of interest recently in degree sequences of graphs, particularly as a result of the work in Network Science on so-called *scale-free* networks. This has led to a great deal of investigation into properties of graphs with specific kinds of degree sequences. For the brave, it is worth looking at [MR95], [ACL01], [BR03] and [Lu01] for interesting mathematical results in this case. To find out why all this investigation started, see [BAJB00].

3.1. Types of Graphs from Degree Sequences.

Definition 2.43 (Complete Graph). Let $G = (V, E)$ be a graph with $|V| = n$ with $n \geq 1$. If the degree sequence of G is $(n - 1, n - 1, \dots, n - 1)$ then G is called a complete graph on n vertices and is denoted K_n. In a complete graph on n vertices each vertex is connected to every other vertex by an edge.

Lemma 2.44. *Let* $K_n = (V, E)$ *be the complete graph on* n *vertices. Then:*

$$|E| = \frac{n(n-1)}{2}$$

□

Corollary 2.45. *Let $G = (V, E)$ be a graph and let $|V| = n$. Then:*

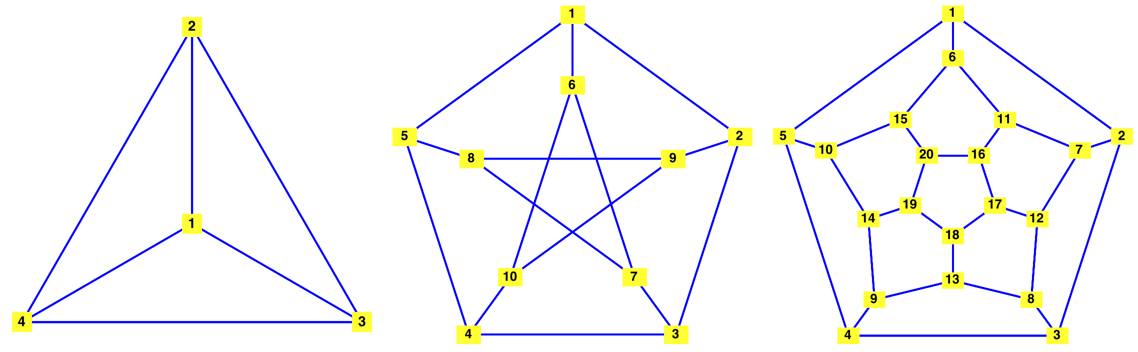
$$0 \leq |E| \leq \binom{n}{2}$$

$\square$

Exercise 9. Prove Lemma 2.44 and Corollary 2.45. [Hint: Use Equation 2.4.]

Definition 2.46 (Regular Graph). Let $G = (V, E)$ be a graph with $|V| = n$. If the degree sequence of G is $(k, k, \ldots, k)$ with $k \leq n - 1$ then G is called a k-regular graph on n vertices.

Example 2.47. We illustrate one complete graph and two (non-complete) regular graphs in Figure 2.8. Obviously every complete graph is a regular graph. Every Platonic solid is



(a) K_4 (b) Petersen Graph (c) Dodecahedron

Figure 2.8. The complete graph, the "Petersen Graph" and the Dodecahedron. All Platonic solids are three-dimensional representations of regular graphs, but not all regular graphs are Platonic solids. These figures were generated with Maple.

also a regular graph, but not every regular graph is a Platonic solid. In Figure 2.8(c) we show a flattened dodecahedron, one of the five platonic solids from classical geometry. The Peteron Graph (Figure 2.8(b)) is a 3-regular graph that is used in many graph theoretic examples.

3.2. Digraphs.

Definition 2.48 (In-Degree, Out-Degree). Let $G = (V, E)$ be a digraph. The *in-degree* of a vertex v in G is the total number of edges in E with *destination* v. The *out-degree* of v is the total number of edges in E with *source* v. We will denote the in-degree of v by $\deg_{in}(v)$ and the out-degree by $\deg_{out}(v)$.

Theorem 2.49. *Let $G = (V, E)$ be a digraph. Then the following holds:*

$$|E| = \sum_{v \in V} \deg_{in}(v) = \sum_{v \in V} \deg_{out}(v) \tag{2.7}$$

Exercise 10. Prove Theorem 2.49.

4. Subgraphs

Definition 2.50 (Subgraph). Let $G = (V, E)$. A graph $H = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. The subgraph H is *proper* if $V' \subsetneq V$ or $E' \subsetneq E$.

Example 2.51. We illustrate the notion of a sub-graph in Figure 2.9. Here we illustrate a sub-graph of the Petersen Graph. The sub-graph contains vertices 6, 7, 8, 9 and 10 and the edges connecting them.

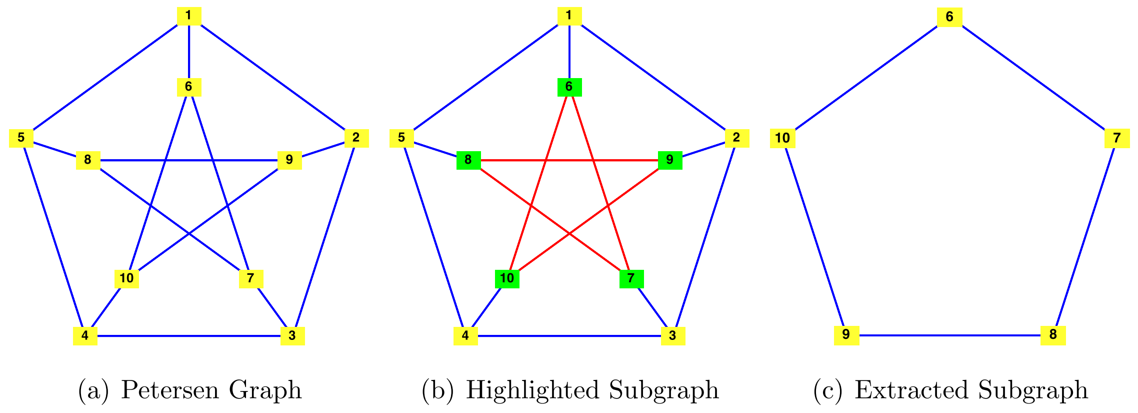


Figure 2.9. The Petersen Graph is shown (a) with a sub-graph highlighted (b) and that sub-graph displayed on its own (c). A sub-graph of a graph is another graph whose vertices and edges are sub-collections of those of the original graph.

Definition 2.52 (Spanning Subgraph). Let $G = (V, E)$ be a graph and $H = (V', E')$ be a subgraph of G. The subgraph H is a *spanning subgraph* of G if $V' = V$.

Definition 2.53 (Edge Induced Subgraph). Let $G = (V, E)$ be a graph. If $E' \subseteq E$. The subgraph of G induced by E' is the graph $H = (V', E')$ where $v \in V'$ if and only if v appears in an edge in E.

Definition 2.54 (Vertex Induced Subgraph). Let $G = (V, E)$ be a graph. If $V' \subseteq E$. The subgraph of G induced by V' is the graph $H = (V', E')$ where $\{v_1, v_2\} \in E'$ if and only if v_1 and v_2 are both in V'.

Remark 2.55. For directed graphs, all sub-graph definitions are modified in the obvious way. Edges become directed as one would expect.

Example 2.56. Using the Petersen Graph we illustrate a subgraph induced by a vertex subset and a spanning subgraph. In Figure 2.10(a) we illustrate the subgraph induced by the vertex subset $V' = \{6, 7, 8, 9, 10\}$ (shown in red). In Figure 2.10(b) we have a spanning subgraph induced by the edge subset:

$$E' = \{\{1,6\},\{2,9\},\{3,7\},\{4,10\},\{5,8\},\{6,7\},\{6,10\},\{7,8\},\{8,9\},\{9,10\}\}$$

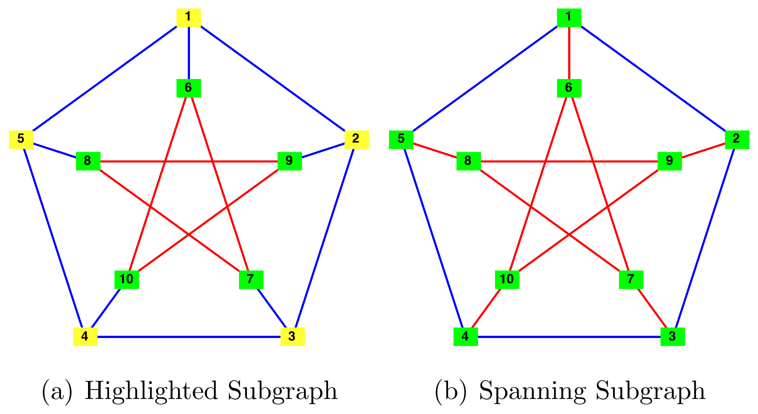


Figure 2.10. The subgraph (a) is induced by the vertex subset $V' = \{6, 7, 8, 9, 10\}$. The subgraph shown in (b) is a spanning sub-graph and is induced by edge subset $E' = \{\{1, 6\}, \{2, 9\}, \{3, 7\}, \{4, 10\}, \{5, 8\}, \{6, 7\}, \{6, 10\}, \{7, 8\}, \{8, 9\}, \{9, 10\}\}$.

5. Graph Complement, Cliques and Independent Sets

Definition 2.57 (Clique). Let $G = (V, E)$ be a graph. A *clique* is a set $S \subseteq V$ of vertices so that:

(1) The subgraph induced by S is a complete graph (or in general graphs, every pair of vertices in S is connected by at least one edge in E) *and*
(2) If $S' \supset S$, there is at least one pair of vertices in S' that are not connected by an edge in E.

Definition 2.58 (Independent Set). Let $G = (V, E)$ be a graph. A *independent set* of G is a set $I \subseteq V$ so that no pair of vertices in I is joined by an edge in E.

Example 2.59. The easiest way to think of cliques is as subgraphs that are K_n but so that no larger set of vertices induces a larger complete graph. Independent sets are the opposite of cliques. The graph illustrated in Figure 2.11(a) has 3 cliques. An independent set is illustrated in Figure 2.11(b).

Definition 2.60 (Clique Number). Let $G = (V, E)$ be a graph. The *clique number* of G, written $\omega(G)$ is the size (number of vertices) of the largest clique in G.

Definition 2.61 (Independence Number). The *independence number* of a graph $G = (V, E)$, written $\alpha(G)$, is the size of the largest independent set of G.

Exercise 11. Find the clique and independence numbers of the graph shown in Figure 2.11(a)/(b).

Definition 2.62 (Graph Complement). Let $G = (V, E)$ be a graph. The *graph complement* of G is a graph $H = (V, E')$ so that:

$$e = \{v_1, v_2\} \in E' \iff \{v_1, v_2\} \notin E$$

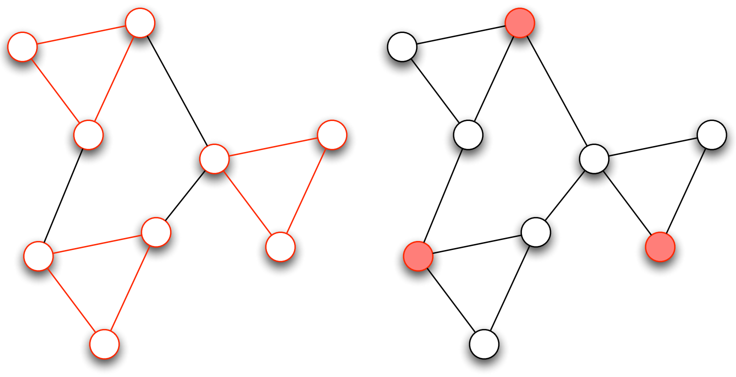


Figure 2.11. A clique is a set of vertices in a graph that induce a complete graph as a subgraph and so that no larger set of vertices has this property. The graph in this figure has 3 cliques.

EXAMPLE 2.63. In Figure 2.12, the graph from Figure 2.11 is illustrated (in a different spatial configuration) with its cliques. The complement of the graph is also illustrated. Notice that in the complement, every clique is now an independent set.

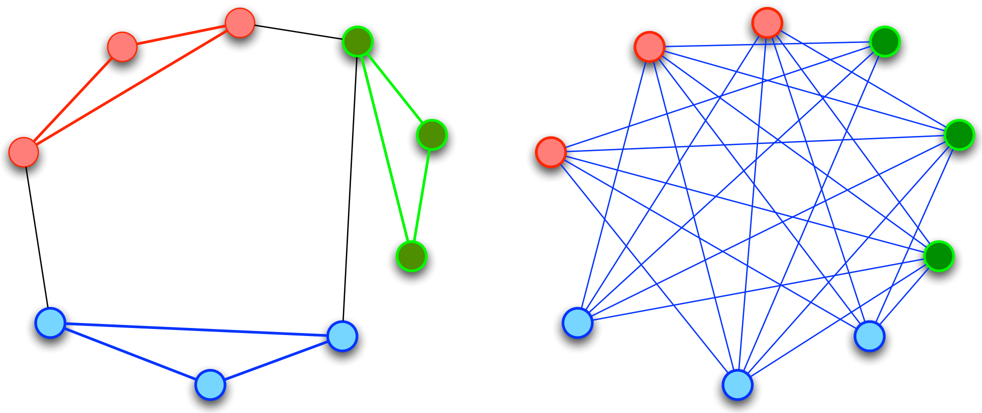


Figure 2.12. A graph and its complement with cliques in one illustrated and independent sets in the other illustrated.

DEFINITION 2.64 (Relative Complement). If $G = (V, E)$ is a graph and $H = (V, E')$ is a spanning sub-graph, then the *relative complement* of H in G is the graph $H' = (V, E'')$ with:

$$e = \{v_1, v_2\} \in E'' \iff \{v_1, v_2\} \in E \text{ and } \{v_1, v_2\} \not\in E'$$

THEOREM 2.65. *Let $G = (V, E)$ be a graph and let $H = (V, E')$ be its complement. A set S is a clique in G if and only if S is a maximal independent set in H.*

EXERCISE 12. Prove Theorem 2.65. [Hint: Use the definition of graph complement and the fact that if an edge is present in a graph G is must be absent in its complement.]

Definition 2.66 (Vertex Cover). Let $G = (V, E)$ be a graph. A vertex cover is a set of vertices $S \subseteq V$ so that for all $e \in E$ at least one element of e is in S; i.e., every edge in E is adjacent to at least one vertex in S.

Example 2.67. A covering is illustrated in Figure 2.13

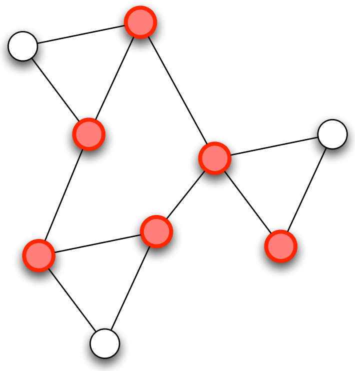


Figure 2.13. A covering is a set of vertices so that ever edge has at least one endpoint inside the covering set.

Exercise 13. Illustrate by exhaustion that removing any vertex from the proposed covering in Figure 2.13 destroys the covering property.

Theorem 2.68. *A set I is an independent set in a graph $G = (V, E)$ if and only if the set $V \setminus I$ is a covering in G.*

Proof. ($\Rightarrow$) Suppose that I is an independent set and choose $e = \{v, v'\} \in E$. If $v \in I$, then clearly $v' \in V \setminus I$. The same is true of v'. It is possible that neither v nor v' is in I, but this does not affect that fact that $V \setminus I$ must be a cover since for every edge $e \in E$ at least one element is in $V \setminus I$.

($\Leftarrow$) Now suppose that $V \setminus I$ is a vertex covering. Choose any two vertices v and v' in I. The fact that $V \setminus I$ is a vertex covering implies that $\{v, v'\}$ cannot be an edge in E because it does not contain at least one element from $V \setminus I$, contradicting our assumption on $V \setminus I$. Thus, I is an independent set since no two vertices in I are connected by an edge in E. This completes the proof. □

Remark 2.69. Theorem 2.68 shows that the problem of identifying a largest independent set is identical to the problem of identifying a minimum (size) vertex covering. As it turns out, both these problems are equivalent to yet a third problem, which we will discuss later called the matching problem. Coverings (and matchings) are useful, but to see one example of their utility imagine a network of outposts is to be established in an area (like a combat theatre). We want to deliver a certain type of supplies (antibiotics for example) to the outposts in such a way so that no outpost is anymore than one link (edge) away from an outpost where the supply is available. The resulting problem is a vertex covering problem. In attempting to find the *minimal vertex covering* asks the question what is the minimum number of outposts that must be given antibiotics?

CHAPTER 3

More Definitions and Theorems

1. Paths, Walks, and Cycles

Definition 3.1 (Walk). Let $G = (V, E)$ be a graph. A walk $w = (v_1, e_1, v_2, e_2, \dots, v_n, e_n, v_{n+1})$ in G is an alternating sequence of vertices and edges in V and E respectively so that for all $i = 1, \dots, n$: $\{v_i, v_{i+1}\} = e_i$. A walk is called *closed* if $v_1 = v_{n+1}$ and *open* otherwise. A walk consisting of only one vertex is called *trivial*.

Definition 3.2 (Sub-Walk). Let $G = (V, E)$ be a graph. If w is a walk in G then a *sub-walk* of w is any walk w' that is also a sub-sequence of w.

Remark 3.3. Let $G = (V, E)$ to each walk $w = (v_1, e_1, v_2, e_2, \dots, v_n, e_n, v_{n+1})$ we can associated a subgraph $H = (V', E')$ with:

(1) $V' = \{v_1, \dots, v_{n+1}\}$
(2) $E' = \{e_1, \dots, e_n\}$

We will call this the sub-graph induced by the walk w.

Definition 3.4 (Trail/Tour). Let $G = (V, E)$ be a graph. A trail in G is a walk in which no edge is repeated. A *tour* is a closed trail. An Eulerian trail is a trail that contains exactly one copy of each edge in E and an *Eulerian tour* is a closed trail (tour) that contains exactly one copy of each edge.

Definition 3.5 (Path). Let $G = (V, E)$ be a graph. A *path* in G is a *non-trivial* walk with no vertex and no edge repeated. A *Hamiltonian path* is a path that contains exactly one copy of each vertex in V[1].

Definition 3.6 (Length). The *length* of a walk w is the number of edges contained in it.

Definition 3.7 (Cycle). A closed walk of length at least 3 and with no repeated edges and in which the *only* repeated vertices are the first and the last is called a cycle. A *Hamiltonian cycle* is a cycle in a graph containing every vertex.

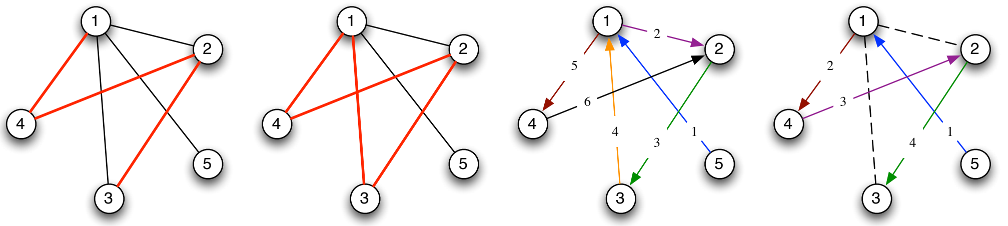
Definition 3.8 (Hamiltonian / Eulerian Graph). A graph $G = (V, E)$ is said to be *Hamiltonian* if it contains a Hamiltonian cycle and Eulerian if it contains an Eulerian tour.

Example 3.9. We illustrate a walk, cycle, Eulerian tour and a Hamiltonian path in Figure 3.1.

A walk is illustrated in Figure 3.1(a). Formally, this walk can be written as:

$$w = (1, \{1, 4\}, 4, \{4, 2\}, 2, \{2, 3\}, 3)$$

[1]Thanks to S. Shekhar for pointing out a missing part of this definition.



(a) Walk (b) Cycle (c) Eulerian Trail (d) Hamiltonian Path

Figure 3.1. A walk (a), cycle (b), Eulerian trail (c) and Hamiltonian path (d) are illustrated.

The cycle shown in Figure 3.1(b) can be formally written as:

$$c = (1, \{1, 4\}, 4, \{4, 2\}, 2, \{2, 3\}, 3, \{3, 1\}, 1)$$

Notice that the cycle begins and ends with the same vertex (that's what makes it a cycle). Also, w is a sub-walk of c. Note further we could easily have represented the walk as:

$$w = (3, \{3, 2\}, 2, \{2, 4\}, 4, \{4, 1\}, 1)$$

We could have shifted the ordering of the cycle in anyway (for example beginning at vertex 2). Thus we see that in an undirected graph, a cycle or walk representation may not be unique.

In Figure 3.1(c) we illustrate an Eulerian Trail. This walk contains every edge in the graph with no repeats. We note that Vertex 1 is repeated in the trail, meaning this is not a *path*. We contrast this with Figure 3.1(d) which shows a Hamiltonian path. Here each vertex occurs exactly once in the illustrated path, but not all the edges are included. In this graph, it is impossible to have either a Hamiltonian Cycle or an Eulerian Tour.

Exercise 14. Prove it is not possible for a Hamiltonian Cycle or Eulerian Tour to exist in the graph in Figure 3.1(a); i.e., prove that the graph is neither Hamiltonian nor Eulerian.

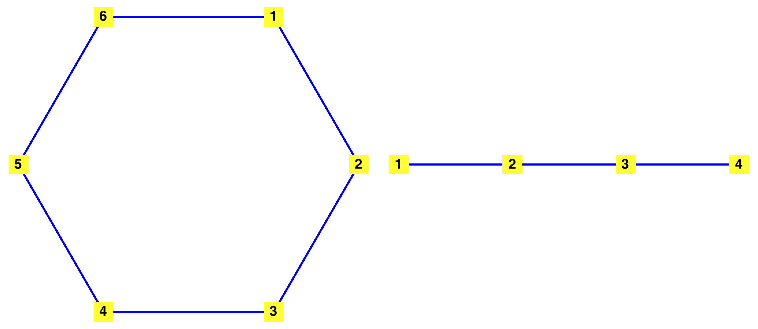
Remark 3.10. If w is a path in a graph $G = (V, E)$ then the subgraph induced by w is simply the graph composed of the vertices and edges in w.

Proposition 3.11. *Let G be a graph and let w be an Eulerian trail (or tour) in G. Then the sub-graph of G induced by w is G itself when G has no isolated vertices.*

Exercise 15. Prove Proposition 3.11.

Definition 3.12 (Path / Cycle Graph). Suppose that $G = (V, E)$ is a graph with $|V| = n$. If w is a Hamiltonian path in G and H is the subgraph induced by w and $H = G$, then G is called a *n-path* or a *Path Graph* on n vertices denoted P_n. If w is a Hamiltonian cycle in G and H is the subgraph induced by w and $H = G$, then G is called a *n-cycle* or a *Cycle Graph* on n vertices denoted C_n.

Example 3.13. We illustrate a cycle graph with 6 vertices (6-cycle or C_6) and a path graph with 4 vertices (4-path or P_4) in Figure 3.2.



(a) 6-cycle (b) 4-path

Figure 3.2. We illustrate the 6-cycle and 4-path.

Remark 3.14. For the most part, the terminology on paths, cycles, tours etc. is standardized. However, not every author adheres to these same terms. It is always wise to identify exactly what words an author is using for walks, paths cycles etc.

Remark 3.15. Walks, cycles, paths and tours can all be extended to the case of digraphs. In this case, the walk, path, cycle or tour *must* respect the edge directionality. Thus, if $w = (\dots, v_i, e_i, v_{i+1}, \dots)$ is a directed walk, then $e_i = (v_i, v_{i+1})$ as an ordered pair.

Exercise 16. Formally define directed walks, directed cycles, directed paths and directed tours for directed graphs. [Hint: Begin with Definition 3.1 and make appropriate changes. Then do this for cycles, tours etc.]

2. More Graph Properties: Diameter, Radius, Circumference, Girth

Definition 3.16 (Distance). Let $G = (V, E)$. The *distance* between v_1 and v_2 in V is the length of the *shortest* walk beginning at v_1 and ending at v_2 if such a walk exists. Otherwise, it is $+\infty$. We will write $d_G(v_1, v_2)$ for the distance from v_1 to v_2 in G.

Definition 3.17 (Directed Distance). Let $G = (V, E)$ be a digraph. The *(directed) distance* between v_1 to v_2 in V is the length of the *shortest* directed walk beginning at v_1 and ending at v_2 if such a walk exists. Otherwise, it is $+\infty$

Definition 3.18 (Diameter). Let $G = (V, E)$ be a graph. The *diameter* of G diam(G) is the length of the largest distance in G. That is:

$$\operatorname{diam}(G) = \max_{v_1, v_2 \in V} d_G(v_1, v_2) \tag{3.1}$$

Definition 3.19 (Eccentricity). Let $G = (V, E)$ and let $v_1 \in V$. The *eccentricity* of v_1 is the largest distance from v_1 to any other vertex v_2 in V. That is:

$$\operatorname{ecc}(v_1) = \max_{v_2 \in V} d_G(v_1, v_2) \tag{3.2}$$

Exercise 17. Show that the diameter of a graph is in fact the maximum eccentricity of any vertex in the graph.

Definition 3.20 (Radius). Let $G = (V, E)$. The *radius* of G is minimum eccentricy of any vertex in V. That is:

$$\operatorname{rad}(G) = \min_{v_1 \in V} \operatorname{ecc}(v_1) = \min_{v_1 \in V} \max_{v_2 \in V} d_G(v_1, v_2) \tag{3.3}$$

Definition 3.21 (Girth). Let $G = (V, E)$ be a graph. If there is a cycle in G (that is G has a cycle-graph as a subgraph), then the *girth* of G is the length of the *shortest* cycle. When G contains no cycle, the girth is defined as 0.

Definition 3.22 (Circumference). Let $G = (V, E)$ be a graph. If there is a cycle in G (that is G has a cycle-graph as a subgraph), then the *circumference* of G is the length of the *longest* cycle. When G contains no cycle, the circumference is defined as $+\infty$.

Example 3.23. The eccentricities of the vertices of the graph shown in Figure 3.3 are:

(1) Vertex 1: 1
(2) Vertex 2: 2
(3) Vertex 3: 2
(4) Vertex 4: 2
(5) Vertex 5: 2

This means that the diameter of the graph is 2 and the radius is 1. We have already seen that there is a 4-cycle subgraph in the graph (see Figure 3.1(b)). This is the largest cycle in the graph, so the circumference of the graph is 4. There are several 3-cycles in the graph (an example being the cycle $(1, \{1, 2\}, 2, \{2, 4\}, 4, \{4, 1\}, 1)$). The smallest possible cycle is a 3-cycle. Thus the girth of the graph is 3.

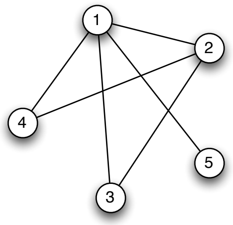


Figure 3.3. The diameter of this graph is 2, the radius is 1. It's girth is 3 and its circumference is 4.

Exercise 18. Compute the diameter, radius, girth and circumference of the Petersen Graph.

3. More on Trails and Cycles

Remark 3.24. Suppose that

$$w = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1})$$

If for some $m \in \{1, \dots, n\}$ and for some $k \in \mathbb{Z}$ we have $v_m = v_{m+k}$. Then

$$w' = (v_m, e_m, \dots, e_{m+k-1}, v_{m+k})$$

is a *closed sub-walk* of w. The walk w' can be deleted from the walk w to obtain a new walk:

$$w'' = (v_1, e_1, v_2, \dots, v_{m+k}, e_{m+k}, v_{m+k+1}, \dots, v_n, e_n, v_{n+1})$$

that is shorter than the original walk. This is illustrated in Figure 3.4. [GY05] calls this a walk reduction, though this notation is not standard.

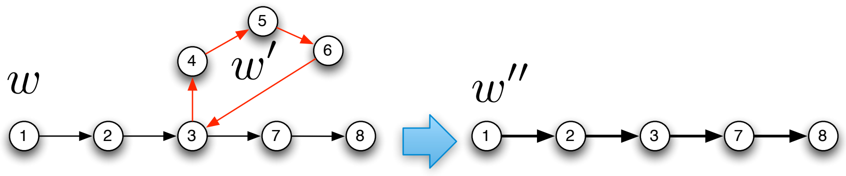


Figure 3.4. We can create a new walk from an existing walk by removing closed sub-walks from the walk.

Lemma 3.25. *Let $G = (V, E)$ be a graph and suppose that t is a non-trivial tour (closed trail) in G. Then t contains a cycle.*

Proof. The fact that t is closed implies that it contains at least one pair of repeated vertices. Therefore a closed sub-walk of t must exist since t is itself has these repeated vertices. Let c be a minimal (length) closed sub-walk of t. We will show that c must be a cycle. By way of contradiction, suppose that c is not a cycle. Then since it is closed it must contain a repeated vertex (that is not its first vertex). If we applied our observation from Remark 3.24 we could produce a smaller closed walk c', contradicting our assumption that c was minimal. Thus c must have been a cycle. This completes the proof. $\square$

Theorem 3.26. *Let $G = (V, E)$ be a graph and suppose that t is a non-trivial tour (closed trail). Then t is composed of edge disjoint cycles.*

Proof. We will proceed by induction. In the base case, assume that t is a one edge closed tour, then G is a non-simple graph that contains a self-loop and this is a single edge in t and thus t is a (non-simple) cycle[2]. Now suppose the theorem holds for all closed trails of length N or less. We will show the result holds for a tour of length $N + 1$. Applying Lemma 3.25, we know there is at least one cycle c in t. If we reduce tour t by c to obtain t', then t is still a tour and has length at most N. We can now apply the induction hypothesis to see that this new tour t' is composed of disjoint cycles. When taken with c, it is clear that t is now composed of disjoint cycles. The theorem is illustrated in Figure 3.5. This completes the proof. $\square$

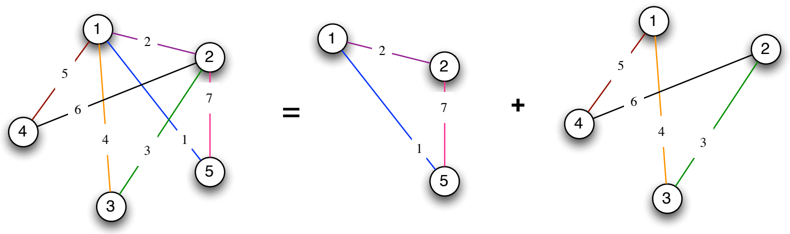
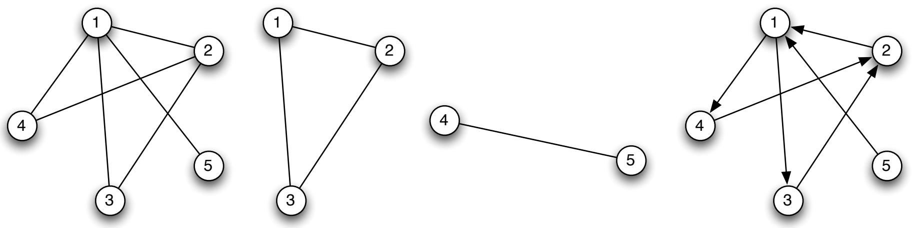


Figure 3.5. We show how to decompose an (Eulerian) tour into an edge disjoint set of cycles, thus illustrating Theorem 3.26.

[2]If we assume that G is simple, then the base case begins with t having length 3. In this case it is a 3-cycle.



(a) Connected (b) Disconnected (c) Not Strongly Connected

Figure 3.6. A connected graph (a), a disconnected graph (b) and a connected digraph that is not strongly connected (c).

4. Graph Components

Definition 3.27 (Reachability). Let $G = (V, E)$ and let v_1 and v_2 be two vertices in V. Then v_2 is *reachable* from v_1 if there is a walk w beginning at v_1 and ending at v_2 (alternatively, the distance from v_1 to v_2 is not $+\infty$). If G is a digraph, we assume that the walk is directed.

Definition 3.28 (Connectedness). A graph G is connected if for every pair of vertices v_1 and v_2 in V, v_2 is reachable from v_1. If G is a digraph, then G is connected if its *underlying graph* is connected. A graph that is not connected is called *disconnected.*

Definition 3.29 (Strong Connectedness). A digraph G is strongly connected if for every pair of vertices v_1 and v_2 in V, v_2 is reachable (by a directed walk) from v_1.

Remark 3.30. In Definition 3.29 we are really requiring, for any pair of vertices v_1 and v_2, that v_1 be reachable from v_2 *and* v_2 be reachable from v_1 by directed walks. If this is not possible, then a directed graph could be connected but not *strongly connected.*

Example 3.31. Figure 3.6 we illustrate a connected graph, a disconnected graph and a connected digraph that is not strongly connected.

Definition 3.32 (Component). Let $G = (V, E)$ be a graph. A subgraph H of G is a *component* of G if:

(1) H is connected and
(2) If K is a subgraph of G and H is a proper subgraph of K, then K is not connected. The number of components of a graph G is written $c(G)$.

Remark 3.33. A component H of a graph G is called a *maximal connected subgraph.* Here *maximal* is taken with respect to the sub-graph ordering. That is, H is less than K in the sub-graph ordering if H is a proper subgraph of K.

Example 3.34. Figure 3.6(b) contains two components: the 3-cycle and 2-path.

Proposition 3.35. *A connected graph G has exactly one component.*

Exercise 19. Prove Proposition 3.35.

Definition 3.36 (Edge Deletion Graph). Let $G = (V, E)$ and let $E' \subseteq E$. Then the graph G' resulting from deleting the edges in E' from G is the sub-graph induced by the edge set $E \setminus E'$. We write this as $G' = G - E'$.

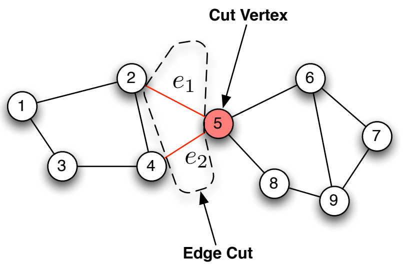
Definition 3.37 (Vertex Deletion Graph). Let $G = (V, E)$ and let $V' \subseteq V$. Then the graph G' resulting from deleting the edges in V' from G is the sub-graph induced by the vertex set $V \setminus V'$. We write this as $G' = G - V'$.

Definition 3.38 (Vertex Cut and Cut Vertex). Let $G = (V, E)$ be a graph. A set $V' \subset V$ is a *vertex cut* if the graph G' resulting from deleting vertices V' from G has *more components* than graph G. If $V' = \{v\}$ is a vertex cut, then v is called a *cut vertex*.

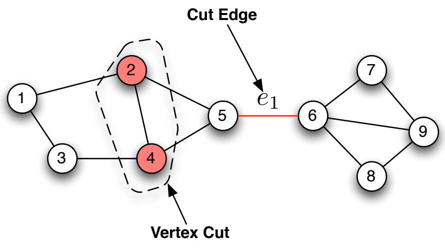
Definition 3.39 (Edge Cut and Cut-Edge). Let $G = (V, E)$ be a graph. A set $E' \subset E$ is a *edge cut* if the graph G' resulting from deleting edges E' from G has *more components* than graph G. If $E' = \{e\}$ is an edge cut, then e is called a *cut-edge*.

Definition 3.40 (Minimal Edge Cut). Let $G = (V, E)$. An edge cut E' of G is *minimal* if when we remove any edge from E' to form E'' the new set E'' is no longer an edge cut.

Example 3.41. In figure 3.7 we illustrate a vertex cut and a cut vertex (a singleton vertex cut) and an edge cut and a cut edge (a singleton edge cut).Note that the edge-cut in Figure 3.7(a) is minimal and cut-edges are always minimal. A cut-edge is sometimes called a *bridge* because it connects two distinct components in a graph. Bridges (and small edge cuts) are a very important part of social network analysis [**KY08**, **CSW05**] because they represent connections between different communities. To see this, suppose that (for



(a) Edge Cut and Cut Vertex



(b) Vertex Cut and Cut Edge

Figure 3.7. We illustrate a vertex cut and a cut vertex (a singleton vertex cut) and an edge cut and a cut edge (a singleton edge cut). Cuts are sets of vertices or edges whose removal from a graph creates a new graph with more components than the original graph.

example) Figure 3.7(b) represents the communications connections between individuals in two terrorist cells. The fact that Member 5 and 6 communicate *and* that these are the only two individuals who communicate between these two cells could be important for finding a way to capture or disrupt this small terrorist network.

Theorem 3.42. *Let $G = (V, E)$ be a connected graph and let $e \in E$. Then $G' = G - \{e\}$ is connected if and only if e lies on a cycle in G.*

PROOF. ($\Leftarrow$) Recall a graph G is connected if and only if for every pair of vertices v_1 and v_{n+1} there is a walk w from v_1 to v_{n+1} with:

$$w = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1})$$

Let $G' = G - \{e\}$. Suppose that e lies on a cycle c in G and choose two vertices v_1 and v_{n+1} in G. If e is not on any walk w connecting v_1 to v_{n+1} in G then the removal of e does not affect the reachability of v_1 and v_{n+1} in G'. Therefore assume that e is in the walk w. The fact that e is in a cycle of G implies we have vertices $u_1, \dots, u_m$ and edges $f_1, \dots, f_m$ so that:

$$c = (u_1, f_1, \dots, u_m, f_m, u_1)$$

is a cycle and e is among the $f_1, \dots, f_m$. Without loss of generality, assume that $e = f_m$ and that $e = \{u_m, u_1\}$. (Otherwise, we can re-order the cyle to make this true.) Then in G' we will have the path:

$$c' = (u_1, f_1, \dots, u_m)$$

The fact that e is in the walk w implies there are vertices v_i and v_{i+1} so that $e = \{v_i, v_{i+1}\}$ (with $v_i = u_1$ and $v_{i+1} = u_m$). In deleting e from G we remove the sub-walk (v_i, e, v_{i+1}) from w. But we can create a new walk with structure:

$$w' = (v_1, e_1, \dots, v_i, f_1, u_2, \dots, u_{m-1}, f_{m-1}, u_m, \dots, e_n, v_{n+1})$$

This is illustrated in Figure 3.8.

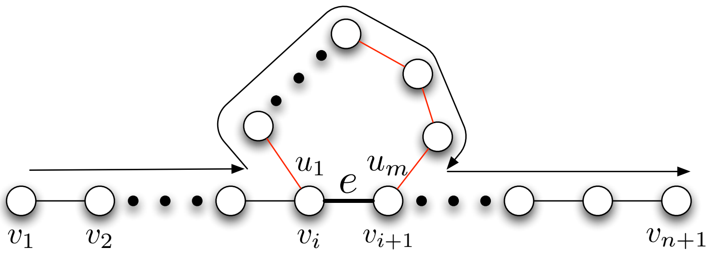


Figure 3.8. If e lies on a cycle, then we can repair path w by *going the long way around the cycle* to reach v_{n+1} from v_1.

($\Rightarrow$) Suppose $G' = G - \{e\}$ is connected. Now let $e = \{v_1, v_{n+1}\}$. Since G' is connected, there is a walk from v_1 to v_{n+1}. Applying Remark 3.24, we can reduce this walk to a path p with:

$$p = (v_1, e_1, \dots, v_n, e_n, v_{n+1})$$

Since p is a path, there are no repeated vertices in p. We can construct a cycle c containing e in G as:

$$p = (v_1, e_1, \dots, v_n, e_n, v_{n+1}, e, v_1)$$

since $e = \{v_1, v_{n+1}\} = \{v_{n+1}, v_1\}$. Thus, e lies on a cycle in G. This completes the proof. $\square$

COROLLARY 3.43. *Let $G = (V, E)$ be a connected graph and let $e \in E$. The edge e is a cut edge if and only if e does not lie on a cycle in G.*

EXERCISE 20. Prove Corollary 3.43.

Remark 3.44. The next result is taken from Extremal Graph Theory, the study of extremes or bounds in properties of graphs. There are a number of results in Extremal Graph Theory that are of interest. See [Bol04] for a complete introduction.

Theorem 3.45. *If $G = (V, E)$ is a graph with n vertices and k components, then:*

$$|E| \leq \frac{(n-k+1)(n-k)}{2}$$

Proof. Assume that each component of G has n_i vertices in it with $\sum_{i=1}^{k} n_i = n$. Applying Lemma 2.44 we know that Component i has at most $n_i(n_i-1)/2$ edges; that is, each component is a complete graph on n_i vertices. This is the largest number of edges that can occur under these assumptions.

Consider the case where $k-1$ of the components has exactly 1 vertex and the remaining component has $n-(k-1)$ vertices. Then the total number of edges in this case is:

$$\frac{(n-(k-1))(n-(k-1)-1)}{2} = \frac{(n-k+1)(n-k)}{2}$$

edges. It now suffices to show that this case has the greatest number of vertices of all cases where the k components are each complete graphs.

Consider the case when component i is K_r and component j is K_s with $r, s \geq 2$ and suppose $r \geq s$. Then the total number of edges in *these two components* is:

$$\frac{r(r-1)+s(s-1)}{2} = \frac{r^2+s^2-r-s}{2}$$

Now, suppose we move one vertex in component j to component i. Then component i is now K_{r+1} and component j is now K_{s-1}. Applying Lemma 2.44, the number of edges in this case is:

$$\frac{(r+1)(r)+(s-1)(s-2)}{2} = \frac{r^2+r+s^2-3s+2}{2}$$

Observe that since $r \geq s$, substituting s for r we have:

$$r^2+r+s^2-3s+2 \geq r^2+s^2-2s+2$$

By a similar argument:

$$r^2+s^2-2s \geq r^2+s^2-r-s$$

Thus we conclude that:

$$\frac{r^2+r+s^2-3s+2}{2} \geq \frac{r^2+s^2-2s+2}{2} \geq \frac{r^2+s^2-2s}{2} \geq \frac{r^2+s^2-r-s}{2}$$

Repeating this argument over and over shows that in a k component graph with n vertices, the largest number of edges must occur in the case when there is one component with $n-(k-1)$ vertex and $k-1$ components with exactly 1 vertex. This completes the proof. □

Corollary 3.46. *Any graph with n vertices and more than $(n-1)(n-2)/2$ edges is connected.*

Exercise 21. Prove Corollary 3.46.

5. Bipartite Graphs

DEFINITION 3.47. A graph $G = (V, E)$ is *bipartite* if $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$ and if $e = E$, then $e = \{v_1, v_2\}$ with $v_1 \in V_1$ and $v_2 \in V_2$. This definition is valid for non-simple graphs as well.

REMARK 3.48. In a bipartite graph, we can think of the vertices as belonging to one of two classes (either V_1 or V_2) and edges only exist between elements of the two classes, not between elements in the same class. We can also define n-partite graphs in which the vertices are in any of n classes and there are only edges between classes, not within classes.

EXAMPLE 3.49. Figure 3.9 shows a bipartite graph in which $V_1 = \{1, 2, 3\}$ and $V_2 = \{4, 5, 6, 7\}$. Notice that there are only edges connecting vertices in V_1 and vertices in V_2. There are not edges connecting elements in V_1 to other elements in V_1 or elements in V_2 to other elements in V_2.

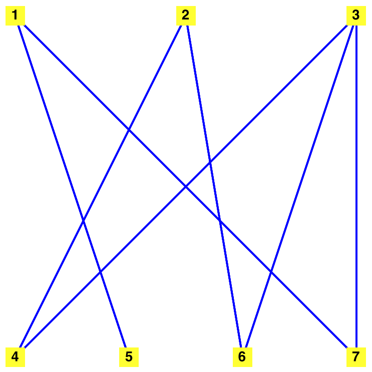


Figure 3.9. A bipartite graph has two classes of vertices and edges in the graph only exists between elements of different classes.

DEFINITION 3.50 (Path Concatenation). Let $p_1 = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1})$ and let $p_2 = (v_{n+1}, e_{n+1}, v_{n+2}, \dots, v_{n+m}, e_{n+m}, v_{n+m+1})$. Then the *concatenation* of path p_1 with path p_2 is the path:

$$p = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1}, e_{n+1}, v_{n+2}, \dots, v_{n+m}, e_{n+m}, v_{n+m+1})$$

REMARK 3.51. Path concatenation is illustrated in the proof of Theorem 3.52.

THEOREM 3.52. *A graph $G = (V, E)$ is bipartite if and only if every cycle in G has even length.*

PROOF. ($\Rightarrow$) Suppose G is bipartite. Every cycle begins and ends at the same vertex and therefore in the same partition, either V_1 or V_2. Starting at a vertex $v_1 \in V_1$ we must take a walk of length 2 to return to V_1. The same is true if we start at a vertex in V_2. Thus every cycle must contain an even number of edges in order to return to either V_1 or V_2.

($\Leftarrow$) Suppose that every cycle in G has even length. Without loss of generality, assume G is connected. We will create a partition of V so that $V = V_1 \cup V_2$ and and $V_1 \cap V_2 = \emptyset$ and there is no edge between vertices if they are in the same class.

Choose an arbitrary vertex $v \in V$ and define:

$$V_1 = \{v' \in V : d_G(v, v') \equiv 0 \mod 2\} \tag{3.4}$$

$$V_2 = \{v' \in V : d_G(v, v') \equiv 1 \mod 2\} \tag{3.5}$$

Clearly V_1 and V_2 constitute a partition of V. Choose $u_1, u_2 \in V_1$ and suppose $e = \{u_1, u_2\} \in E$. The distance from v to u_1 is even, so there is a path p_1 with an even number of edges beginning at v and ending at u_1. Likewise the distance from v to u_2 is even, so there is a path p_2 beginning at u_2 and ending at v with an even number of edges. If we concatenate paths p_1 and the length 1 path $q = (u_1, \{u_1, u_2\}, u_2)$ and path p_2 we obtain a cycle in G that has odd length. Therefore, there can be no edge connecting two vertices in V_1.

Choose $u_1, u_2 \in V_2$ and suppose that $e = \{u_1, u_2\} \in E$. Using the same argument, there is a path p_1 of odd length from v to u_1 and a path p_2 of odd length from u_2 to v. If we concatenate paths p_1 and the length 1 path $q = (u_1, \{u_1, u_2\}, u_2)$ and path p_2 we again obtain a cycle in G that has odd length. Therefore, there can be no edge connecting two vertices in V_2. These arguments are illustrated in Figure 3.10

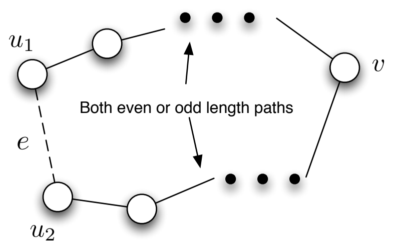


Figure 3.10. Illustration of the main argument in the proof that a graph is bipartite if and only if all cycles have even length.

In the case when G has more than one component, execute the process described above for each component to obtain partitions $V_1, V_2, V_3, V_4, \dots, V_{2n}$. Create a bipartition U_1 and U_2 of V with:

$$U_1 = \bigcup_{k=1}^{n} V_{2k-1} \tag{3.6}$$

$$U_2 = \bigcup_{k=1}^{n} V_{2k} \tag{3.7}$$

Clearly there can be no edge connecting a vertex in U_1 with a vertex in U_2. This completes the proof. □

6. Acyclic Graphs and Trees

Definition 3.53 (Acyclic Graph). A graph that contains *no* cycles is called *acyclic*.

Definition 3.54 (Forests and Trees). Let $G = (V, E)$ be an acyclic graph. If G has more than one component, then G is called a *forest*. If G has one component, then G is called a *tree*.

Example 3.55. A randomly generated tree with 10 vertices is shown in Figure 3.11. Note that a tree (if drawn upside down) can be made to look exactly like a real tree growing up from the ground.

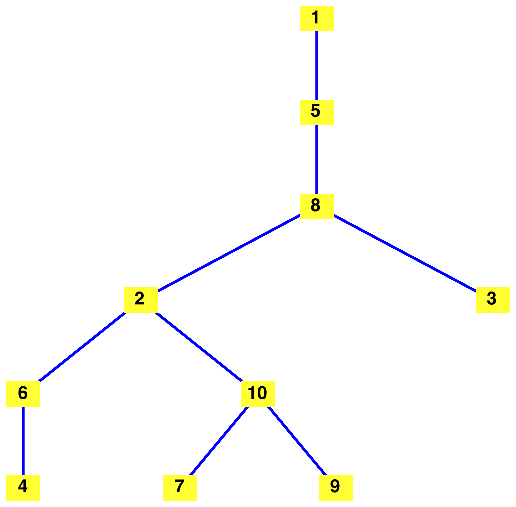


Figure 3.11. A tree is shown. Imagining the tree upside down illustrates the tree like nature of the graph structure.

Remark 3.56. We can define directed trees and directed forests as acyclic directed graphs. Generally speaking, we require the underlying graphs to be acyclic rather than just having no directed cycles. See Chapter 4 of [Gri11a] (http://www.personal.psu.edu/cxg286/Math486.pdf) for the use of directed trees in Game Theory. For the remainder of this chapter we will deal undirected trees, but results will apply to directed trees unless otherwise noted.

Definition 3.57 (Spanning Forest). Let $G = (V, E)$ be a graph. If $F = (V', E')$ is an acyclic subgraph of G such that $V = V'$ then F is called a *spanning forest* of G. If F has exactly one component, then F is called a *spanning tree*.

Example 3.58. The Petersen Graph is illustrated in Figure 3.12 as is a spanning tree for the Petersen Graph. Since the Petersen Graph is connected it is easy to see we do not *need* a spanning forest to construct an acyclic spanning subgraph.

Theorem 3.59. *If $G = (V, E)$ is a connected graph, then there is a spanning tree $T = (V, E')$ of G.*

Proof. We proceed by induction on the number of vertices in G. If $|V| = 1$, then G is itself a (degenerate) tree and thus has a spanning tree. Now, suppose that the statement is true for all graphs G with $|V| \leq n$. Consider a graph G with $n + 1$ vertices. Choose an arbitrary vertex v_{n+1} and remove it and all edges of the form $\{v, v_{n+1}\}$ from G to form G' with vertex set $V' = \{v_1, \dots, v_n\}$. The graph G' has n vertices and may have $m \geq 1$ components ($m > 1$ if v_{n+1} was a cut-vertex). By the induction hypothesis there is a spanning tree for each component of G' since each of these components has at most n vertices. Let $T_1, \dots, T_m$ be the spanning trees for these components.

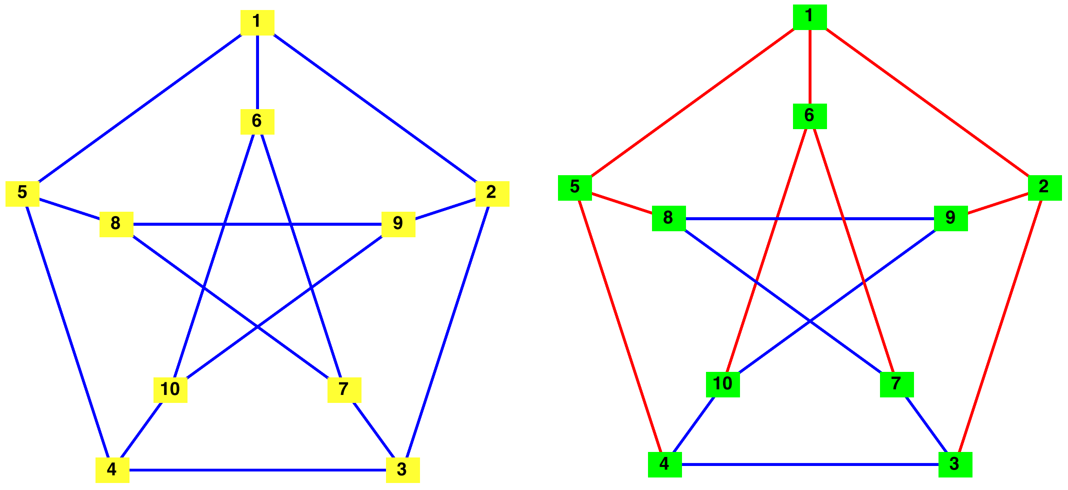


Figure 3.12. The Petersen Graph is shown on the left while a spanning tree is shown on the right in red.

Let T' be the acyclic subgraph of G consisting of all the components' spanning trees. For each spanning tree, choose exactly one edge of the form $e^{(i)} = \{v_{n+1}, v^{(i)}\}$ where $v^{(i)}$ is a vertex in component i and add this edge to T'. It is easy to see that no cycle is created in T through these operations because, by construction, each edge $e^{(i)}$ is a cut-edge and by Corollary 3.43 it cannot lie on a cycle. The graph T contains every vertex of G and is connected and acyclic. Therefore it is a spanning tree of G. The theorem then follows by induction. □

Corollary 3.60. *Every graph $G = (V, E)$ has a spanning forest $F = (V, E')$.*

Exercise 22. Prove Corollary 3.60.

Definition 3.61 (Leaf). Let $T = (V, E)$. If $v \in V$ and $\deg(v) = 1$, then v is called a *leaf* of T.

Lemma 3.62. *Every tree with one edge has at least two leaves.*

Proof. Let:

$$w = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1})$$

be a path of maximal length in T. Consider vertex v_{n+1}. If $\deg(v_{n+1}) > 1$, then there are two possibilities: (i) there is an edge e_{n+1} and a vertex v_{n+2} with v_{n+2} *not* in the sequence w. In this case, we can extend w to w' defined as:

$$w' = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1}, e_{n+1}, v_{n+2})$$

which contradicts our assumption that w was maximal in length. (ii) there is an edge e_{n+1} and a vertex v_{n+2} and for some $k \in \{1, \dots, n\}$, $v_{n+2} = v_k$; i.e., v_{n+2} is in the sequence w. In this case, there is a closed sub-walk:

$$w' = (v_k, e_k, v_{k+1}, \dots, v_{n+1}, e_{n+1}, v_{n+2})$$

Since w is a path, there are no other repeated vertices in the sequence w' and thus w' is a cycle in T, contradicting our assumption that T was a tree. The reasoning above holds for vertex v_1 as well, thus the two end points of every maximal path in a tree must be leaves. This completes the proof. □

Corollary 3.63. *Let $G = (V, E)$ be a graph. If each vertex in V has degree at least 2, then G contains a cycle.*

Exercise 23. Prove Corollary 3.63.

Lemma 3.64. *Let $T = (V, E)$ be a tree with $|V| = n$. Then $|E| = n - 1$.*

Proof. We will proceed by induction. For the case when $n = 1$, this statement must be true. Now, suppose that the statement is true $|V| \leq n$. We will show that when $|V| = n + 1$, then $|E| = n$, assuming that $T = (V, E)$ is a tree. By Lemma 3.62 we know that if T is a tree, then it contains one component and at least two leaves. Therefore, choose a vertex $v \in V$ that is a leaf in T. There is some edge $e = \{v', v\} \in E$. Consider the graph $T' = (V', E')$ with: $V' = V \setminus \{v\}$ and $E' = E \setminus \{e\}$. This new graph T' must:

(1) have one component since v was connected to only one other vertex $v' \in V$ and T had only one component and
(2) by acyclic since T itself was acyclic and we have not introduced new edges to create a cycle.

Therefore T' is a tree with n vertices and by the induction hypothesis it must contain $n - 1$ edges. Since we removed exactly one edge (and one vertex) to construct T' from T it follows that T had exactly n edges and our originally assumed $n + 1$ vertices. The required result follows immediately from induction. □

Corollary 3.65. *If $G = (V, E)$ is a forest with n vertices, then G has $n - c(G)$ edges. (Recall $c(G)$ is the number of components in G).*

Exercise 24. Prove Corollary 3.65.

Theorem 3.66. *A graph $G = (V, E)$ is connected if and only if it has a spanning tree.* □

Exercise 25. Prove Theorem 3.66.

Theorem 3.67. *Let $T = (V, E)$ be a graph with $|V| = n$. Then the following are equivalent:*

(1) *T is a tree.*
(2) *T is acyclic and has exactly $n - 1$ edges.*
(3) *T is connected and has exactly $n - 1$ edges.*
(4) *T is connected and every edge is a cut-edge.*
(5) *Any two vertices of T are connected by exactly one path.*
(6) *T is acyclic and the addition of any new edge creates exactly one cycle in the resulting graph.*

Proof. ($1 \implies 2$) Assume T is a tree. Then by definition T is acyclic and the fact that it has $n - 1$ edges follows from Lemma 3.64.

($2 \implies 3$) Since T is acyclic, it must be a forest and by Corollary 3.65 $|E| = n - c(T)$. Since we assumed that T has $n - 1$ edges, we must have $n - c(T) = n - 1$ and thus the number of components of T is 1 and thus T must be connected.

($3 \implies 4$) The fact that T is connected is assumed from 3. Suppose we consider the graph $T' = (V, E')$ where $E' = E \backslash \{e\}$. Then the number of edges in T' is $n-2$. The graph T' contains n vertices and must still be acyclic (that is a forest) and therefore $n-2 = n - c(T')$. Thus $c(T') = 2$ and e was a cut-edge.

($4 \implies 5$) Choose two vertices v and v' in V. The fact that there is a path between v and v' is guaranteed by our assumption that T is connected. By way of contradiction, suppose that there are at least two paths from v to v' in T. These two paths must diverge at some vertex $w \in V$ and recombine at some other vertex w'. (See Figure 3.13.) We can construct a cycle in T by beginning at vertex w following the first path to w' and the following the second path back to w from w'.

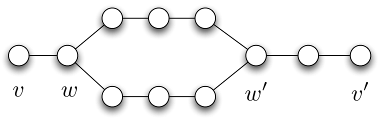


Figure 3.13. The proof of $4 \implies 5$ requires us to assume the existence of two paths in graph T connecting vertex v to vertex v'. This assumption implies the existence of a cycle, contradicting our assumptions on T.

By Theorem 3.42 removing any edge in this cycle cannot result in a disconnected graph. Thus, no edge in the constructed cycle in a cut-edge, contradicting our assumption on T. Thus, two paths connecting v and v' cannot exist.

($5 \implies 6$) The fact that any pair of vertices is connected in T implies T is connected (i.e., has one component). Now suppose that T has a cycle (like the one illustrated in Figure 3.13). Then it is easy to see there are (at least) two paths connecting w and w' contradicting our assumption. Therefore, T is acyclic. The fact that adding an edge creates exactly one cycle can be seen in the following way: Consider two vertices v and v' and suppose the edge $\{v, v'\}$ is not in E. We know there is a path:

$$(v, \{v, u_1\}, u_1, \dots, u_n, \{u_n, v'\}, v')$$

in T connecting v and v' and it is unique. Adding the edge $\{v, v'\}$ creates the cycle:

$$c_1 = (v, \{v, u_1\}, u_1, \dots, u_n, \{u_n, v'\}, v', \{v, v'\}, v)$$

so at least one cycle is created. To see that this cycle is unique, note that if there is another cycle present then it must contain the edge $\{v, v'\}$. Suppose that this cycle is:

$$c_2 = (v, \{v, w_1\}, w_1, \dots, w_n, \{w_n, v'\}, v', \{v, v'\}, v)$$

where there is at least one vertex w_i not present in the set $\{u_1, \dots, u_n\}$ (otherwise the two cycles are identical). We now see there must be two disjoint paths connecting v and v', namely:

$$(v, \{v, u_1\}, u_1, \dots, u_n, \{u_n, v'\}, v')$$

and

$$(v, \{v, w_1\}, w_1, \dots, w_n, \{w_n, v'\}, v')$$

this contradicts our assumption on T. Thus the created cycle is unique.

($6 \implies 1$) It suffices to show that T has a single component. Suppose not, there are at least two components of T. Chose two vertices v and v' in V so that these two vertices are not in the same component. Then the edge $e = \{v, v'\}$ is not in E and adding it to E cannot create a cycle. To see why, not that if T' is the graph that results from the addition of e, then e is now a cut-edge. Applying Corollary 3.43 we see that e cannot lie on a cycle and thus the addition of this edge does not create a cycle, contradicting our assumption on T. Thus, T must have a single component. Since it is acyclic and connected, T is a tree. This completes the proof. □

Definition 3.68 (Tree-Graphic Sequence). Recall from Definition 2.35 a tuple $\mathbf{d} = (d_1, \dots, d_n)$ is graphic if there exists a graph G with degree sequence $\mathbf{d}$. The tuple $\mathbf{d}$ is *tree-graphic* if it is both graphic *and* there exists a tree with degree sequence $\mathbf{d}$.

Theorem 3.69. *A degree sequence* $\mathbf{d} = (d_1, \dots, d_n)$ *is tree-graphic if and only if*

(1) $n = 1$ *and*

$$\sum_{i=1}^{n} d_i = 2n - 2 \tag{3.8}$$

(2) $n \geq 2$, $d_i > 0$ *for* $i = 1, \dots, n$ *and Equation 3.8 holds.*

Proof. ($\Rightarrow$) See Exercise 26.

($\Leftarrow$) Now suppose that Equation 3.8 holds. If $n = 1$, then $2n - 2 = 0$ and thus $d_1 = 0$ and $\mathbf{d} = (d_1)$. This is the degree sequence of a degenerate tree with one vertex. We now proceed by induction to establish the remainder of the theorem. If $n = 2$, $2n - 2 = 2$ and if $d_1, d_2 > 0$ then then $d_1 = d_2 = 1$ by necessity. This is the degree sequence for a tree with two vertices joined by a single edge and thus it is a tree-graphic degree sequence. Now assume the statement holds for all integers up to some n. We will show it is true for $n + 1$. Consider a degree sequence $(d_1, \dots, d_{n+1})$ such that:

$$\sum_{i=1}^{n+1} d_i = 2(n + 1) - 2 = 2n$$

We assume that the degrees are ordered (largest first) and positive. Therefore, $d_1 \geq 2$ (because otherwise $d_1 + \cdots + d_{n+1} \leq n + 1$) and that $d_1 \leq n$ (note in the case that $d_1 = n$ we must have $d_2 = d_3 = \cdots = d_{n+1} = 1$). Moreover, if $d_1 = d_2 = \cdots = d_{n-1} = 2$, then $d_n = d_{n+1} = 1$. Since $d_i > 0$ for $i = 1, \dots, n + 1$ from the previous two facts we see that for any positive value of d_1, we must have $d_n = d_{n+1} = 1$ in order to ensure that $d_1 + d_2 + \cdots + d_{n+1} = 2n$. Consider the sequence of degrees:

$$\mathbf{d}' = (d_1 - 1, d_2, \dots, d_n)$$

Since $d_{n+1} = 1$, we can see that $(d_1 - 1) + d_2 + \cdots + d_n = 2n - 2$. Thus, a permutation of $\mathbf{d}'$ to correct the order leads to a tree-graphic sequence by the induction hypothesis. Let T' be the tree that results from this tree-graphic degree sequence and let v_1 be the vertex with degree $d_1 - 1$ in T'. Then by adding a new vertex v_{n+1} to T' along with edge $\{v_1, v_{n+1}\}$ we have constructed a tree T with the original degree sequence $\mathbf{d}$. That is, clearly this new graph T must be connected since T' was connected and we have connected v_{n+1} to the vertices in T'

and it must be acyclic since we have not connected two vertices already in T' with the edge $\{v_1, v_{n+1}\}$. The result follows by induction. □

Exercise 26. Prove the necessity part of Theorem 3.69. [Hint: Use Theorem 2.33.]

Remark 3.70. This final theorem characterizes Eulerian graphs and will be useful later. We use results derived from our study of trees to prove the following theorem.

Theorem 3.71. *Let $G = (V, E)$ be a non-trivial connected graph G with more than one vertex*[3]*. Then the following are equivalent:*

(1) *G is Eulerian.*
(2) *The degree of every vertex in G is even.*
(3) *The set E is the union of the edge sets of a collection of edge-disjoint cycles in G.*

Proof. ($1 \implies 2$) Assume G is Eulerian, then there is an Eulerian tour t of G. Let v be a vertex in G. Each time v is traversed while following the tour, we must enter v by one edge and leave by another. Thus, v must have an even number of edges adjacent to it. If v is the initial (and final) vertex in the tour, then we leave v in the very first step of the tour and return in the last stage, thus the initial (and final) vertex of the tour must also have an even degree. Thus every vertex has an even degree.

($2 \implies 3$) Since G is connected and every vertex has an even degree, it follows that the degree of each vertex is at least 2. Applying Corollary 3.63 we see that G must contain a cycle C. If this cycle includes every edge in G, then (3) is established. Suppose otherwise. Consider the graph G' obtained by removing all edges in C. If we consider C as a subgraph of G, then each vertex in C has exactly two edges adjacent to it. Thus if v is a vertex in the cycle, then removing the edges in C that are adjacent to it will result in a vertex v having 2 fewer edges in G' then it did in G. Since we assumed that every vertex in G had an even degree, it follows that every vertex in G' must also have an even degree (since we removed) either 2 or 0 edges from each vertex in G to obtain G'. We can repeat the previous process of constructing a cycle in G' and if necessary forming G''. Since there are a finite number of edges in G, this process must stop at some point and we will be left with a collection of edge disjoint cycles $\mathcal{C} = \{C, C', \dots\}$ whose union is the entire edge set of G.

($3 \implies 1$) Assume that G is connected and that its edge set is the union of a collection of edge-disjoint cycles. We proceed by induction on the number of cycles. If there is only one cycle, then we simply follow this cycle in either direction to obtain a tour of G. Now suppose that the statement is true for a graph whose edge set is the union of $\leq n$ edge disjoint cycles. We'll show that the statement is true for a graph whose edge set is composed of $n+1$ edge disjoint cycles. Denote the cycles $C_1, \dots, C_{n+1}$. A subgraph G' of G composed of only cycles $C_1, \dots, C_n$ will have m components with $1 \leq m \leq n$. Each component is composed of at most n edge disjoint cycles and therefore applying the induction hypothesis, each has a tour. Denote the components $K_1, \dots, K_m$. The cycle C_{n+1} shares one vertex in common with at least one of these components (and perhaps all of them). Without loss of generality, assume that K_1 is a component sharing a vertex in common with C_{n+1} (if not, reorder the components to make this true). Begin following the tour around K_1 until we encounter the vertex v_1 that component K_1 and C_{n+1} share. At this point, break the tour

[3]Thanks to S. Shekhar for pointing out this requirement. Note, this is not present in Gross and Yellen's book, but should be.

of K_1 and begin traversing C_{n+1} until (i) we return to v_1 or (ii) we encounter a vertex v_2 that is shared by another component (say K_2). In case (i), we complete the tour of K_1 and necessarily we must have completed a tour of the entire graph since it is connected. In case (ii) we follow the tour of K_2 until we return to v_2 and then continue following C_{n+1} until either case (i) occurs or case (ii) occurs again. In either case, we apply the same logic as before. Since there are a finite number of components, this process will eventually terminate with case (i), we complete the tour of K_1 and thus we will have constructed a tour of the entire graph.

This theorem is illustrated in Figure 3.14. This completes the proof. □

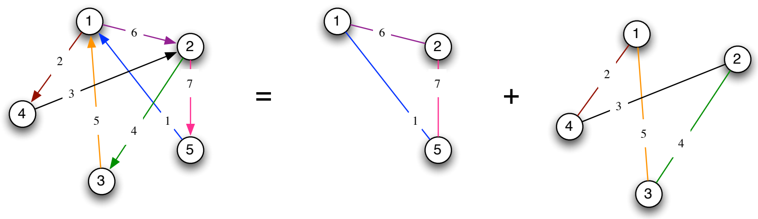


Figure 3.14. We illustrate an Eulerian graph and note that each vertex has even degree. We also show how to decompose this Eulerian graph's edge set into the union of edge-disjoint cycles, thus illustrating Theorem 3.71. Following the tour construction procedure (starting at Vertex 5), will give the illustrated Eulerian tour.

Exercise 27. Show by example that Theorem 3.71 does not necessarily hold if we are only interested in Eulerian *trails*.

CHAPTER 4

Some Algebraic Graph Theory

1. Isomorphism and Automorphism

DEFINITION 4.1 (Injective Mapping). Let S and T be sets. A function $f : S \to T$ is *injective* (sometimes *one-to-one*) if for all $s_1, s_2 \in S$: $f(s_1) = f(s_2) \iff s_1 = s_2$.

DEFINITION 4.2 (Surjective Mapping). Let S and T be sets. A function $f : S \to T$ is *surjective* (sometimes *onto*) if for all $t \in T$ there exists an $s \in S$ such that $f(s) = t$.

DEFINITION 4.3 (Bijective Mapping). Let S and T be sets. A function $f : S \to T$ is *bijective* if f is both injective and surjective.

DEFINITION 4.4 (Isomorphism). Let $G = (V, E)$ and let $G' = (V', E')$. The graphs G and G' are *isomorphic* if there is a bijective mapping $f : V \to V'$ such that for all $v_1, v_2 \in V$ we have:

$$\{v_1, v_2\} \in E \iff \{f(v_1), f(v_2)\} \in E' \tag{4.1}$$

In this case the mapping f is called a *graph isomorphism.* If G and G' are isomorphic, we write $G \cong G'$.

EXERCISE 28. Prove that $\cong$ is an equivalence relation. [Hint: Recall an equivalence relation is a binary relation $\sim$ defined on a set S so that (i) for all $s \in S$, $s \sim s$ (reflexiveness); (ii) for all $s, t \in S$, $s \sim t \iff t \sim s$ (symmetry) and (iii) for all $r, s, t \in T$ $r \sim s$ and $s \sim t$ implies $r \sim t$ (transitivity). Here the set is the set of all graphs.]

DEFINITION 4.5. Let $G = (V, E)$ be a graph. Then the set $\{H : H \cong G\}$ is called the *isomorphism type* (or *isomorphism class*) of G.

THEOREM 4.6. *Suppose $G = (V, E)$ and $G' = (V', E')$ are graphs with $G \cong G'$ with $f : V \to V'$ the graph isomorphism between the graphs. Further suppose that the degree sequence of G is $\mathbf{d}$ and the degree sequence of G' is $\mathbf{d}'$. Then:*

(1) $|V| = |V'|$ *and* $|E| = |E'|$,
(2) *For all* $v \in V$, $\deg(v) = \deg(f(v))$
(3) $\mathbf{d} = \mathbf{d}'$,
(4) *For all* $v \in V$, $\operatorname{ecc}(v) = \operatorname{ecc}(f(v))$
(5) $\omega(G) = \omega(G')$ *(recall $\omega(G)$ is the clique number of G),*
(6) $\alpha(G) = \alpha(G')$ *(recall $\alpha(G)$ is the independence number of G),*
(7) $c(G) = c(G')$ *(recall $c(G)$ is the number of components of G),*
(8) $\operatorname{diam}(G) = \operatorname{diam}(G')$,
(9) $\operatorname{rad}(G) = \operatorname{rad}(G')$
(10) *The girth of G is equal to the girth of G',*
(11) *The circumference of G is equal to the circumference of G'.*

□

Remark 4.7. The proof of Theorem 4.6 is long and should be clear from the definition of isomorphism. Isomorphism is really just a way of *renaming* vertices; we assume that the vertices in graph G are named from the set V, while the vertices in the set G' are named from the set V'. If the graphs are identical except for the names we give the vertices (and thus the names that appear in the edges), then the graphs are isomorphic and all structural properties are preserved as a result of this. We should not that the converse of Theorem 4.6 does not hold. We illustrate this in Exampleex:DegreeIsom.

Example 4.8. Given two graphs G and G', we can see through example that the degree sequence does not uniquely specify the graph G and thus if G and G' have degree sequences $\mathbf{d}$ and $\mathbf{d}'$ it is necessary that $\mathbf{d} = \mathbf{d}'$ when $G \cong G'$ but not sufficient to establish isomorphism. To see this, consider the graphs shown in Figure 4.1. It's clear that $\mathbf{d} = (2, 2, 2, 2, 2, 2) = \mathbf{d}'$,

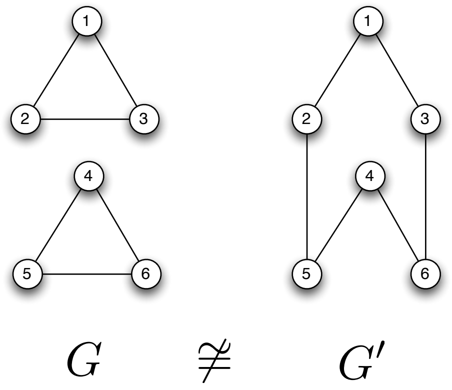


Figure 4.1. Two graphs that have identical degree sequences, but are not isomorphic.

but these graphs cannot be isomorphic, since they have different numbers of components.

The same is true with the other graph properties. The equality between a property of G and that same property for G' is a necessary criterion for the isomorphism of G and G', but not sufficient. We will not encounter any property of a graph that provides such a necessary and sufficient condition.(See Remark 4.12).

Theorem 4.9. *Suppose $G = (V, E)$ and $G' = (V', E')$ are graphs with $G \cong G'$ with $f : V \to V'$ the graph isomorphism between the graphs. If H is a subgraph of G, then $H' = f(H)$ is a subgraph of G'. (Here $f(H)$ is the image of the subgraph H under the isomorphism f.)* □

Exercise 29. Prove Theorem 4.9. [Hint: The proof does not have to be extensive in detail. Simply write enough to convince yourself that the isomorphisms preserve the subgraph property.]

Definition 4.10 (Graph Isomorphism Problem). Given two graphs $G = (V, E)$ and $G' = (V', E')$ the *graph isomorphism problem* is to determine whether or not G and G' are isomorphic.

Definition 4.11 (Subgraph Isomorphism). Given two graphs $G = (V, E)$ and $H = (V', E')$ the *subgraph isomorphism problem* is to determine whether G contains a subgraph that is isomorhic to H.

Remark 4.12. In general, the subgraph isomorphism problem is very (very) hard. In fact, sub-graph isomorphism is a so-called NP-complete problem. (Here the "NP" stands for non-deterministic turing machine solvable in polynomial time.) This is the class of some of the hardest practical problems. Interested readers might consider looking at [**CLRS01**] for more details.

The graph isomorphism problem (interestingly enough) is a bit of an enigma. We do not know exactly how hard this problem is to solve. We do know that it is not quite as hard as the subgraph isomorphism problem. It is worthwhile noting, however, that there is a linear time algorithm for determining the isomorphism of two trees. (See Page 84 of [**AHU74**].)

Exercise 30. List some ways to determine that two graphs are not isomorphic. That is, what are some tests one might do, to see whether two graphs are *not* isomorphic?

Definition 4.13 (Automorphism). Let $G = (V, E)$ be a graph. An *automorphism* is an isomorphism from G to itself. That is, a bijection $f : V \to V$ so that for all $v_1, v_2 \in V$, $\{v_1, v_2\} \in E \iff \{f(v_1), f(v_2)\} \in E$.

Remark 4.14 (Inverse Automorphism). Recall that an isomorphism (and hence an automorphism) is a bijective function and hence it has a well defined inverse. That is, if $G = (V, E)$ is a graph and $f : V \to V$ is an automorphism, then if $f(v_1) = f(v_2)$, we know that $v_1 = v_2$ (because f is injective). Further, we know that for every $v_2 \in V$ there is a (unique) $v_1 \in V$ so that $f(v_1) = v_2$ (because f is surjective). Thus, if $v_2 \in V$ we can define $f^{-1}(v_2)$ to be the unique v_1 so that $f(v_1) = v_2$.

Lemma 4.15. *Let $G = (V, E)$ be a graph. Suppose that $f : V \to V$ is an automorphism. Then $f^{-1} : V \to V$ is also an automorphism.*

Proof. The fact that f is a bijection implies that f^{-1} is itself a bijection. We know for all v_1 and v_2 in V that:

$$\{v_1, v_2\} \in E \iff \{f(v_1), f(v_2)\} \in E$$

For every vertex pair u_1 and u_2 in V there are unique vertices v_1 and v_2 in V so that $u_1 = f(v_1)$ and $u_2 = f(v_2)$. Furthermore, by the previous observation:

$$\{u_1, u_2\} \in E \iff \{v_1, v_2\} \in E$$

But this means that for all u_1 and u_2 in V we have:

$$\{f^{-1}(u_1), f^{-1}(u_2)\} \in E \iff \{u_1, u_2\} \in E \tag{4.2}$$

Thus f^{-1} is a bijection that preserves the edge relation. This completes the proof. □

Exercise 31. Prove carefully that if f is a bijection then so is f^{-1}. [Hint: Most of the proof is in Remark 4.14.]

Lemma 4.16 (Composition). *Let $G = (V, E)$ be a graph. Suppose that $f : V \to V$ and $g : V \to V$ are automorphisms. Then $f \circ g$ is also an automorphism.*

Exercise 32. Prove Lemma 4.16

Definition 4.17 (Group). A *group* is a pair $(S, \circ)$ where S is a set and $\circ : S \times S \to S$ is a binary operation so that:

(1) The binary operation $\circ$ is associative; that is, if s_1, s_2 and s_3 are in S, then $(s_1 \circ s_2) \circ s_3 = s_1 \circ (s_2 \circ s_3)$.
(2) There is a unique identity element $e \in S$ so that for all $s \in S$, $e \circ s = s \circ e = s$.
(3) For every element $s \in S$ there is an inverse element $s^{-1} \in S$ so that $s \circ s^{-1} = s^{-1} \circ s = e$.

If $\circ$ is commutative, that is for all $s_1, s_2 \in S$ we have $s_1 \circ s_2 = s_2 \circ s_1$, then $(S, \circ)$ is called a *commutative group* (or *abelian group*).

Example 4.18. This course is *not* about group theory. If you're interested in groups in the more abstract sense, it's worth considering taking Math 435, which is all about abstract algebra. One of the simplest examples of a group is the set of integers $\mathbb{Z}$ under the binary operation of addition.

Definition 4.19 (Sub-Group). Let $(S, \circ)$ be a group. A *subgroup* of $(S, \circ)$ is a group $(T, \circ)$ so that $T \subseteq S$. The subgroup $(T, \circ)$ shares the identify of the group $(S, \circ)$.

Example 4.20. Consider the group $(\mathbb{Z}, +)$. If $2\mathbb{Z}$ is the set of even integers, then $(2\mathbb{Z}, +)$ is a subgroup of $(\mathbb{Z}, +)$ because that even integers are closed under addition.

Theorem 4.21. *Let $G = (V, E)$ be a graph. Let $\text{Aut}(G)$ be the set of all automorphisms on G. Then $(\text{Aut}(G), \circ)$ is a group.*

Proof. By Lemma 4.16, we can see that functional composition is a binary operation $\circ : \text{Aut}(G) \to \text{Aut}(G)$. Associativity is a property of functional composition, since if $f : V \to V$ and $g : V \to V$ and $h : V \to V$ it is easy to see that for all $v \in V$:

$$((f \circ g) \circ h)(v) = (f \circ g)(h(v)) = f(g(h(v))) = f \circ (g(h(v))) = (f \circ (g \circ h))(v) \tag{4.3}$$

The identity function $e : V \to V$ defined by $e(v) = v$ for all $v \in V$ is an automorphism of V. Finally, by Lemma 4.15, each element of $\text{Aut}(G)$ has an inverse. This completes the proof. □

Definition 4.22 (Permutation / Permutation Group). A *permutation* on a set $V = \{1, \dots, n\}$ of n elements is a bijective mapping f from V to itself. A *permutation group* on a set V is a set of permutations with the binary operation of functional composition.

Example 4.23. Consider the set $V = \{1, 2, 3, 4\}$. A permutation on this set that maps 1 to 2 and 2 to 3 and 3 to 1 can be written as: $(1, 2, 3)(4)$ indicating the cyclic behavior that $1 \to 2 \to 3 \to 1$ and 4 is fixed. In general, we write $(1, 2, 3)$ instead of $(1, 2, 3)(4)$ and suppress any elements that do not move under the permutation.

For the permutation taking 1 to 3 and 3 to 1 and 2 to 4 and 4 to 2 we write $(1, 3)(2, 4)$ and say that this is the *product* of $(1, 3)$ and $(2, 4)$. When determining the impact of a permutation on a number, we read the permutation from right to left. Thus, if we want to determine the impact on 2, we read from right to left and see that 2 goes to 4. By contrast, if we had the permutation: $(1, 3)(1, 2)$ then this permutation would take 2 to 1 first and then 1 to 3 thus 2 would be mapped to 3. The number 1 would be first mapped to 2 and then stop. The number 3 would be mapped to 1. Thus we can see that $(1, 3)(1, 2)$ has the same action as the permutation $(1, 2, 3)$.

Definition 4.24 (Symmetric Group). Consider a set V with n elements in it. The permutation group S_n contains every possible permutation of the set with n elements.

Example 4.25. Consider the set $V = \{1, 2, 3\}$. The symmetric group on V is the set S_3 and it contains the permutations:

1. The identity: (1)(2)(3)
2. (12)(3)
3. (13)(2)
4. (23)(1)
5. (123)
6. (132)

Proposition 4.26. *For each n, $|S_n| = n!$.*

Exercise 33. Prove Proposition 4.26

Definition 4.27 (Transposition). A permutation of the form (a_1, a_2) is called a transposition.

Theorem 4.28. *Every permutation can be expressed as the product of transpositions.*

Proof. Consider the permuation $(a_1, a_2, \dots, a_n)$. We may write:

$$(a_1, a_2, \dots, a_n) = (a_1, a_n)(a_1, a_{n-1}) \cdots (a_1, a_2) \tag{4.4}$$

Observe the effect of these two permutations on a_i. For $i \neq 1$ and $i \neq n$, then reading from right to left (as the permutation is applied) we see that a_i maps to a_1, which reading further right to left is mapped to a_{i+1} as we expect. If $i = 1$, then a_1 maps to a_2 and there is no further mapping. Finally, if $i = n$, then we read left to right to the only transposition containing a_n and see that a_n maps to a_1. Thus Equation 4.4 holds. This completes the proof. □

Remark 4.29. The following theorem is useful for our work on matrices in the second part of this chapter, but its proof is outside the scope of these notes. The interested reader can see Chapter 2.2 of [**Fra99**].

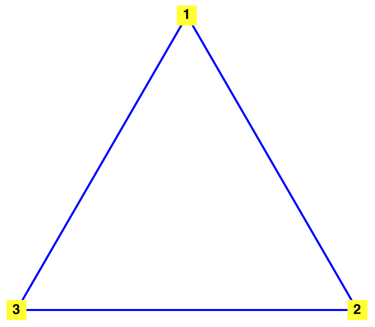
Theorem 4.30. *No permutation can be expressed as both a product of an even and an odd number of transpositions.* □

Definition 4.31 (Even/Odd Permutation). Let $\sigma \in S_n$ be a permutation. If σ can be expressed as an *even* number of transpositions, then it is *even*, otherwise σ is *odd*. The *signature* of the permutation is:

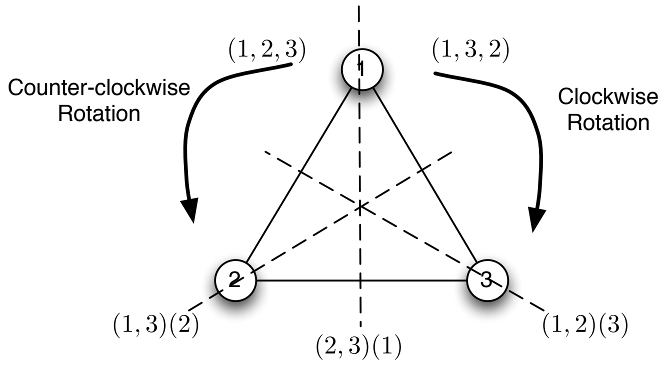
$$\operatorname{sgn}(\sigma) = \begin{cases} -1 & \sigma \text{ is odd} \\ 1 & \sigma \text{ is even} \end{cases} \tag{4.5}$$

Remark 4.32. Let $G = (V, E)$ be a graph. If $f \in \operatorname{Aut}(G)$, then f is a permutation on the vertices of G. Thus the graph automorphism group is just a permutation group that respects vertex adjacency.

Example 4.33. Consider the graph K_3, the complete graph on 3 vertices (see Figure 4.2(a).) The graph K_3 has six automorphisms, one for each element in S_3 the set of all permutations on 3 objects. These automorphisms are (i) the identity automorphism that maps all vertices to themselves, which is the permutation (1)(2)(3); (ii) the automorphism that exchanges vertex 1 and 2, which is the permutation $(1, 2)(3)$; (iii) the automorphism



(a) K_3



(b) *Symmetries*

Figure 4.2. The graph K_3 has six automorphisms, one for each element in S_3 the set of all permutations on 3 objects. These automorphisms are (i) the identity automorphism that maps all vertices to themselves; (ii) the automorphism that exchanges vertex 1 and 2; (iii) the automorphism that exchanges vertex 1 and 3; (iv) the automorphism that exchanges vertex 2 and 3; (v) the automorphism that sends vertex 1 to 2 and 2 to 3 and 3 to 1; and (vi) the automorphism that sends vertex 1 to 3 and 3 to 2 and 2 to 1.

that exchanges vertex 1 and 3, which is the permutation $(1, 3)(2)$; (iv) the automorphism that exchanges vertex 2 and 3, which is the permutation $(1)(2, 3)$; (v) the automorphism that sends vertex 1 to 2 and 2 to 3 and 3 to 1, which is the permutation $(1, 2, 3)$; and (vi) the automorphism that sends vertex 1 to 3 and 3 to 2 and 2 to 1, which is the permutation $(1, 3, 2)$.

Notice that each of these automorphisms is illustrated by a symmetry in the graphical representation of K_3. The permutations $(1, 2)(3)$, $(1, 3)(2)$, and $(2, 3)(1)$ are flips about an axis of symmetry, while the permutations $(1, 2, 3)$ and $(1, 3, 2)$ are rotations. This is illustrated in Figure 4.2(b).

It should be noted, that this method of drawing a graph to find its automorphism group does not work in general, but for some graphs (like complete graphs or cycle graphs) this can be useful.

Exercise 34. Characterize the automorphism group of the cycle graph C_4.

Lemma 4.34. *The automorphism group of K_n is S_n, thus $|\text{Aut}(K_n)| = n!$.*

Exercise 35. Prove Lemma 4.34

Definition 4.35 (Star Graph). A *star graph* on $n + 1$ vertices (unfortunately denoted S_n) is a graph with vertex set $V = \{v_0, \dots, v_n\}$ and edge set E so that:

$$e \in E \iff e = \{v_0, v_i\} \quad i \in \{1, \dots, n\}$$

Thus the graph S_n has $n + 1$ vertices and n edges.

Remark 4.36. It is unfortunate that the symmetric group on n items and star graph with $n + 1$ vertices have the same representation. We will differentiate between the two explicitly to prevent conclusion. It is also worth noting that some references define the star graph S_n to have n vertices and $n-1$ edges.

Example 4.37. The star graph S_3 with 4 vertices and 3 edges is shown in Figure 4.3 as is the graph S_9.

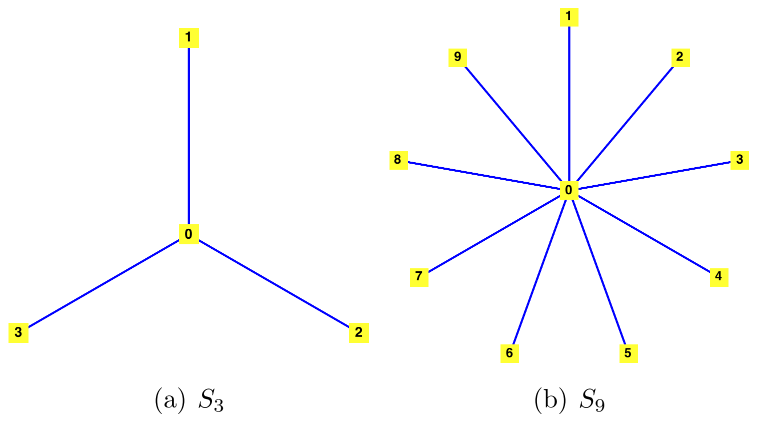


Figure 4.3. The star graphs S_3 and S_9.

Exercise 36. Show that the automorphism group of the star graph S_3 is also identical to the symmetric permutation group S_3. As a result, show that two non-isomorphic graphs can share an automorphism group. (Remember $\text{Aut}(K_3)$ is also the symmetric permutation group on 3 elements.)

Exercise 37 (Project). Study the problem of graph automorphism in detail. Explore the computational complexity of determining the automorphism group of a graph or family of graphs. Explore any automorphism groups for specific types of graphs like cycle graphs, star graphs, hypercubes etc.

2. Fields and Matrices

Definition 4.38 (Field). A *field* is a tuple $(S, +, \cdot, 0, 1)$ where:

(1) $(S, +)$ is a commutative group with unit 0,
(2) $(S \setminus \{0\}, \cdot)$ is a commutative group with unit 1
(3) The operation $\cdot$ *distributes* over the operation $+$ so that if a_1, a_2, and a_3 are elements of F, then $a_1 \cdot (a_2 + a_3) = a_1 \cdot a_2 + a_1 \cdot a_3$.

Example 4.39. The archetypal example of a field is the field of real numbers $\mathbb{R}$ with addition and multiplication playing the expected roles. Another common field is the field of complex numbers $\mathbb{C}$ (numbers of the form $a + bi$ with $i = \sqrt{-1}$ the imaginary unit) with their addition and multiplication rules defined as expected.

Definition 4.40 (Matrix). An $m \times n$ matrix is a rectangular array of values (*scalars*), drawn from a field. If F is the field, we write $F^{m \times n}$ to denote the set of $m \times n$ matrices with entries drawn from F.

REMARK 4.41. For most of the time, we will focus exclusively on matrices with entries from the field $\mathbb{R}$. However, we will make use of other fields when we discuss the so-called *edge space* of a graph at the end of this chapter.

REMARK 4.42. If $\mathbf{A} \in \mathbb{R}^{m\times n}$, then the matrix consists of m rows and n columns. The element in the i^{th} row and j^{th} column of $\mathbf{A}$ is written as $\mathbf{A}_{ij}$. The j^{th} column of $\mathbf{A}$ can be written as $\mathbf{A}_{\cdot j}$, where the $\cdot$ is interpreted as ranging over every value of i (from 1 to m). Similarly, the i^{th} row of $\mathbf{A}$ can be written as $\mathbf{A}_{i\cdot}$. When $m = n$, then the matrix $\mathbf{A}$ is called *square*.

DEFINITION 4.43 (Row/Column Vector). A $1 \times n$ matrix is called a *row vector*, and a $m \times 1$ matrix is called a *column vector*. For the remainder of these notes, every vector will be thought of **column vector** unless otherwise noted.

REMARK 4.44. It should be clear that any row of matrix $\mathbf{A}$ could be considered a row vector in $\mathbb{R}^n$ and any column of $\mathbf{A}$ could be considered a column vector in $\mathbb{R}^m$.

DEFINITION 4.45 (Dot Product). Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors (either row or column) with n elements. Then the *dot* product of x with y is:

$$\mathbf{x}\cdot\mathbf{y} = \sum_{i=1}^{n} x_i y_i \tag{4.6}$$

DEFINITION 4.46. Two vectors $\mathbf{x}$ and $\mathbf{y}$ are *orthogonal* if $\mathbf{x}\cdot\mathbf{y} = 0$. (Here 0 is the zero in the field over which the vectors are defined.)

DEFINITION 4.47 (Matrix Multiplication). If $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, then $\mathbf{C} = \mathbf{AB}$ is the *matrix product* of $\mathbf{A}$ and $\mathbf{B}$ and

$$\mathbf{C}_{ij} = \mathbf{A}_{i\cdot} \cdot \mathbf{B}_{\cdot j} \tag{4.7}$$

Note, $\mathbf{A}_{i\cdot} \in \mathbb{R}^{1\times n}$ (an n-dimensional vector) and $\mathbf{B}_{\cdot j} \in \mathbb{R}^{n\times 1}$ (another n-dimensional vector), thus making the dot product meaningful.

EXAMPLE 4.48.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1(5)+2(7) & 1(6)+2(8) \\ 3(5)+4(7) & 3(6)+4(8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \tag{4.8}$$

DEFINITION 4.49 (Matrix Transpose). If $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a $m \times n$ matrix, then the *transpose* of $\mathbf{A}$ dented $\mathbf{A}^T$ is an $m \times n$ matrix defined as:

$$\mathbf{A}^T_{ij} = \mathbf{A}_{ji} \tag{4.9}$$

EXAMPLE 4.50.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \tag{4.10}$$

The matrix transpose is a particularly useful operation and makes it easy to transform column vectors into row vectors, which enables multiplication. For example, suppose $\mathbf{x}$ is an $n \times 1$ column vector (i.e., $\mathbf{x}$ is a vector in $\mathbb{R}^n$) and suppose $\mathbf{y}$ is an $n \times 1$ column vector. Then:

$$\mathbf{x}\cdot\mathbf{y} = \mathbf{x}^T\mathbf{y} \tag{4.11}$$

EXERCISE 38. Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$. Prove by example that $\mathbf{AB} \neq \mathbf{BA}$; that is, matrix multiplication is *not commutative*. [Hint: Almost any pair of matrices you pick (that can be multiplied) will not commute.]

EXERCISE 39. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and let, $\mathbf{B} \in \mathbb{R}^{n \times p}$. Use the definitions of matrix multiplication and transpose to prove that:

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T \tag{4.12}$$

[Hint: Note that $\mathbf{C}_{ij} = \mathbf{A}_{i\cdot} \cdot \mathbf{B}_{\cdot j}$, which moves to the (j, i) position. Now figure out what is in the (j, i) position of $\mathbf{B}^T\mathbf{A}^T$.]

3. Special Matrices and Vectors

DEFINITION 4.51 (Identify Matrix). The $n \times n$ *identify matrix* is:

$$\mathbf{I}_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{4.13}$$

DEFINITION 4.52 (Zero Matrix). The $n \times n$ *zero* matrix an $n \times n$ consisting entirely of 0.

EXERCISE 40. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$. Show that $\mathbf{AI}_n = \mathbf{I}_n\mathbf{A} = \mathbf{A}$. Hence, $\mathbf{I}$ is an identify for the matrix multiplication operation on square matrices. [Hint: Do the multiplication out long hand.]

DEFINITION 4.53 (Symmetric Matrix). Let $\mathbf{M} \in \mathbb{R}^{n \times n}$ be a matrix. The matrix $\mathbf{M}$ is symmetric if $\mathbf{M} = \mathbf{M}^T$.

DEFINITION 4.54 (Invertible Matrix). Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square matrix. If there is a matrix $\mathbf{A}^{-1}$ such that

$$\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n \tag{4.14}$$

then matrix $\mathbf{A}$ is said to be *invertible* (or *nonsingular*) and $\mathbf{A}^{-1}$ is called its *inverse*. If $\mathbf{A}$ is not invertible, it is called a *singular* matrix.

4. Matrix Representations of Graphs

DEFINITION 4.55 (Adjacency Matrix). Let $G = (V, E)$ be a graph and assume that $V = \{v_1, \dots, v_n\}$. The *adjacency matrix* of G is an $n \times n$ matrix $\mathbf{M}$ defined as:

$$\mathbf{M}_{ij} = \begin{cases} 1 & \{v_i, v_j\} \in E \\ 0 & \text{else} \end{cases}$$

PROPOSITION 4.56. *The adjacency matrix of a (simple) graph is symmetric.*

EXERCISE 41. Prove Proposition 4.56.

THEOREM 4.57. *Let $G = (V, E)$ be a graph with $V = \{v_1, \dots, v_n\}$ and let M be its adjacency matrix. For $k \geq 0$, the (i, j) entry of $\mathbf{M}^k$ is the number of walks of length k from v_i to v_j.*

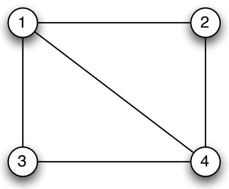


Figure 4.4. The adjacency matrix of a graph with n vertices is an $n \times n$ matrix with a 1 at element (i, j) if and only if there is an edge connecting vertex i to vertex j; otherwise element (i, j) is a zero.

Proof. We will proceed by induction. By definition, $\mathbf{M}^0$ is the $n \times n$ identity matrix and the number of walks of length 0 between v_i and v_j is 0 if $i \neq j$ and 1 otherwise, thus the base case is established.

Now suppose that the (i, j) entry of $\mathbf{M}^k$ is the number of walks of length k from v_i to v_j. We will show this is true for $k + 1$. We know that:

$$\mathbf{M}^{k+1} = \mathbf{M}^k\mathbf{M} \tag{4.15}$$

Consider vertices v_i and v_j. The (i, j) element of $\mathbf{M}^{k+1}$ is:

$$\mathbf{M}^{k+1}_{ij} = \left(\mathbf{M}^k_{i\cdot}\right)^T \mathbf{M}_{\cdot j} \tag{4.16}$$

Let:

$$\mathbf{M}^k_{i\cdot} = \begin{bmatrix} r_1 & \dots & r_n \end{bmatrix} \tag{4.17}$$

where r_l, $(l = 1, \dots, n)$, is the number of walks of length k from v_i to v_l by the induction hypothesis. Let:

$$\mathbf{M}_{\cdot j} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \tag{4.18}$$

where b_l, $(l = 1, \dots, n)$, is a 1 if and only there is an edge $\{v_l, v_j\} \in E$ and 0 otherwise. Then the (i, j) term of $\mathbf{M}^{k+1}$ is:

$$\mathbf{M}^{k+1}_{ij} = \mathbf{M}^k_{i\cdot}\mathbf{M}_{\cdot j} = \sum_{l=1}^{n} r_l b_l \tag{4.19}$$

This is the total number of walks of length k leading to a vertex v_l, $(l = 1, \dots, n)$, from vertex v_i such that there is also an edge connecting v_l to v_j. Thus $\mathbf{M}^{k+1}_{ij}$ is the number of walks of length $k + 1$ from v_i to v_j. The result follows by induction. □

Example 4.58. Consider the graph in Figure 4.4. The adjacency matrix for this graph is:

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{4.20}$$

Consider $\mathbf{M}^2$:

$$\mathbf{M}^2 = \begin{bmatrix} 3 & 1 & 1 & 2 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 3 \end{bmatrix} \tag{4.21}$$

This tells us that there are three distinct walks of length 2 from vertex v_1 to itself. These walks are obvious:

(1) $(v_1, \{v_1, v_2\}, v_2, \{v_1, v_2\}, v_1)$
(2) $(v_1, \{v_1, v_2\}, v_3, \{v_1, v_3\}, v_1)$
(3) $(v_1, \{v_1, v_4\}, v_4, \{v_1, v_4\}, v_1)$

We also see there is 1 path of length 2 from v_1 to v_2: $(v_1, \{v_1, v_4\}, v_4, \{v_2, v_4\}, v_2)$. We can verify each of the other numbers of paths in $\mathbf{M}^2$.

Exercise 42. Devise an inefficient test for isomorphism between two graphs G and G' using their adjacency matrix representations. Assume it takes 1 time unit to test whether two $n \times n$ matrices are equal. What is the maximum amount of time your algorithm takes to determine that $G \not\cong G'$? [Hint: Continue to re-order the vertices of G' and test the adjacency matrices for equality.]

Definition 4.59 (Directed Adjacency Matrix). Let $G = (V, E)$ be a directed graph and assume that $V = \{v_1, \dots, v_n\}$. The *adjacency matrix* of G is an $n \times n$ matrix $\mathbf{M}$ defined as:

$$\mathbf{M}_{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & \text{else} \end{cases}$$

Theorem 4.60. *Let $G = (V, E)$ be a digraph with $V = \{v_1, \dots, v_n\}$ and let M be its adjacency matrix. For $k \geq 0$, the (i, j) entry of $\mathbf{M}^k$ is the number of directed walks of length k from v_i to v_j.*

Exercise 43. Prove Theorem 4.60. [Hint: Use the approach in the proof of Theorem 4.57.]

Definition 4.61 (Incidence Matrix). Let $G = (V, E)$ be a graph with $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Then the *incidence matrix* of G is an $m \times n$ matrix $\mathbf{A}$ with:

$$\mathbf{A}_{ij} = \begin{cases} 0 & \text{if } v_i \text{ is not in } e_j \\ 1 & \text{if } v_i \text{ is in } e_j \text{ and } e_j \text{ is not a self-loop} \\ 2 & \text{if } v_i \text{ is in } e_j \text{ and } e_j \text{ is a self-loop} \end{cases} \tag{4.22}$$

Theorem 4.62. *Let $G = (V, E)$ be a graph with $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$ with incidence matrix $\mathbf{A}$. The sum of every column in $\mathbf{A}$ is 2 and the sum of each row in $\mathbf{A}$ is the degree of the vertex corresponding to that row.*

Proof. Consider any column in $\mathbf{A}$; it corresponds to an edge e of G. If the edge is a self-loop, there is only one vertex adjacent to e and thus only one non-zero entry in this column. Therefore its sum is 2. Conversely, if e connects two vertices, then there are precisely two vertices adjacent to e and thus two entries in this column that are non-zero both with value 1, thus again the sum of the column is 2.

Now consider any row in $\mathbf{A}$; it corresponds to a vertex v of G. The entries in this row are 1 if there is some edge that is adjacent to v and 2 if there is a self-loop at v. From Definition 2.8, we see that adding these values up yields the degree of the vertex v. This completes the proof. □

Exercise 44. Use Theorem 4.62 to prove Theorem 2.33 a new way.

Definition 4.63. Let $G = (V, E)$ be a digraph with $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Then the *incidence matrix* of G is an $m \times n$ matrix $\mathbf{A}$ with:

$$\mathbf{A}_{ij} = \begin{cases} 0 & \text{if } v_i \text{ is not in } e_j \\ 1 & \text{if } v_i \text{ is the source of } e_j \text{ and } e_j \text{ is not a self-loop} \\ -1 & \text{if } v_i \text{ is the destination of } e_j \text{ and } e_j \text{ is not a self-loop} \\ 2 & \text{if } v_i \text{ is in } e_j \text{ and } e_j \text{ is a self-loop} \end{cases} \tag{4.23}$$

Remark 4.64. The adjacency matrices of simple directed graphs (those with no self-loops) have very useful properties, which we will come to when we study network flows. In particular, these matrices have the property that every square sub-matrix has a determinant that is either 1, -1 or 0. This property is called total unimodularity and it is particularly important in the analysis of network flows.

5. Determinants, Eigenvalue and Eigenvectors

Definition 4.65 (Determinant). Let $\mathbf{M} \in \mathbb{R}^{n\times n}$. The *determinant* of $\mathbf{M}$ is:

$$\det(\mathbf{A}) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} \mathbf{A}_{i\sigma(i)} \tag{4.24}$$

Here $\sigma \in S_n$ represents a permutation over the set $\{1, \dots, n\}$ and $\sigma(i)$ represents the value to which i is mapped under σ.

Example 4.66. Consider an arbitrary 2×2 matrix:

$$\mathbf{M} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

There are only two permutations in the set S_2: the identity permutation (which is even) and the transposition $(1, 2)$ which is odd. Thus, we have:

$$\det(\mathbf{M}) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = \mathbf{M}_{11}\mathbf{M}_{22} - \mathbf{M}_{12}\mathbf{M}_{21} = ad - bc$$

This is the formula that one would expect from a course in matrices (like Math 220).

Definition 4.67 (Eigenvalue and (Right) Eigenvector). Let $\mathbf{M} \in \mathbb{R}^{n\times n}$. An eigenvalue, eigenvector pair $(\lambda, \mathbf{x})$ is a scalar and $n \times 1$ vector such that:

$$\mathbf{Mx} = \lambda\mathbf{x} \tag{4.25}$$

Remark 4.68. A *left eigenvector* is defined analogously with $\mathbf{x}^T\mathbf{M} = \lambda\mathbf{x}^T$, when $\mathbf{x}$ is considered a column vector. We will deal exclusively with right eigenvectors and hence when we say "eigenvector" we mean a right eigenvector.

Definition 4.69 (Characteristic Polynomial). If $\mathbf{M} \in \mathbb{R}^{n\times n}$ then its *characteristic polynomial* is:

$$\det\left(\lambda\mathbf{I}_n - \mathbf{M}\right) \tag{4.26}$$

Remark 4.70. The following theorem is useful for computing eigenvalues of small matrices and defines the characteristic polynomial for a matrix. Its proof is outside the scope of these notes, but would occur in a Math 436 class. (See Chapter 8.2 of [Lan87].)

Theorem 4.71. *A value λ is an eigenvalue for $\mathbf{M} \in \mathbb{R}^{n\times n}$ if and only if it satisfies the characteristic equation:*

$$\det\left(\lambda\mathbf{I}_n - \mathbf{M}\right) = 0$$

Furthermore, $\mathbf{M}$ and $\mathbf{M}^T$ share eigenvalues. □

Example 4.72. Consider the matrix:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

The characteristic polynomial is computed as:

$$\det\left(\lambda\mathbf{I}_n - \mathbf{M}\right) = \begin{vmatrix} \lambda - 1 & 0 \\ 0 & \lambda - 2 \end{vmatrix} = (\lambda - 1)(\lambda - 2) - 0 = 0$$

Thus the characteristic polynomial for this matrix is:

$$\lambda^2 - 3\lambda + 2 \tag{4.27}$$

The roots of this polynomial are $\lambda_1 = 1$ and $\lambda_2 = 2$. Using these eigenvalues, we can compute eigenvectors:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.28}$$

$$\mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.29}$$

and observe that:

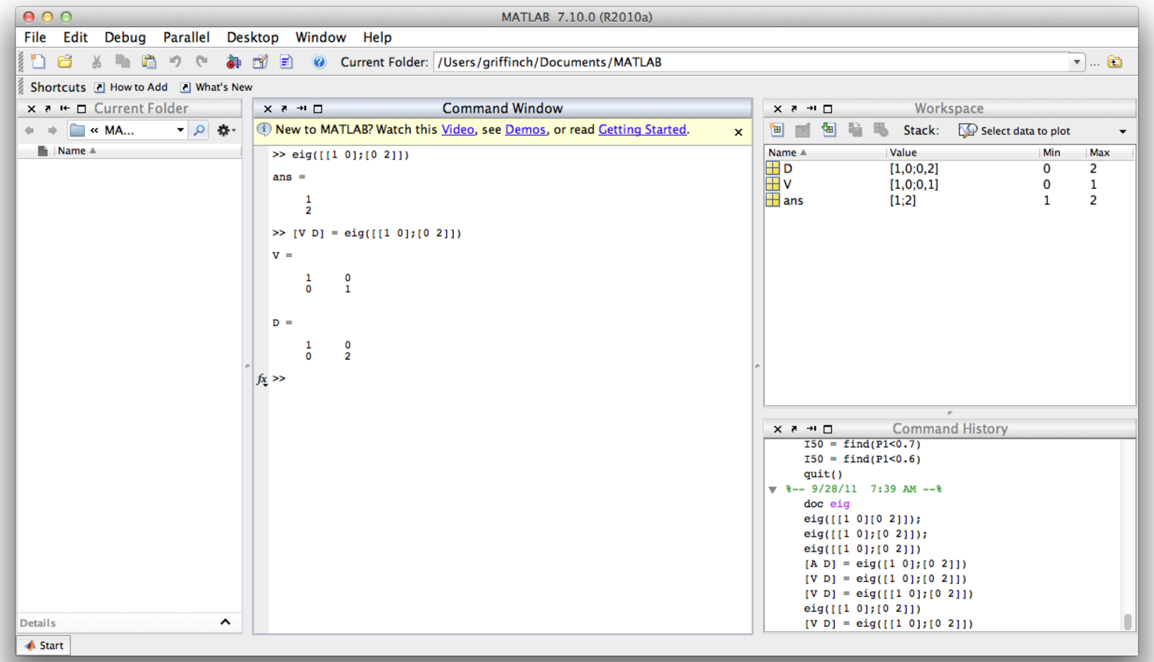
$$\mathbf{M}\mathbf{x}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \lambda_1\mathbf{x}_1 \tag{4.30}$$

and

$$\mathbf{M}\mathbf{x}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = 2\begin{bmatrix} 0 \\ 1 \end{bmatrix}\lambda_2\mathbf{x}_2 \tag{4.31}$$

as required. Computation of eigenvalues and eigenvectors is usually accomplished by computer and several algorithms have been developed. Those interested readers should consult (e.g.) [Dat95].

Example 4.73. You can use Matlab to compute the eigenvalues of a matrix using the `eig` command. The same command can also return the eigenvalues (as the diagonals of a matrix) and the corresponding eigenvectors in a second matrix. An example is shown in Figure 4.5. This command will return the eigenvalues when used as: `d = eig(A)` and


MATLAB 7.10.0 (R2010a)
File Edit Debug Parallel Desktop Window Help
Current Folder: /Users/griffinch/Documents/MATLAB
Shortcuts How to Add What's New
Current Folder
« MA...
Name
Details
Start
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Workspace
Stack: Select data to plot
Command History

```
>> eig([[1 0];[0 2]])

ans =

     1
     2

>> [V D] = eig([[1 0];[0 2]])

V =

     1     0
     0     1


D =

     1     0
     0     2

>>
```

Name	Value	Min	Max
D	[1,0;0,2]	0	2
V	[1,0;0,1]	0	1
ans	[1;2]	1	2

```
    I50 = find(P1<0.7)
    I50 = find(P1<0.6)
    quit()
%-- 9/28/11  7:39 AM --%
    doc eig
    eig([[1 0][0 2]]);
    eig([[1 0];[0 2]]);
    eig([[1 0];[0 2]])
    [A D] = eig([1 0];[0 2]])
    [V D] = eig([1 0];[0 2]])
    [V D] = eig([[1 0];[0 2]])
    eig([[1 0];[0 2]])
    [V D] = eig([[1 0];[0 2]])
```

Figure 4.5. Computing the eigenvalues and eigenvectors of a matrix in Matlab can be accomplished with the *eig* command. This command will return the eigenvalues when used as: `d = eig(A)` and the eigenvalues and eigenvectors when used as `[V D] = eig(A)`. The eigenvectors are the columns of the matrix `V`.

the eigenvalues and eigenvectors when used as `[V D] = eig(A)`. The eigenvectors are the columns of the matrix `V`.

Remark 4.74. It is important to remember that eigenvectors are unique *up to scale*. That is, if $\mathbf{M}$ is a square matrix and $(\lambda, \mathbf{x})$ is an eigenvalue eigenvector pair for $\mathbf{M}$, then so is $(\lambda, \alpha\mathbf{x})$ for $\alpha \neq 0$. This is because:

$$\mathbf{M}\mathbf{x} = \lambda\mathbf{x} \implies \mathbf{M}(\alpha\mathbf{x}) = \lambda(\alpha\mathbf{x}) \tag{4.32}$$

Definition 4.75 (Degenerate Eigenvalue). An eigenvalue is *degenerate* if it is a *multiple root* of the characteristic polynomial. The multiplicity of the root is the multiplicity of the eigenvalue.

Example 4.76. Consider the identify matrix $\mathbf{I}_2$. It has characteristic polynomial $(\lambda - 1)^2$, which has one multiple root 1. Thus $\lambda = 1$ is a degenerate eigenvalue for this matrix. However, this matrix does have two eigenvectors $[1 \;\; 0]^T$ and $[0 \;\; 1]^T$.

REMARK 4.77. The theory of eigenvalues and eigenvectors of matrices is deep and well understood. A substantial part of this theory should be covered in Math 436, for those interested. We will use only a few result in our study of graphs. The following results are proved in Chapter 8 of [GR01]. Unfortunately, the proofs are well outside the scope of the class.

THEOREM 4.78 (Spectral Theorem for Real Symmetric Matrices). *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be a symmetric matrix. Then the eigenvalues of* $\mathbf{M}$ *are all real.* □

6. Properties of the Eigenvalues of the Adjacency Matrix

LEMMA 4.79. *Let* $a_n x^n + \cdots + a_1 x + a_0$ *for* $x = p/q$ *with* $\gcd(p, q) = 1$ *and* $a_n, \dots, a_0 \in \mathbb{Z}$. *Then* p *is divisible by* a_0 *and* q *is divisible by* a_n. □

THEOREM 4.80. *Let* $G = (V, E)$ *be a graph with adjacency matrix* $\mathbf{M}$. *Then:*

(1) *Every eigenvalue of* M *is real and*
(2) *If* λ *is a rational eigenvalue of* M, *then it is integer.*

□

EXERCISE 45 (Project). Prove the Spectral Theorem for Real Symmetric Matrics and then use it to obtain Part 1 of Theorem 4.80. Then prove and apply Lemma 4.79 to prove Part 2 of Theorem 4.80. You should discuss the proof of the Spectral Theorem for Hermitian Matrices. [Hint: All these proofs are available in references or online, expand on these sources in your own words.]

REMARK 4.81. Two graphs that are not isomorphic can have the same set of eigenvalues. This can be illustrated through an example that can be found in Chapter 8 of [GR01]. The graphs are shown in Figure 4.6. We can see the two graphs are not isomorphic since there

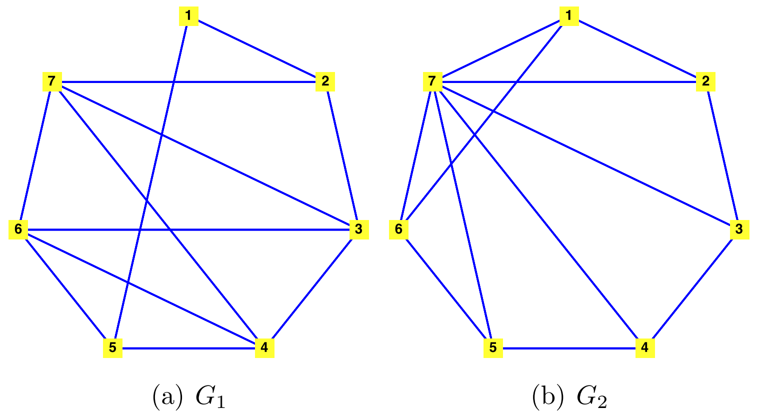


Figure 4.6. Two graphs with the same eigenvalues that are not isomorphic are illustrated.

is no vertex in Graph G_1 that has a degee of 6 unlike Vertex 7 of graph G_2. The adjacency matrices for the two graphs are:

$$\mathbf{M}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{M}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

However, (using a computer) one can determine that they share the same set of eigenvalues:

$$\{-2, 1-\sqrt{7}, 1+\sqrt{7}, 1, -1, 1, -1\}$$

Definition 4.82 (Irreducible Matrix). A matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ is *irreducible* if for each (i, j) pair, there is some $k \in \mathbb{Z}$ with $k > 0$ so that $\mathbf{M}^k_{ij} > 0$.

Lemma 4.83. *If $G = (V, E)$ is a connected graph with adjacency matrix $\mathbf{M}$, then $\mathbf{M}$ is irreducible.*

Exercise 46. Prove Lemma 4.83.

Theorem 4.84 (Perron-Frobenius Theorem). *If $\mathbf{M}$ is an irreducible matrix, then $\mathbf{M}$ has an eigenvalue λ_0 with the following properties:*

1. *The eigenvalue λ_0 is positive and if λ is an alternative eigenvalue of $\mathbf{M}$, then $\lambda_0 \geq |\lambda|$,*
2. *The matrix $\mathbf{M}$ has an eigenvectors $\mathbf{v}_0$ corresponding to λ_0 with only positive entries,*
3. *The eigenvalue λ is a simple root of the characteristic equation for $\mathbf{M}$ and therefore has a unique (up to scale) eigenvectors $\mathbf{v}_0$.*
4. *The eigenvector $\mathbf{v}_0$ is the only eigenvector of $\mathbf{M}$ that can have all positive entries when properly scaled.*

Remark 4.85. The Perron-Frobenius theorem is a classical result in Linear Algebra with several proofs (see [Mey01]). Meyer says of the theorem,

In addition to saying something useful, the PerronFrobenius theory is elegant. It is a testament to the fact that beautiful mathematics eventually tends to be useful, and useful mathematics eventually tends to be beautiful.

One should note that you can say more than we state in this version of the Perron-Frobenius Theorem. See Chapter 8 of [**Mey01**] for details and a proof.

Corollary 4.86. *If $G = (V, E)$ is a connected graph with adjacency matrix $\mathbf{M}$, then it has a unique largest eigenvalue which corresponds to an eigenvector that is positive when properly scaled.*

Proof. Applying Lemma 4.83 we see that $\mathbf{M}$ is irreducible. Further, we know that there is an eigenvalue λ_0 of $\mathbf{M}$ that is (i) greater than or equal to in absolute value all other eigenvalues of $\mathbf{M}$ and (ii) a simple root. From Theorem 4.80, we know that all eigenvalues of $\mathbf{M}$ are real. But for (i) and (ii) to hold, no other (real) eigenvalue can have value equal to λ_0 (otherwise it would not be a simple root). Thus, λ_0 is the unique largest eigenvalue of $\mathbf{M}$. This completes the proof. $\square$

CHAPTER 5

Applications of Algebraic Graph Theory: Eigenvector Centrality and Page-Rank

Remark 5.1. In this chapter, we're going to explore two applications of Algebraic Graph Theory: Eigenvector Centrality and Page-Rank. The goal is to devise ways for ranking the vertices of a graph. This topic is actually very important. Google uses Page-Rank to rank the search results they return. Social scientists have used eigenvector centrality as a way of determining leaders in organizations. We'll first review a key set of definitions from Linear Algebra and then discuss eigenvector centrality. We then move on to Markov chains and Page-Rank.

1. Basis of $\mathbb{R}^n$

Remark 5.2. We will be considering the field $\mathbb{R}$ and *vectors* defined over it. By this, we mean $n \times 1$ matrices, which are just column vectors. Therefore, by a vector in $\mathbb{R}^n$ we really mean a matrix $\mathbf{x} \in \mathbb{R}^{n\times 1}$.

Definition 5.3. Let $\mathbf{x}_1, \dots, \mathbf{x}_m$ be vectors in $\in \mathbb{R}^n$ and let $\alpha_1, \dots, \alpha_m \in \mathbb{R}$ be scalars. Then

$$\alpha_1\mathbf{x}_1 + \cdots + \alpha_m\mathbf{x}_m \tag{5.1}$$

is a *linear combination* of the vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$.

Definition 5.4 (Span). Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ be a set of vectors in $\in \mathbb{R}^n$, then the span of $\mathcal{X}$ is the set:

$$\text{span}(\mathcal{X}) = \{\mathbf{y} \in \mathbb{R}^n | \mathbf{y} \text{ is a linear combination of vectors in } \mathcal{X}\} \tag{5.2}$$

Definition 5.5 (Linear Independence). Let $\mathbf{x}_1, \dots, \mathbf{x}_m$ be vectors in $\in \mathbb{R}^n$. The vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$ are *linearly dependent* if there exists $\alpha_1, \dots, \alpha_m \in \mathbb{R}$, not all zero, such that

$$\alpha_1\mathbf{x}_1 + \cdots + \alpha_m\mathbf{x}_m = \mathbf{0} \tag{5.3}$$

If the set of vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$ is not linearly dependent, then they are *linearly independent* and Equation 5.3 holds just in case $\alpha_i = 0$ for all $i = 1, \dots, n$.

Example 5.6. In $\mathbb{R}^3$, consider the vectors:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix},\ \mathbf{x}_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},\ \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

We can show these vectors are linearly independent: Suppose there are values $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ such that

$$\alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \alpha_3\mathbf{x}_3 = 0$$

Then:

$$\begin{bmatrix}\alpha_1\\\alpha_1\\0\end{bmatrix} + \begin{bmatrix}\alpha_2\\0\\\alpha_2\end{bmatrix} \begin{bmatrix}0\\\alpha_3\\\alpha_3\end{bmatrix} = \begin{bmatrix}\alpha_1+\alpha_2\\\alpha_1+\alpha_3\\\alpha_2+\alpha_3\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

Thus we have the system of linear equations:

$$\begin{aligned}\alpha_1 \quad +\alpha_2 \qquad &= 0\\ \alpha_1 \qquad\quad + \alpha_3 &= 0\\ \alpha_2 + \alpha_3 &= 0\end{aligned}$$

Solving this problem yields the unique solution: $\alpha_1 = \alpha_2 = \alpha_3 = 0$. Thus these vectors are linearly independent.

Definition 5.7 (Basis). Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ be a set of vectors in $\mathbb{R}^n$. The set $\mathcal{X}$ is called a *basis* of $\mathbb{R}^n$ if $\mathcal{X}$ is a linearly independent set of vectors and every vector in $\mathbb{R}^n$ is in the span of $\mathcal{X}$. That is, for any vector $\mathbf{w} \in \mathbb{R}^n$ we can find scalar values $\alpha_1, \dots, \alpha_m$ such that

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \tag{5.4}$$

Example 5.8. We can show that the vectors:

$$\mathbf{x}_1 = \begin{bmatrix}1\\1\\0\end{bmatrix},\ \mathbf{x}_2 = \begin{bmatrix}1\\0\\1\end{bmatrix},\ \mathbf{x}_3 = \begin{bmatrix}0\\1\\1\end{bmatrix}$$

form a basis of $\mathbb{R}^3$. We already know that the vectors are linearly independent. To show that $\mathbb{R}^3$ is in their span, chose an arbitrary vector in $\mathbb{R}^m$: $[a, b, c]^T$. Then we hope to find coefficients α_1, α_2 and α_3 so that:

$$\alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \alpha_3\mathbf{x}_3 = \begin{bmatrix}a\\b\\c\end{bmatrix}$$

Expanding this, we must find α_1, α_2 and α_3 so that:

$$\begin{bmatrix}\alpha_1\\\alpha_1\\0\end{bmatrix} + \begin{bmatrix}\alpha_2\\0\\\alpha_2\end{bmatrix} + \begin{bmatrix}0\\\alpha_3\\\alpha_3\end{bmatrix} = \begin{bmatrix}a\\b\\c\end{bmatrix}$$

This problem can be solved in terms of a, b, and c to yield:

$$\begin{aligned}\alpha_1 &= 1/2\,a + 1/2\,b - 1/2\,c\\ \alpha_2 &= -1/2\,b + 1/2\,a + 1/2\,c\\ \alpha_3 &= 1/2\,c + 1/2\,b - 1/2\,a\end{aligned}$$

which clearly has a solution for all a, b, and c.

Remark 5.9. The following theorem on the size of a basis in $\mathbb{R}^n$ is outside the scope of this course. A proof can be found in [Lan87].

Theorem 5.10. *If $\mathcal{X}$ is a basis of $\mathbb{R}^n$, then $\mathcal{X}$ contains precisely n vectors.*

2. Eigenvector Centrality

Remark 5.11. This approach to justifying eigenvector centrality comes from Leo Spizzirri [Spi11]. It is reasonably nice, and fairly rigorous. This is not meant to be anymore than a justification. It is not a proof of correctness. Before proceeding, we require a theorem, whose proof is outside the scope of the course. (See [GR01], Chapter 8, Page 170.)

Theorem 5.12 (Principle Axis Theorem). *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be a symmetric matrix. Then* $\mathbb{R}^n$ *has a basis consisting of eigenvectors of* $\mathbf{M}$. □

Remark 5.13 (Eigenvector Centrality). We can assign to each vertex of a graph $G = (V, E)$ a score (called its eigenvector centrality) that will determine its *relative importance* in the graph. Here importance it measured in a self-referential way: important vertices are important precisely because they are adjacent to other important vertices. This self-referential definition can be resolved in the following way.

Let x_i be the (unknown) score of vertex $v_i \in V$ and let $x_i = \kappa(v_i)$ with κ being the function returning the score of each vertex in V. We may define x_i as a pseudo-average of the scores of its neighbors. That is, we may write:

$$x_i = \frac{1}{\lambda}\sum_{v\in N(v_i)} \kappa(v) \tag{5.5}$$

Here λ will be chosen endogenously during computation.

Recall that $\mathbf{M}_{i\cdot}$ is the i^{th} row of the adjacency matrix $\mathbf{M}$ and contains a 1 in position j if and only if v_i is adjacent to v_j; that is to say $v_j \in N(v_i)$. Thus we can rewrite Equation 5.5 as:

$$x_i = \frac{1}{\lambda}\sum_{j=1}^{n} \mathbf{M}_{ij}x_j$$

This leads to n equations, one for vertex in V (or each row of $\mathbf{M}$). Written as a matrix expression we have:

$$\mathbf{x} = \frac{1}{\lambda}\mathbf{M}\mathbf{x} \implies \lambda\mathbf{x} = \mathbf{M}\mathbf{x} \tag{5.6}$$

Thus $\mathbf{x}$ is an eigenvector of $\mathbf{M}$ and λ is its eigenvalue.

Clearly, there may be several eigenvectors and eigenvalues for $\mathbf{M}$. The question is, which eigenvalue / eigenvector pair should be chosen? The answer is to choose the eigenvector with all positive entries corresponding to the largest eigenvalue. We know such an eigenvalue / eigenvector pair exists and is unique as a result of the Perron-Frobenius Theorem and Lemma 4.83.

Theorem 5.14. *Let* $G = (V, E)$ *be a connected graph with adjacency matrix* $\mathbf{M} \in \mathbb{R}^{n\times n}$. *Suppose that* λ_0 *is the largest real eigenvalue of* $\mathbf{M}$ *and has corresponding eigenvalue* $\mathbf{v}_0$. *If* $\mathbf{x} \in \mathbb{R}^{n\times 1}$ *is a column vector so that* $\mathbf{x}\cdot\mathbf{v_0} \neq 0$, *then*

$$\lim_{k\to\infty} \frac{\mathbf{M}^k\mathbf{x}}{\lambda_0^k} = \alpha_0\mathbf{v_0} \tag{5.7}$$

Proof. Applying Theorem 5.12 we see that the eigenvectors of $\mathbf{M}$ must form a basis for $\mathbb{R}^n$. Thus, we can express:

$$\mathbf{x} = \alpha_0\mathbf{v}_0 + \alpha_1\mathbf{v}_1 + \cdots + \alpha_n\mathbf{v}_n \tag{5.8}$$

Multiplying both sides by $\mathbf{M}^k$ yields:

$$\mathbf{M}^k\mathbf{x} = \alpha_0\mathbf{M}^k\mathbf{v}_0 + \alpha_1\mathbf{M}^k\mathbf{v}_1 + \cdots + \alpha_n\mathbf{M}^k\mathbf{v}_n = \alpha_0\lambda_0^k\mathbf{v}_0 + \alpha_1\lambda_1^k\mathbf{v}_1 + \cdots + \alpha_n\lambda_n^k\mathbf{v}_n \tag{5.9}$$

because $\mathbf{M}^k\mathbf{v}_i = \lambda_i^k\mathbf{v}_i$ for any eigenvalue $\mathbf{v}_i$. Dividing by λ_0^k yields:

$$\frac{\mathbf{M}^k\mathbf{x}}{\lambda_0^k} = \alpha_0\mathbf{v}_0 + \alpha_1\frac{\lambda_1^k}{\lambda_0^k}\mathbf{v}_1 + \cdots + \alpha_n\frac{\lambda_n^k}{\lambda_0^k}\mathbf{v}_n \tag{5.10}$$

Applying the Perron-Frobenius Theorem (and Lemma 4.83) we see that λ_0 is greater than the absolute value of any other eigenvalue and thus we have:

$$\lim_{k\to\infty}\frac{\lambda_i^k}{\lambda_0^k} = 0 \tag{5.11}$$

for $i \neq 0$. Thus:

$$\lim_{k\to\infty}\frac{\mathbf{M}^k\mathbf{x}}{\lambda_0^k} = \alpha_0\mathbf{v_0} \tag{5.12}$$

□

Remark 5.15. We can use Theorem 5.14 to justify our definition of eigenvector centrality as the eigenvector corresponding to the largest eigenvalue. Let $\mathbf{x}$ be a vector with a 1 at index i and 0 everywhere else. This vector corresponds to beginning at vertex v_i in graph G with n vertices. If $\mathbf{M}$ is the adjacency matrix, then $\mathbf{Mx}$ is the i^{th} column of $\mathbf{M}$ whose j^{th} index tells us the number of walks of length 1 leading from vertex v_j to vertex v_i and *by symmetry* the number of walks leading from vertex v_i to vertex v_j. We can repeat this logic to see that $\mathbf{M}^k\mathbf{x}$ gives us a vector of whose j^{th} element is the number of walks of length k from v_i to v_j. Note for the remainder of this discussion, we will exploit the symmetry that the (i, j) element of M^k is both the number of walks from i to j and the number of walks from j *to* i.

From Theorem 5.14 we know that no matter which vertex we choose in creating $\mathbf{x}$ that:

$$\lim_{k\to\infty}\frac{\mathbf{M}^k\mathbf{x}}{\lambda_0} = \alpha_0\mathbf{v_0} \tag{5.13}$$

Reinterpreting Equation 5.13 we observe that as $k \to \infty$, $\mathbf{M}^k\mathbf{x}$ will converge to some multiple of the eigenvector corresponding to the eigenvalue λ_0. That is, the eigenvector corresponding to the largest eigenvalue is a multiple of the number of walks of length k leading from some initial vertex i, since the Perron-Frobeinus eigenvector is unique (up to a scale).

Now, suppose that we were going to choose one of these (many) walks of length k at random, we might ask what is the probability of arriving at vertex j given that we started at vertex i. If we let $\mathbf{y} = \mathbf{M}^k\mathbf{x}$ then the total number of walks leading from i is:

$$T = \sum_{j=1}^{n}\mathbf{y}_j \tag{5.14}$$

The probability of ending up at vertex j then is the j^{th} element of the vector $\frac{1}{T}\mathbf{M}^k\mathbf{x}$. Put more intuitively, if we start at a vertex and begin wandering along the edges of the graph at random, the probability of ending up at vertex j after k steps is the j^{th} element of $\frac{1}{T}\mathbf{M}^k\mathbf{x}$.

If we let: $|\mathbf{x}|_1$ be the sum of the components of a vector $\mathbf{x}$, then from Equation 5.13 we may deduce:

$$\lim_{k\to\infty}\frac{\mathbf{M}^k\mathbf{x}}{|\mathbf{M}^k\mathbf{x}|_1} = \frac{\mathbf{v_0}}{|\mathbf{v}_0|_1} \tag{5.15}$$

Thus, eigenvector centrality tells us a kind of probability of ending up at a specific vertex j assuming we are allowed to wander along the edges of a graph (from any starting vertex) for an infinite amount of time. Thus, the more central a vertex is, the more likely we will arrive at it as we move through the edges of the graph. More importantly, we see the self-referential nature of eigenvector centrality: the more likely we are to arrive at a given vertex after walking along the edges of a graph, the more likely we are to arrive at one of its neighbors. Thus, eigenvector centrality is a legitimate measure of vertex importance if one wishes to measure the chances of ending up at a vertex when wondering around a graph. We will discuss this type of model further when we investigate random walks on graph.

Example 5.16. Consider the graph shown in Figure 5.1. Recall from Example 4.58 this

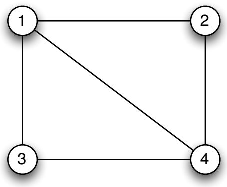


Figure 5.1. A matrix with 4 vertices and 5 edges. Intuitively, vertices 1 and 4 should have the same eigenvector centrality score as vertices 2 and 3.

graph had adjacency matrix:

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

We can use a computer to determine the eigenvalues and eigenvectors of $\mathbf{M}$. The eigenvalues are:

$$\left\{0, -1, \frac{1}{2}+\frac{1}{2}\sqrt{17}, \frac{1}{2}-\frac{1}{2}\sqrt{17},\right\}$$

while the corresponding floating point approximations of the eigenvalues the columns of the matrix:

$$\begin{bmatrix} 0.0 & -1.0 & 1.0 & 1.000000001 \\ -1.0 & 0.0 & 0.7807764064 & -1.280776407 \\ 1.0 & 0.0 & 0.7807764069 & -1.280776408 \\ 0.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

The largest eigenvalue is $\lambda_0 = \frac{1}{2} + \frac{1}{2}\sqrt{17}$ which has corresponding eigenvector:

$$\mathbf{v}_0 = \begin{bmatrix} 1.0 \\ 0.7807764064 \\ 0.7807764064 \\ 1.0 \end{bmatrix}$$

We can normalize this vector to be:

$$\mathbf{v}_0 = \begin{bmatrix} 0.2807764065 \\ 0.2192235937 \\ 0.2192235937 \\ 0.2807764065 \end{bmatrix}$$

Illustrating that vertices 1 and 4 have identical (larger) eigenvector centrality scores and vertices 2 and 3 have identical (smaller) eigenvector centrality scores. By way of comparison, consider the vector:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We consider $\mathbf{M}^k\mathbf{x}/|\mathbf{M}^k\mathbf{x}|_1$ for various values of k:

$$\frac{\mathbf{M}^1\mathbf{x}}{|\mathbf{M}^1\mathbf{x}|_1} = \begin{bmatrix} 0.0 \\ 0.3333333333 \\ 0.3333333333 \\ 0.3333333333 \end{bmatrix} \quad \frac{\mathbf{M}^{10}\mathbf{x}}{|\mathbf{M}^{10}\mathbf{x}|_1} = \begin{bmatrix} 0.2822190823 \\ 0.2178181007 \\ 0.2178181007 \\ 0.2821447163 \end{bmatrix}$$

$$\frac{\mathbf{M}^{20}\mathbf{x}}{|\mathbf{M}^{20}\mathbf{x}|_1} = \begin{bmatrix} 0.2807863651 \\ 0.2192136380 \\ 0.2192136380 \\ 0.2807863590 \end{bmatrix} \quad \frac{\mathbf{M}^{40}\mathbf{x}}{|\mathbf{M}^{40}\mathbf{x}|_1} = \begin{bmatrix} 0.2807764069 \\ 0.2192235931 \\ 0.2192235931 \\ 0.2807764069 \end{bmatrix}$$

It's easy to see that as $k \to \infty$, $\mathbf{M}^k\mathbf{x}/|\mathbf{M}^k\mathbf{x}|_1$ approaches the normalized eigenvector centrality scores as we expected.

3. Markov Chains and Random Walks

Remark 5.17. Markov Chains are a type of directed graph in which we assign to each edge a probability of walking along that edge given we imagine ourselves standing in a specific vertex adjacent to the edge. Our goal is to define Markov chains, and random walks on a graph in reference to a Markov chain and show that some of the properties of graphs can be used to derive interesting properties of Markov chains. We'll then discuss another way of ranking vertices; this one is used (more-or-less) by Google for ranking webpages in their search.

Definition 5.18 (Markov Chain). A *discrete time Markov Chain* is a tuple $\mathcal{M} = (G, p)$ where $G = (V, E)$ is a *directed* graph and the set of vertices is usually referred to as the *states*, the set of edges are called the *transitions* and $p : E \to [0, 1]$ is a probability assignment function satisfying:

$$\sum_{v' \in N_o(v)} p(v, v') = 1 \tag{5.16}$$

for all $v \in V$. Here, $N_o(v)$ is the neighborhood reachable by out-edge from v. If there is no edge $(v, v') \in E$ then $p(v, v') = 0$.

Remark 5.19. There are continuous time Markov chains, but these are not in the scope of these notes. When we say Markov chain, we mean discrete time Markov chain.

Example 5.20. A simple Markov chain is shown in Figure 5.2. We can think of a Markov chain as governing the evolution of state as follows. Think of the states as cities with airports. If there is an out-edge connecting the current city to another city, then we can fly from our current city to this next city and we do so with some probability. When we do fly (or perhaps don't fly and remain at the current location) our state updates to the next city. In this case, time is treated *discretely.*

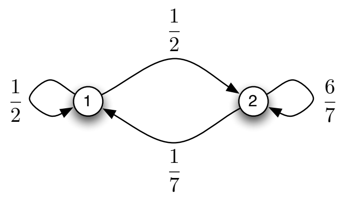


Figure 5.2. A Markov chain is a directed graph to which we assign edge probabilities so that the sum of the probabilities of the out-edges at any vertex is always 1.

A walk along the vertices of a Markov chain governed by the probability function is called a *random walk.*

Definition 5.21 (Stochastic Matrix). Let $\mathcal{M} = (G, p)$ be a Markov chain. Then the *stochastic matrix* (or probability transition matrix) of $\mathcal{M}$ is:

$$\mathbf{M}_{ij} = p(v_i, v_j) \tag{5.17}$$

Example 5.22. The stochastic matrix for the Markov chain in Figure 5.2 is:

$$\mathbf{M} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{7} & \frac{6}{7} \end{bmatrix}$$

Thus a stochastic matrix is very much like an adjacency matrix where the 0's and 1's indicating the presence or absence of an edge are replaced by the probabilities associated to the edges in the Markov chain.

Definition 5.23 (State Probability Vector). If $\mathcal{M} = (G, p)$ be a Markov chain with n states (vertices) then a *state probability vector* is a vector $\mathbf{x} \in \mathbb{R}^{n \times 1}$ such that $x_1 + x_2 + \cdots + x_n = 1$ and $x_i \geq 0$ for $i = 1, \dots, n$ and x_i represents the probability that we are in state i (at vertex i).

REMARK 5.24. The next theorem can be proved in exactly the same way that Theorem 4.57 is proved.

THEOREM 5.25. *Let $\mathcal{M} = (G, p)$ be a Markov chain with n states (vertices). Let $\mathbf{x}^{(0)} \in \mathbb{R}^{n\times 1}$ be an (initial) state probability vector. Then assuming we take a random walk of length k in $\mathcal{M}$ using initial state probability vector $\mathbf{x}^{(0)}$, the final state probability vector is:*

$$\mathbf{x}^{(k)} = \left(\mathbf{M}^T\right)^k \mathbf{x}^{(0)} \tag{5.18}$$

□

EXERCISE 47. Prove Theorem 5.25. [Hint: Use the same inductive argument from the proof of Theorem 4.57.]

EXAMPLE 5.26. Consider the Markov chain in Figure 5.2. The state vector:

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

states that we will start in State 1 with probability 1. From Example 5.22 we know what $\mathbf{M}$ is. Then it is easy to see that:

$$\mathbf{x}^{(1)} = \left(\mathbf{M}^T\right)^k \mathbf{x}^{(0)} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Which is precisely the state probability vector we would expect after a random walk of length 1 in $\mathcal{M}$.

DEFINITION 5.27 (Stationary Vector). Let $\mathcal{M} = (G, p)$ be a Markov chain. Then a vector $\mathbf{x}^*$ is stationary for $\mathcal{M}$ if

$$\mathbf{x}^* = \mathbf{M}^T\mathbf{x}^* \tag{5.19}$$

REMARK 5.28. Expression 5.19 should look familiar. It says that $\mathbf{M}^T$ has an eigenvalue of 1 and a corresponding eigenvector whose entries are all non-negative (so that the vector can be scaled so its components sum to 1). Furthermore, this looks very similar to the equation we used for eigenvector centrality.

LEMMA 5.29. *Let $\mathcal{M} = (G, p)$ be a Markov chain with n states and with stochastic matrix $\mathcal{M}$. Then:*

$$\sum_j \mathbf{M}_{ij} = 1 \tag{5.20}$$

for all $i = 1, \ldots, n$.

EXERCISE 48. Prove Lemma 5.29.

LEMMA 5.30. *$\mathcal{M} = (G, p)$ be a Markov chain with n states and with stochastic matrix $\mathbf{M}$. If G is strongly connected, then $\mathbf{M}$ and $\mathbf{M}^T$ are irreducible.*

PROOF. If G is strongly connected, then there is a directed walk from any vertex v_i to any other vertex v_j in V, the vertex set of G. Consider any length k walk connecting v_i to v_j (such a walk exists for some k). Let $\mathbf{e}_i$ be the vector with 1 in its i^{th} component and 0 everywhere else. Then $(\mathbf{M}^T)^k\mathbf{e}_i$ is the final state probability vector associated with a walk of length k starting at vertex v_i. Since there is a walk of length k from v_i to v_j, we know that the j^{th} element of this vector must be non-zero. That is:

$$\mathbf{e}_j^T(\mathbf{M}^T)^k\mathbf{e}_i > 0$$

where $\mathbf{e}_j$ is defined just as $\mathbf{e}_i$ is but with the 1 at the j^{th} position. Thus, $(\mathbf{M}^T)^k_{ij} > 0$ for some k for every (i, j) pair and thus $\mathbf{M}^T$ is irreducible. The fact that M is irreducible follows immediately from the fact that $(M^T)^k = (M^k)^T$. This completes the proof. □

Theorem 5.31 (Perron-Frobenius Theorem Redux). *If $\mathbf{M}$ is an irreducible matrix, then $\mathbf{M}$ has an eigenvalue λ_0 with the following properties:*

1. *The eigenvalue λ_0 is positive and if λ is an alternative eigenvalue of $\mathbf{M}$, then $\lambda_0 \geq |\lambda|$,*
2. *The matrix $\mathbf{M}$ has an eigenvectors $\mathbf{v}_0$ corresponding to λ_0 with only positive entries,*
3. *The eigenvalue λ is a simple root of the characteristic equation for $\mathbf{M}$ and therefore has a unique (up to scale) eigenvectors $\mathbf{v}_0$.*
4. *The eigenvector $\mathbf{v}_0$ is the only eigenvector of $\mathbf{M}$ that can have all positive entries when properly scaled.*
5. *The following inequalities hold:*

$$\min_i \sum_j \mathbf{M}_{ij} \leq \lambda_0 \leq \max_i \sum_j \mathbf{M}_{ij}$$

Theorem 5.32. *Let $\mathcal{M} = (G, p)$ be a Markov chain with stochastic matrix $\mathbf{M}$, $\mathbf{M}^T$, is irreducible then $\mathcal{M}$ has a unique stationary probability distribution.*

Proof. From Theorem 4.71 we know that $\mathbf{M}$ and $\mathbf{M}^T$ have identical eigenvalues. By the Perron-Frobenius theorem, $\mathbf{M}$ has a largest positive eigenvalue λ_0 that satisfies:

$$\min_i \sum_j \mathbf{M}_{ij} \leq \lambda_0 \leq \max_i \sum_j \mathbf{M}_{ij}$$

By Lemma 5.29, we know that:

$$\min_i \sum_j \mathbf{M}_{ij} = \max_i \sum_j \mathbf{M}_{ij} = 1$$

Therefore, by the squeezing lemma $\lambda_0 = 1$. The fact that $\mathbf{M}^T$ has exactly one strictly positive eigenvector $\mathbf{v}_0$ corresponding to $\lambda_0 = 1$ means that:

$$\mathbf{M}^T\mathbf{v}_0 = \mathbf{v}_0 \tag{5.21}$$

Thus $\mathbf{v}_0$ is the unique stationary state probability vector for $\mathcal{M} = (G, p)$. This completes the proof. □

4. Page Rank

Definition 5.33 (Induced Markov Chain). Let $G = (V, E)$ be a graph. Then the *induced Markov chain* from G is the one obtained by defining a new directed graph $G' = (V, E')$ with each edge $\{v, v'\} \in E$ replaced by two directional edges (v, v') and (v', v) in E and defining the probability function p so that:

$$(5.22) \quad p(v, v') = \frac{1}{\deg_{out_{G'}} v}$$

Example 5.34. An induced Markov chain is shown in Figure 5.3. The Markov chain in

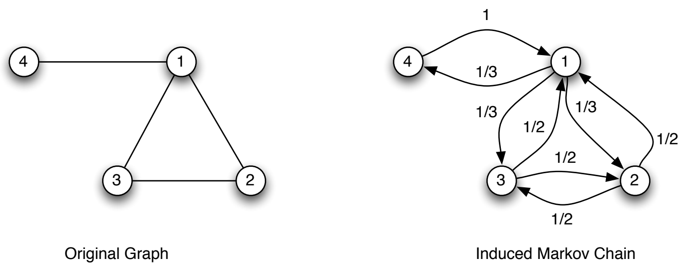


Figure 5.3. An induced Markov chain is constructed from a graph by replacing every edge with a pair of directed edges (going in opposite directions) and assigning a probability equal to the out-degree of each vertex to every edge leaving that vertex.

the figure has the stationary state probability vector:

$$\mathbf{x}^* = \begin{bmatrix} \frac{3}{8} \\ \frac{2}{8} \\ \frac{2}{8} \\ \frac{1}{8} \end{bmatrix}$$

which is the eigenvector corresponding to the eigenvalue 1 in the matrix $\mathbf{M}^T$. Arguing as we did in the proof of Theorem 5.14 and Example 5.16, we could expect that for any state vector $\mathbf{x}$ we would have:

$$\lim_{k \to \infty} \left(\mathbf{M}^T\right)^k \mathbf{x} = \mathbf{x}^*$$

and we would be correct. When this convergence happens *quickly* (where we leave quickly poorly defined) the graph is said to have a *fast mixing* property.

If we used the stationary probability of a vertex in the induced Markov chain as a measure of importance, then clearly vertex 1 would be most important followed by vertices 2 and 3 and lastly vertex 4. We can compare this with the eigenvector centrality measure, which assigns a rank vector of:

$$\mathbf{x}^+ = \begin{bmatrix} 0.3154488065 \\ 0.2695944375 \\ 0.2695944375 \\ 0.1453623195 \end{bmatrix}$$

Thus eigenvector centrality gives the same ordinal ranking as using the stationary state probability vector, but there are subtle differences in the values produced by these two ranking schemes. This leads us to PageRank [**BP98**].

Remark 5.35. Consider a collection of web pages each with links. We can construct a directed graph G with the vertex set V consisting of the we web pages and E consisting of the directed links among the pages. Imagine a random web surfer who will click among these web pages following links until a dead-end is reached (a page with no outbound links). In this case, the web surfer will type a new URL in (chosen from the set of web pages available) and the process will continue.

From this model, we can induce a Markov chain in which we define a new graph G' with edge set E' so that if $v \in V$ has out-degree 0, then we create an edge in E' to every other vertex in V and we then define:

$$p(v, v') = \frac{1}{\deg_{out_{G'}} v} \tag{5.23}$$

exactly as before. In the absence of *any further* insight, the PageRank algorithm simply assigns to each web page a score equal to the stationary probability of its state in the induced Markov chain. For the remainder of this remark, let $\mathbf{M}$ be the stochastic matrix of the induced Markov chain.

In general, however, PageRank assumes that surfers will get bored after some number of clicks (or new URL's) and will stop (and move to a new page) with some probability $d \in [0, 1]$ called the damping factor. This factor is usually estimated. Assuming there are n web pages, let $\mathbf{r} \in \mathbb{R}^{n\times 1}$ be the PageRank score for each page. Taking boredom into account leads to a new expression for rank (similar to Equation 5.5 for Eigenvector centrality):

$$r_i = \frac{1-d}{n} + d\left(\sum_{j=1}^{n} M_{ji} r_j\right) \quad \text{for } i = 1, \dots, n \tag{5.24}$$

Here the d term acts like a damping factor on walks through the Markov chain. In essence, it stalls people as they walk, making it less likely a searcher will keep walking forever. The original System of Equations 5.24 can be written in matrix form as:

$$\mathbf{r} = \left(\frac{1-d}{n}\right)\mathbf{1} + d\mathbf{M}^T\mathbf{r} \tag{5.25}$$

where $\mathbf{1}$ is a $n \times 1$ vector consisting of all 1's. It is easy to see that when $d = 1$ $\mathbf{r}$ is precisely the stationary state probability vector for the induced Markov chain. When $d \neq 1$, $\mathbf{r}$ is usually computed iteratively by starting with an initial value of $r_i^0 = 1/n$ for all $i = 1, \dots, n$ and computing:

$$\mathbf{r}^{(k)} = \left(\frac{1-d}{n}\right)\mathbf{1} + d\mathbf{M}^T\mathbf{r}^{(k-1)}$$

The reason is that for large n, the analytic solution:

$$\mathbf{r} = \left(\mathbf{I}_n - d\mathbf{M}^T\right)^{-1}\left(\frac{1-d}{n}\right)\mathbf{1} \tag{5.26}$$

is not computationally tractable[1].

[1]Note, $\left(\mathbf{I}_n - d\mathbf{M}^T\right)^{-1}$ computes a matrix inverse, which we reviewed *briefly* in Chapter 4. We should note that for stochastic matrices, this inverse is guaranteed to exist. For those interested, please consult and of [**Dat95**, **Lan87**, **Mey01**].

Example 5.36. Consider the induced Markov chain in Figure 5.3 and suppose we wish to compute PageRank on these vertices with $d = 0.85$ (which is a common assumption). We might begin with:

$$\mathbf{r}^{(0)} = \begin{bmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{bmatrix}$$

We would then compute:

$$\mathbf{r}^{(1)} = \left(\frac{1-d}{n}\right)\mathbf{1} + d\mathbf{M}^T\mathbf{r}^{(0)} = \begin{bmatrix} 0.462499999999999967 \\ 0.214583333333333320 \\ 0.214583333333333320 \\ 0.108333333333333337 \end{bmatrix}$$

We would repeat this again to obtain:

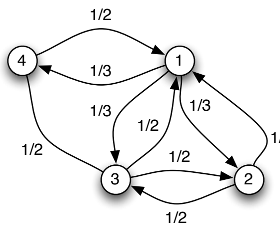
$$\mathbf{r}^{(2)} = \left(\frac{1-d}{n}\right)\mathbf{1} + d\mathbf{M}^T\mathbf{r}^{(1)} = \begin{bmatrix} 0.311979166666666641 \\ 0.259739583333333302 \\ 0.259739583333333302 \\ 0.168541666666666673 \end{bmatrix}$$

This would continue until the difference between in the values of $\mathbf{r}^{(k)}$ and $r^{(k-1)}$ was small. The final solution would be close to the exact solution:

$$\mathbf{r}^* = \begin{bmatrix} 0.366735867135100591 \\ 0.245927818588310476 \\ 0.245927818588310393 \\ 0.141408495688278513 \end{bmatrix}$$

Note this is (again) very close to the stationary probabilities and the eigenvector centralities we observed earlier. This vector is normalized so that all the entries sum to 1.

Exercise 49. Consider the Markov chain shown below:



Suppose this is the induced Markov chain from 4 web pages. Compute the page-rank of these web pages using $d = 0.85$.

Exercise 50. Find an expression for $\mathbf{r}^{(2)}$ in terms of $\mathbf{r}^{(0)}$. Explain how the damping factor occurs and how it decreases the chance of taking long walks through the induced Markov chain. Can you generalize your expression for $\mathbf{r}^{(2)}$ to an expression for $\mathbf{r}^{(k)}$ in terms of $\mathbf{r}^{(0)}$?

CHAPTER 6

Trees, Algorithms and Matroids

1. Two Tree Search Algorithms

Remark 6.1. Tree searching is the process of enumerating the vertices of a tree (for the purpose of "searching" them). One can consider this process as generating a walk hitting all the vertices of the tree at least once or as a way to create a sequence of the vertices. In this section, we will take the latter view, though it will be clear how to create a walk as well.

The first algorithm, called Breadth First Search, explores the vertices of a tree starting from a given vertex by exploring all the neighbors of this given vertex, then the neighbors of the neighbors and so on until all the vertices have been encountered.

```
Breadth First Search
Input: $T = (V, E)$ a tree, $v_0$ a starting vertex
Initialize: $F = (v_0)$ {A sequence of vertices to enumerate.}
Initialize: $F_{\text{next}} = ()$ {The sequence of next vertices to enumerate.}
Initialize: $w = ()$ {The sequence of vertices traversed.}
    (1) while $F \neq \emptyset$
    (2)     for each $v \in F$ do
    (3)         Remove $v$ from $F$
    (4)         Append $v$ to $w$
    (5)         for each $v' \in V$ do
    (6)             if $v' \notin w$ and $\{v, v'\} \in E$ then
    (7)                 Append $v'$ to $F_{\text{next}}$
    (8)             end if
    (9)         end for
   (10)     end for
   (11)     $F = F_{\text{next}}$
   (12)     $F_{\text{next}} = ()$
   (13) end while
Output: $w$ {A breadth first sequence of vertices in $T$.}
```

Algorithm 1. Breadth First Search

Example 6.2. Figure 6.1 shows the order the vertices are added to w during a breadth first search of the tree.

Proposition 6.3. *A breadth first search of a tree $T = (V, E)$ enumerates all vertices in w.*

Proof. We proceed by induction. If T has one vertex, then clearly v_0 in the algorithm is that vertex. The vertex is added to w in the first iteration of the while loop at Line 1 and

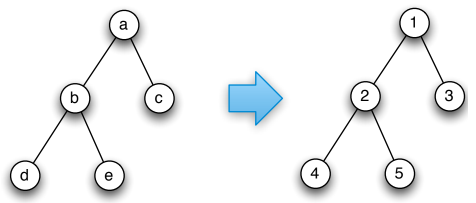


Figure 6.1. The breadth first walk of a tree explores the tree in an ever widening pattern.

F_{next} is the empty set, thus the algorithm terminates. Now suppose that the statement is true for all trees with at most n vertices. We will show the statement is true for a tree with $n+1$ vertices. To see this, construct a new tree T' in which we remove a leaf vertex v' from T. Clearly the algorithm must enumerate every vertex in T' and therefore there is a point in which we reach Line 3 with some vertex v that is adjacent to v' in T. At this point, v' would be added to F_{next} and it would be added to w in the next execution through the while loop since $F \neq \emptyset$ the next time. Thus, every vertex of T must be enumerated in w. This completes the proof. □

Remark 6.4. Breadth First Search can be modified for directed trees in the obvious way. Necessarily, we need v_0 to be strongly connected to every other vertex in order to ensure that BFS enumerates every possible vertex.

Remark 6.5. Another algorithm for enumerating the vertices of a tree is the depth first search algorithm. This algorithm works by descending into the tree as deeply as possible (until a leaf is identified) and then working back up. We present Depth First Search as a *recursive* algorithm.

```
Depth First Search
Input: $T = (V, E)$ a tree, $v_0$ a starting vertex
Initialize: $v_{\text{now}} = v_0$ {The current vertex.}
Initialize: $w = (v_0)$ {The sequence of next vertices to enumerate.}
Initialize: $v_{\text{last}} = \emptyset$ {The sequence of vertices traversed.}
    (1) Recurse($T, v_{\text{now}}, v_{\text{last}}, w$)
Output: $w$ {A breadth first sequence of vertices in $T$.}

Recurse
Input: $T = (V, E)$ a tree, $v_{\text{now}}$ current vertex, $v_{\text{last}}$ the last vertex, $w$ the sequence
    (1) for each $v \in V$ do
    (2)     if $\{v_{\text{now}}, v\} \in E$ and $v \notin w$ then
    (3)         Append $v$ to $w$
    (4)         Recurse($T, v, v_{\text{now}}, w$)
    (5)     end if
    (6) end for
```

Algorithm 2. Depth First Search

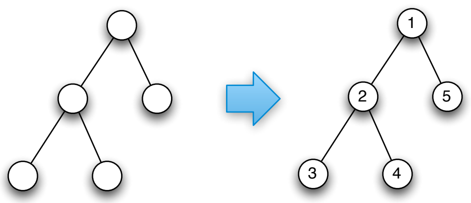


Figure 6.2. The depth first walk of a tree explores the tree in an ever deepening pattern.

Example 6.6. Figure 6.2 shows the order the vertices are added to w during a depth first search of the tree.

Proposition 6.7. *A depth first search of a tree $T = (V, E)$ enumerates all vertices in w.*

Exercise 51. Prove proposition 6.7. [Hint: The proof is almost identical to the proof for Breadth First Search.]

Remark 6.8. We note that breadth and depth first search can be trivially modified to search through connected graph structures and construct spanning trees for these graphs. We also note that BFS and DFS can be modified to function on directed trees (and graphs) and that all vertices will be enumerated provided that every vertex is reachable by a directed path from v_0.

Remark 6.9. In terms of implementation, we note that the recursive implementation of Depth First Search works on most computing systems provided the graph your are searching has a longest path of at most some specified value. This is because most operating systems prevent a recursive algorithm from making any more than a specified number of recursion calls.

Exercise 52. Rewrite the breadth first search algorithm to produce a spanning tree of a connected graph.

2. Prim's Spanning Tree Algorithm

Definition 6.10 (Weighted Graph). A *weighted graph* is a pair (G, w) where $G = (V, E)$ is a graph and $w : E \to \mathbb{R}$ is a weight function.

Remark 6.11. A weighted digraph is defined analagously. A Markov chain is an example of a weighted graph where the probabilities play the role of the edge weights.

Example 6.12. Consider the graph shown in Figure 6.3. A weighted graph is simply a graph with a real number (the weight) assigned to each edge. Weighted graphs arise in several instances, such as travel planning, communications and military planning.

Remark 6.13. Any graph can be thought of as a weighted graph in which we assign the weight of 1 to each edge. The distance between two vertices in a graph can then easily be generalized in a weighted graph. If $p = (v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1})$ is a path, then the weight

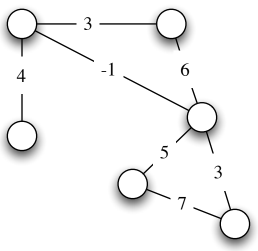


Figure 6.3. A weighted graph is simply a graph with a real number (the weight) assigned to each edge.

of the path is:

$$\sum_{i=1}^{n} w(e_i)$$

Thus in a weighted graph, the distance between two vertices v_1 and v_2 is the weight of the weight of the least weight path connecting v_1 and v_2. We study the problem of finding this distance in Section 5.

Definition 6.14 ((Sub)Graph Weight). Let (G, w) be a weighted graph with $G = (V, E)$. If $H = (V', E')$ is a subgraph of G, then the *weight of H* is:

$$w(H) = \sum_{e \in E'} w(e)$$

Definition 6.15 (Minimum Spanning Forrest Problem). Let (G, w) be a weighted graph with $G = (V, E)$. The *minimum spanning forest problem* for G is to find a forest $F = (V', E')$ that is a spanning subgraph of G that has the smallest possible weight.

Remark 6.16. If (G, w) is a weighted graph and G is connected, then the minimum spanning forest problem becomes the minimum spanning tree problem.

Example 6.17. A minimum spanning tree for the weighted graph shown in Figure 6.3 is shown in Figure 6.4. In the minimum spanning tree problem, we attempt to find a spanning

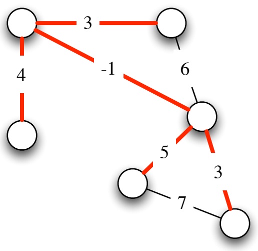


Figure 6.4. In the minimum spanning tree problem, we attempt to find a spanning subgraph of a graph G that is a tree and has minimal weight (among all spanning trees).

subgraph of a graph G that is a tree and has minimal weight (among all spanning trees).

We will verify that the proposed spanning tree is minimal when we derive algorithms for constructing a minimum spanning forest.

REMARK 6.18. The next algorithm, commonly called Prim's Algorithm [Pri57] will construct a minimum spanning tree for a connected graph.

```
Prim's Algorithm
Input: $(G, w)$ a weighted connected graph with $G = (V, E)$, $v_0$ a starting vertex
Initialize: $E' = \emptyset$ {The edge set of the spanning tree.}
Initialize: $V' = \{v_0\}$ {New vertices added to the spanning tree.}
    (1) while $V' \neq V$
    (2)     Set $X := V \setminus V'$
    (3)     Choose edge $e = \{v, v'\}$ so (i) $v \in V'$; (ii) $v' \in X$ and:
                $w(e) = \min_{u \in U, u' \in X} w\left(\{u, u'\}\right)$
    (4)     Set $E' = E' \cup \{e\}$
    (5)     Set $V' = V' \cup \{v'\}$
    (6) end while
Output: $T = (V', E')$ {$T$ is a minimum spanning tree.}
```

Algorithm 3. Prim's Algorithm

EXAMPLE 6.19. We illustrate the successive steps of Prim's Algorithm in Figure 6.5. At the start, we initialize our set $V' = \{1\}$ and the edge set is empty. At each successive iteration, we add an edge that connects a vertex in V' with a vertex not in V' that has minimum weight. Note at Iteration 2, we could have chosen to add either edge $\{1, 3\}$ or edge $\{4, 6\}$ the order doesn't matter, so any tie breaking algorithm will suffice. We continue adding edges until all vertices in the original graph are in the spanning tree.

THEOREM 6.20. *Let (G, w) be a weighted connected graph. Then Prim's algorithm returns a minimum spanning tree.*

PROOF. We will show by induction that at each iteration of Prim's algorithm, the tree (V', E') is a subtree of a minimum spanning tree T of (G, w). If this is the case, then at the termination of the algorithm, (V', E') must be equal to the minimum spanning tree T.

To establish the base case, not that at the first iteration $V' = \{v_0\}$ and $E' = \emptyset$ and therefore (V', E') must be a subtree of T a minimum spanning tree of (G, w). Now, suppose that the statement is true for all iterations up to and including k and let $T_k = (V', E')$ at iteration k. Suppose at iteration $k+1$ we add edge $e = \{v, v'\}$ to T_k to obtain $T_{k+1} = (U, F)$ with $U = V' \cup \{v'\}$ and $F = E' = E \cup \{e\}$. Suppose that T_{k+1} is not a subtree of T, then e is not an edge in T and thus e must generate a cycle in T. On this cycle, there is some edge $e' = \{u, u'\}$ with $u \in V'$ and $u' \notin V'$. At iteration $k+1$, we must have considered adding this edge to E' but by selection of e, we know that $w(e) \leq w(e')$ and thus if we construct T' from T by removing e' and adding e, we know that T' must span G (this is illustrated in Figure 6.6) and $w(T') \leq w(T)$ thus T' is a minimum spanning tree of G and T_{k+1} is a subtree of T'. The result follows by induction. □

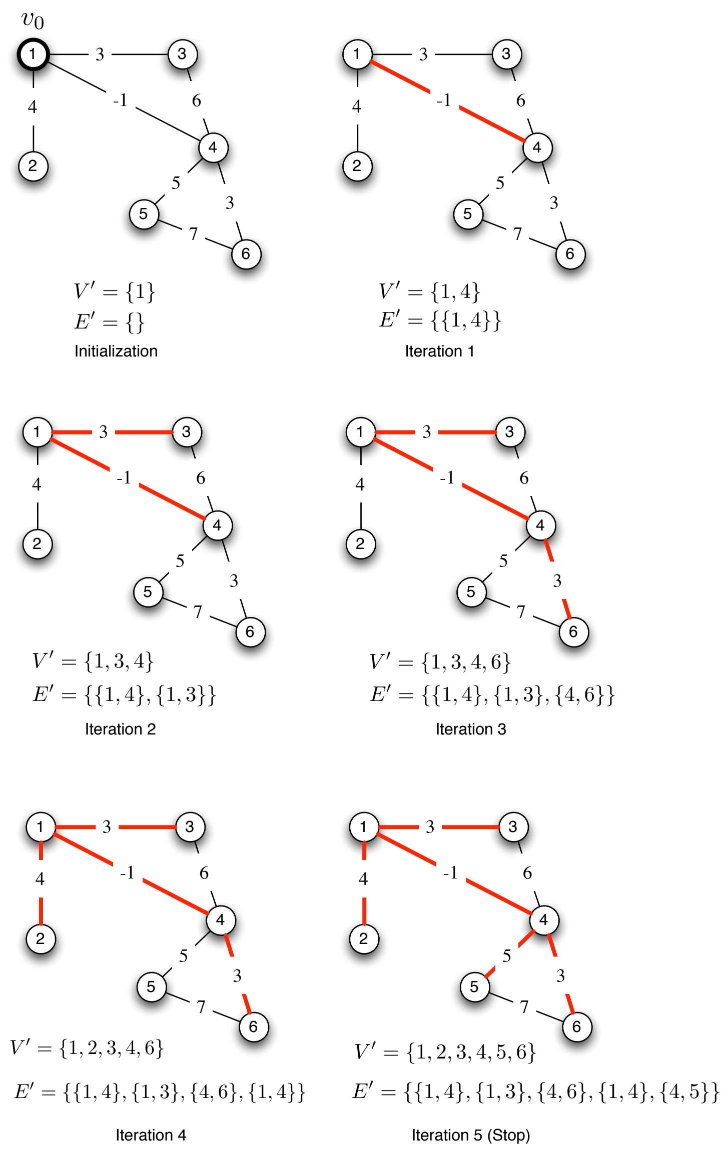


Figure 6.5. Prim's algorithm constructs a minimum spanning tree by successively adding edges to an acyclic subgraph until every vertex is inside the spanning tree. Edges with minimal weight are added at each iteration.

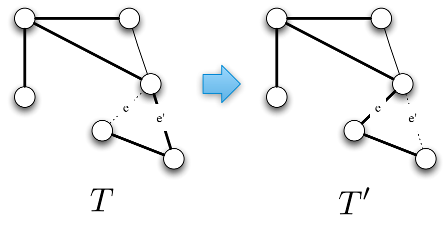
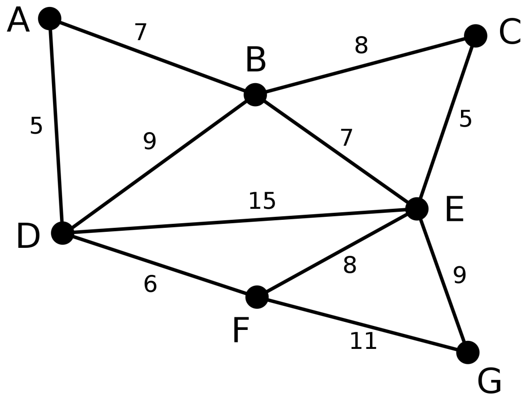


Figure 6.6. When we remove an edge (e') from a spanning tree we disconnect the tree into two components. By adding a new edge (e) edge that connects vertices in these two distinct components, we reconnect the tree and it is still a spanning tree.

Exercise 53. Use Prim's algorithm to find a minimum spanning tree for the graph shown below:



Exercise 54. Modify Algorithm 3 so that it returns a minimum spanning forest when G is not connected. Prove your algorithm works.

3. Computational Complexity of Prim's Algorithm

Remark 6.21. In this section, we'll discuss computational complexity. This is a subject that has its own course in many computer science departments (and some math departments). Therefore, we can only scrape the surface on this fascinating topic and we will *not* be able to provide completely formal definitions for all concepts. When definitions are informal, they will occur in remarks, rather than definition blocks.

Definition 6.22 (Big-O). Let $f, g : \mathbb{R} \to \mathbb{R}$. The function $f(x)$ is in the family $O(g(x))$ if there is an $N \in \mathbb{R}$ and a $c \in \mathbb{R}$ so that for all $x > N$, $|f(x)| \leq c|g(x)|$.

Remark 6.23. For the remainder of this section (and these notes) we will use the following, rather informal, definition of an algorithm. An *algorithm* is a set of steps (or operations) that can be followed to achieve a certain goal. We can think of an algorithm as having an input x and we will obtain an output y. If you're particularly interested in the formal definition of algorithms, see [**HU79**]. This material is *well outside* the scope of these notes, but it is important.

Example 6.24. Prim's Algorithm (Algorithm 3) is an example of an algorithm in this context. The inputs are the weighted graph (G, w) and the initial vertex v_0 and the output is the spanning tree T.

Remark 6.25. We have the following informal definition of *algorithmic running time*. The *running time* of an algorithm is the count of the number of steps an algorithm takes from the time we begin executing it to the time we obtain the output. We must be sure to include each time through any loops in the algorithm. This is not to be confused with the *wall clock* running time of an algorithm, which is dependent on the processor (a computer, a human, etc.) as well as the algorithmic details. Again, a more formal definition for algorithmic running time is given in [**HU79**] or you could take Math 457 [**Sim05**] or CSE 468.

Remark 6.26. In computing algorithmic running time, we need to be very careful in how we interpret the steps in the algorithm. For example, Prim's Algorithm uses the word "Choose" in line (3). But for a computer, this involves an enumeration of all the edges that might be connected to a specific vertex. If we reinterpret Prim's algorithm so that it uses an adjacency matrix, we can compute an exact running time. See Algorithm 4[1].

```
Prim's Algorithm (with Adjacency Matrix)
Input: $(G, w)$ a weighted connected graph with $G = (V, E)$, $\mathbf{M}$ the adjacency matrix of $G$, $v_0$ a starting vertex
Initialize: $E' = \emptyset$ {The edge set of the spanning tree.}
Initialize: $V' = \{v_0\}$ {New vertices added to the spanning tree.}
  (1) while $V' \neq V$
  (2)     Set $X := V \setminus V'$
  (3)     Set $e := \emptyset$
  (4)     Set $w^* = \infty$
  (5)     for each $v \in V'$
  (6)         Let $i$ be the row index of $v$ in $\mathbf{M}$
  (7)         for each $v' \in X$
  (8)             Let $j$ be the row index corresponding to row $v'$
  (9)             if $\mathbf{M}_{ij} \neq 0$ and $w(\{v, v'\}) < w^*$
 (10)                 $w^* = w(\{v, v'\})$
 (11)                 $e := \{v, v'\}$
 (12)             end if
 (13)         end for
 (14)     end for
 (15)     Set $E' = E' \cup \{e\}$
 (16)     Set $V' = V' \cup \{v'\}$
 (17) end while
Output: $T = (V', E')$ {$T$ is a minimum spanning tree.}
```

Algorithm 4. Prim's Algorithm (Matrix Form)

Theorem 6.27. *The running time of Algorithm 4 is $O(|V|^3)$.*

Proof. Consider the steps in the while loop starting at Line 1. If there are n vertices, then at iteration k of the while loop we know that $|V'| = k$ and $|X| = n - k$ since we add one new vertex to V' at each while loop iteration (and thus remove one vertex from X at each while loop iteration). The for loop beginning at Line 5 will have k iterations and the for loop starting at Line 7 will have $n - k$ iterations. This means that for any iteration of the while loop, we will perform $O(k(n - k))$ operations. Thus, for the whole algorithm we will perform:

$$O\left(\sum_{k=1}^{n-1} k(n-k)\right) = O\left(\frac{1}{3}n^3 - \frac{1}{6}n\right)$$

Thus, the running time for Algorithm 4 is $O(n^3) = O(|V|^3)$. □

[1]Algorithm 4 is not optimal. It is intentionally not optimal so that we can compute its complexity in Theorem 6.27 easily and we do not have to appeal to special data structures. See Remark 6.28 for more on this.

Remark 6.28. As it turns out, the implementation of Prim's algorithm can have a *substantial* impact on the running time. There are implementations of Prim's algorithm that run in $O(|V|^2)$, $O(|E|\log(|V|))$ and $O(|E| + |V|\log(|V|))$ [**CLRS01**]. Thus, we cannot just say, Prim's algorithm is an $O(g(x))$ algorithm, we must know which implementation of Prim's algorithm we are using. Clearly, the implementation in Algorithm 4 is not a very good one. To learn how to properly implement Prim's algorithm, you might consider taking CSE 465, which covers data structures and algorithms. You can also refer to [**CLRS01**].

Definition 6.29 (Polynomial Time). For a specific implementation of an algorithm, its running time is *polynomial* if there is some polynomial $p(x)$ so that when the running time of the algorithm is $f(x)$ then $f(x) \in O(p(x))$.

Theorem 6.30. *There is an implementation of Prim's algorithm that is polynomial.* □

4. Kruskal's Algorithm

Remark 6.31. In this section, we will discuss Kruskal's algorithm [**Kru56**], an alternative way to construct a minimum spanning tree of a weighted graph (G, w). The algorithm is shown in Algorithm 5.

```
Kruskal's Algorithm
Input: $(G, w)$ a weighted connected graph with $G = (V, E)$ and $n = |V|$
Initialize: $Q = E$
Initizlize: $V' = V$
Initizlize: $E' = \emptyset$
Initialize: For all $v \in V$ define $C(v) := \{v\}$ {$C(v)$ is the set of vertices connected to $v$ at each iteration.}
   (1) while $E'$ has fewer than $n - 1$ edges
   (2)     Choose the edge $e = (v, v')$ in $Q$ with minimum weight.
   (3)     if $C(v) \neq C(v')$
   (4)         for each $u \in C(v)$: $C(u) := C(u) \cup C(v')$
   (5)         for each $u \in C(v')$: $C(u) := C(u) \cup C(v)$
   (6)         $E' := E' \cup \{e\}$
   (7)         $Q := Q \setminus \{e\}$
   (8)     else
   (9)         $Q := Q \setminus \{e\}$
  (10)         GOTO 2
  (11)     end if
  (12) end while
Output: $T = (V', E')$ {$T$ is a minimum spanning tree.}
```

Algorithm 5. Kruskal's Algorithm

Example 6.32. We illustrate Kruskal's Algorithm in Figure 6.7. The spanning sub-graph starts with each vertex in the graph and no edges. In each iteration, we add the edge with the lowest edge weight provided that it does not cause a cycle to emerge in the existing sub-graph. In this example, there is never an edge chosen that causes a cycle to appear (because of the way the weights are chosen). In this example, the construction of

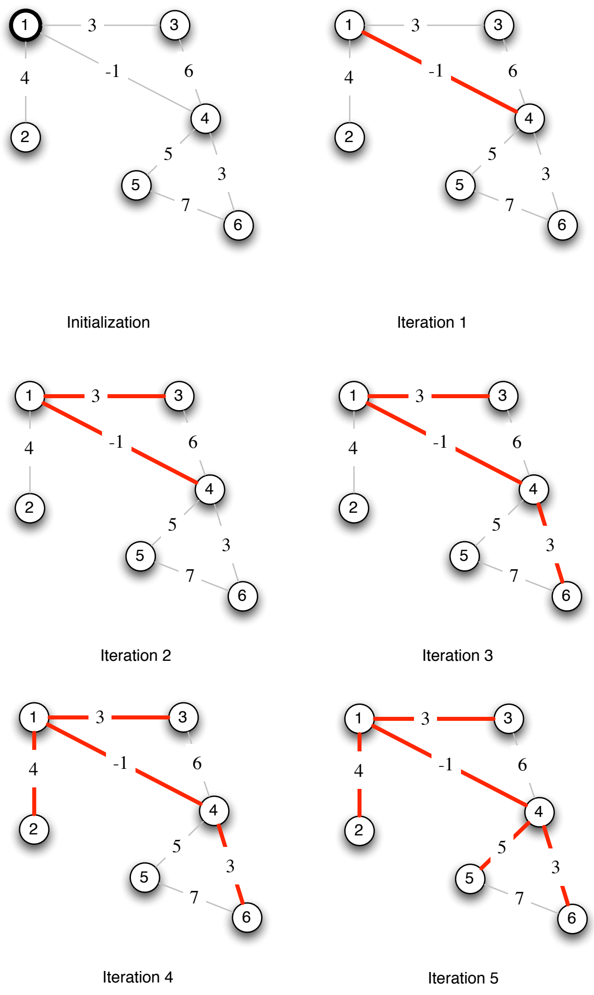


Figure 6.7. Kruskal's algorithm constructs a minimum spanning tree by successively adding edges and maintaining and acyclic disconnected subgraph containing every vertex until that subgraph contains $n - 1$ edges at which point we are sure it is a tree. Edges with minimal weight are added at each iteration.

the spanning tree occurs in exactly the same set of steps as Prim's algorithm. This is not always the case.

Exercise 55. Use Kruskal's Algorithm to determine a minimum spanning tree for the graph from Exercise 53.

Exercise 56. In the graph from Example 6.19, choose a starting vertex other than 1 and execute Prim's algorithm to show that Prim's Algorithm and Kruskal's algorithm do not always add edges in the same order.

Remark 6.33. We will prove the following theorem in the last section of this chapter using a very generalized method. It can be shown by induction, just as we did in Theorem 6.20.

Theorem 6.34. *Let (G, w) be a weighted connected graph. Then Kruskal's algorithm returns a minimum spanning tree.* $\square$

Remark 6.35. The proof of the following theorem is beyond the scope of this course, however it is useful to know the computational complexity of Kruskal's algorithm. See [**CLRS01**] for a proof.

Theorem 6.36. *There is an implementation of Kruskal's algorithm whose running time is* $O(|E|\log(|V|))$. □

Exercise 57. Compare the running time of an implementation of Kruskal's Algorithm $O(|E|\log(|V|))$ to the best running time of and implementation of Prim's algorithm $O(|E| + |V|\log(|V|))$. Under what circumstances might you use each algorithm? [Hint: Suppose that G has n vertices. Think about what happens when $|E|$ is big (say $n(n-1)/2$) vs. when $|E|$ is small (say 0). Try plotting the two cases for various sizes of n.]

5. Shortest Path Problem in a Positively Weighted Graph

Remark 6.37. The shortest path problem in a weighted graph is the problem of finding the least weight path connecting a given vertex v to a given vertex v'. Dijkstra's Algorithm [**Dij59**] answers this question by growing a spanning tree starting at v so that the unique path from v to any other vertex v' in the tree is the shortest. The algorithm is shown in Algorithm 6. It is worth noting that this algorithm *only* works when the weights in the graph are all positive. We will discuss Floyd's Algorithm [**Flo62**] for this instance when we discuss Network Programming in a later chapter.

Example 6.38. An example execution of Dijkstra's Algorithm is shown in Figure 6.8. At the start of the algorithm, we have Vertex 1 (v_0) as the vertex in the set Q that is closest to v_0 (it has distance 0, obviously). Investigating its neighbor set, we identify three vertices 2, 3 and 4 and the path length from Vertex 1 to each of these vertices is smaller than the initialized distance of ∞ and so these vertices are assigned a parent ($p(v)$) as Vertex 1 and the new distances are recorded. Vertex 1 is then removed from the set Q. In the second iteration, we see that Vertex 3 is closest to v_0 (Vertex 1) and investigating its neighborhood, we see that the distance from Vertex 1 to 3 and then from 3 to 4 is 9 and smaller than the currently recorded distance of Vertex 1 to Vertex 4. Thus, we update the parent function of Vertex 4 so that it returns Vertex 3 instead of Vertex 1, as well as the distance function, and continue to the next iteration. The next closest vertex to v_0 is Vertex 2. Investigating its neighbors shows that no changes need to be made to the distance or parent function. We continue in this way until all the vertices are exhausted.

Theorem 6.39. *Let (G, w) be a weighted graph with vertex v_0. Then Dijkstra's algorithm returns a spanning tree T so that the distance from v_0 to any vertex v in T is the minimum distance from v_0 to v in (G, w).*

Proof. We proceed by induction to show that the distances from v_0 to every vertex $v \in V \setminus Q$ are correct when v is removed from Q. To do this, define $X = V \setminus Q$ and let T_k be the tree generated by the vertices in X and the function $p(v)$.

In the base case, when v_0 is removed from Q (added to X) it is clear that $d(v_0, v_0) = 0$ is correct. Now assume that the statement is true up through the k^{th} iteration so that (i) for any vertex v' added to X prior to the k^{th} iteration, $d(v_0, v')$ is correct and the unique path

```
Dijkstra's Algorithm
Input: $(G, w)$ a weighted connected graph with $G = (V, E)$, $v_0$ an initial vertex.
Initizlize: $Q = V$
Initizlize: $E' = \emptyset$
Initialize: For all $v \in V$ if $v \neq v_0$ define $d(v_0, v) := \infty$ otherwise define $d(v_0, v) := 0$ {$d(v_0, v)$ is
the best distance from $v_0$ to $v$.}
Initialize: For all $v \in V$, $p(v) :=$ undefined {A "parent" function that will be used to build
the tree.}
   (1)  while $Q \neq \emptyset$
   (2)      Choose $v \in Q$ so that $d(v_0, v)$ is minimal
   (3)      $Q := Q \setminus \{v\}$
   (4)      for each $v' \in N(v)$
   (5)          Define $\delta(v_0, v') := d(v_0, v) + w(\{v, v'\})$
   (6)          if $\delta(v_0, v') < d(v_0, v')$
   (7)              $d(v_0, v') = \delta(v_0, v')$
   (8)              $p(v') = v$
   (9)          end if
  (10)      end for
  (11)  end while
  (12)  Set $V' := V$
  (13)  Set $E' = \emptyset$
  (14)  for each $v \in V$
  (15)      if $p(v) \neq$ undefined
  (16)          $E' = E' \cup \{v, p(v)\}$
  (17)      end if
  (18)  end for
Output: $T = (V', E')$ and $d(v_0, \cdot)$ {$T$ is a Dijkstra tree, $d(v_0, \cdot)$ provides the distances.}
```

Algorithm 6. Dijkstra's Algorithm (Adapted from Wikipedia's Pseudo-code, http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)

in T_k from v_0 to v' defined by the function $p(v)$ is the minimum distance path from v_0 to v' in (G, w).

Before proceeding note, that for any vertex v' added to X at iteration k, $p(v')$ is fixed permanently after that iteration. Thus, the path from v_0 to v' in T_k is the same as the path from v_0 to v' in T_{k+1}. Thus, assuming that $d(v_0, v')$ and $p(v)$ are correct at iteration k means it must also hold at future iterations (or more generally) that it is correct for (G, w).

Suppose vertex v is added to set X (removed from Q) at iteration $k+1$ but the shortest path from v_0 to v is not the unique path from v_0 to v in the tree T_{k+1} constructed from the vertices in X and the function $p(v)$. Since G is connected, there is a shortest path and we now have two possibilities: (i) the shortest path connecting v_0 to v passes through a vertex not in X or (ii) the shortest path connecting v_0 to v passes through only vertices in X.

In the first case, if the true shortest path connecting v_0 to v passes through a vertex u not in X, then we have two new possibilities: (i) $d(v_0, u) = \infty$ or $d(v_0, u) = r < \infty$. We may dismiss the first case as infeasible, and thus we have $d(v_0, u) = r < \infty$. In order for the distance from v_0 to v to be less along the path containing u, we know that $d(v_0, u) < d(v_0, v)$. But if that's true, then in Step 2 of Algorithm 6, we should have evaluated the neighborhood

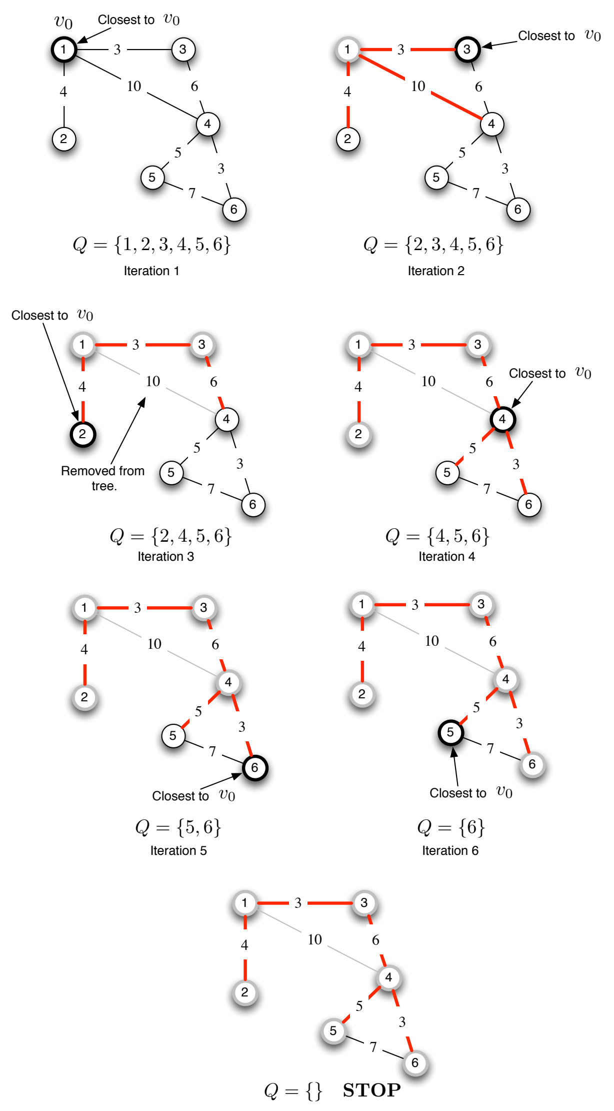
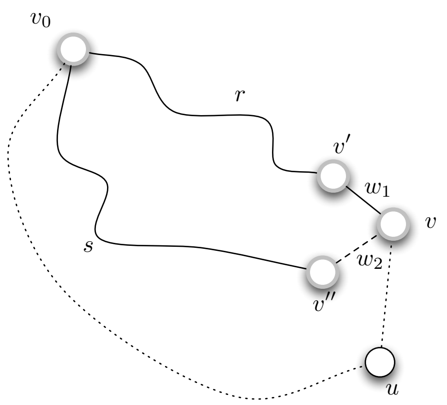


Figure 6.8. Dijkstra's Algorithm iteratively builds a tree of shortest paths from a given vertex v_0 in a graph. Dijkstra's algorithm can correct itself, as we see from Iteration 2 and Iteration 3.

of u well before evaluating the neighborhood of v in the for-loop starting at Line 4 and thus u must be an element of X (i.e., not in Q). This leads to the second case.

Suppose now that the true shortest path from v_0 to v leads to a vertex v'' before reaching v while the path recorded in T_{k+1} reaches v' before reaching v as illustrated below.



Ignore this path, it contains vertices not in X

Let $w_1 = w(v', v)$ and $w_2 = w(v'', v)$. Then it follows that $d(v_0, v') + w_1 > d(v_0, v'') + w_2$. By the induction hypothesis, $d(v_0, v')$ and $d(v_0, v'')$ are both correct as is their path in T_{k+1}. However, since both v' and v'' are in X, we know that the neighborhoods of both these vertices were evaluated in the for-loop at Line 4. If $p(v) = v''$ when $N(v')$ was evaluated, then $p(v) = v''$ since Line 6 specifically forbids changes to $p(v)$ unless $d(v_0, v') + w_1 < d(v_0, v'') + w_2$. On the other hand, if $p(v) = v'$ when $N(v'')$ was evaluated, then it's clear at once that $p(v) = v''$ at the end of the evaluation of the if-statement at Line 6. In either case, $d(v_0, v)$ could not be incorrect in T_{k+1}. The correctness of Dijkstra's algorithm follows from induction. □

Remark 6.40. The following theorem has a proof that is outside the scope of the course. See [**CLRS01**] for details.

Theorem 6.41. *There is an implementation of Dijkstra's Algorithm that has running time in* $O\left(|E| + |V|\log(|V|)\right)$. □

Remark 6.42. Dijkstra's Algorithm is an example of a *Dynamic Programming* [**Bel57**] approach to finding the shortest path in a graph. Dynamic programming a sub-discipline of Mathematical Programming (or Optimization), which we will encounter in the coming chapters.

Exercise 58. Use Dijkstra's algorithm to grow a Dijkstra tree for the graph in Exercise 53 starting at vertex D. Find the distance from D to each vertex in the graph.

Exercise 59 (Project). The A^* heuristic is a variation on Dijkstra's Algorithm, which in the worst case defaults to Dijkstra's Algorithm. It is fundamental to the study of Artificial Intelligence. Investigate the A^* heuristic, describe how it operates and compare it to Dijkstra's Algorithm. Create two examples for the use of the A^* heuristic, one that out-performs Dijkstra's Algorithm and the other that defaults to Dijkstra's algorithm. You do not have to code the algorithms, but you can.

Exercise 60 (Project). Using [**CLRS01**] (or some other book on algorithms) implement Breadth and Depth First Search (for generating a spanning tree), Prim's, Kruskal's and Dijkstra's algorithm in the language of your choice. Write code to generate a connected graph with an arbitrarily large number of vertices and edges. Empirically test the running time of your three algorithms to see how well the predicted running times match the actual running times as a function of the number of vertices and edges. Using these empirical results, decide whether your answer to Exercise 57 was correct or incorrect.

6. Greedy Algorithms and Matroids

Definition 6.43 (Hereditary System). A *hereditary system* is a pair $(E, \mathcal{I})$ so that E is a finite set and $\mathcal{I} \subset 2^E$ a set of *independent sets* such that if $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$.

Remark 6.44. Any subset of E that is not in $\mathcal{I}$ is called a *dependent set*.

Proposition 6.45. *If $(E, \mathcal{I})$ is a hereditary system, then $\emptyset \in \mathcal{I}$.*

Exercise 61. Prove Proposition 6.45.

Proposition 6.46. *Suppose $G = (V, E)$ is a graph and let $\mathcal{I}$ be the set of subsets of E such that if $E' \in \mathcal{I}$ then the subgraph of G induced by E' is a acyclic (e.g., a sub-forest of G). Then $(E, \mathcal{I})$ is a hereditary system.*

Exercise 62. Prove Proposition 6.46.

Definition 6.47 (Weighted Hereditary System). A *weighted hereditary system* is a triple $(E, \mathcal{I}, w)$ so that $(E, \mathcal{I})$ is a hereditary system and $w : E \to \mathbb{R}$ is a weight function on the elements of E.

Example 6.48. In light of Proposition 6.46 we could think of a weighted graph (G, w) with $G = (V, E)$ as giving rise to a weighted hereditary system $(E, \mathcal{I}, w)$ so that $\mathcal{I}$ is the collection of edge subsets of E that induce acyclic graphs and w is just the edge weighting.

Definition 6.49 (Minimum Weight Problem). Let $(E, \mathcal{I}, w)$ be a weighted hereditary system. Then the *minimum weight problem* is to identify a set $E' \in \mathcal{I}$ (an independent set) such that:

$$w(E') = \sum_{e \in E'} w(e) \tag{6.1}$$

is as small as possible and E' is a maximal subset of E (that is there is no other set $I \in \mathcal{I}$ so that $E' \subset I$).

Remark 6.50. One can define a maximum weight problem in precisely the same way, if we replace the word minimum with maximum and small with large. The following algorithm called the Greedy Algorithm can be used (in some cases) to solve the minimum weight problem. The name, 'Greedy' makes a little more sense for maximum weight problems, but the examples we give are minimum weight problems.

Remark 6.51. Let (G, w) be a weighted graph and consider the weighted hereditary system with $(E, \mathcal{I})$ with $\mathcal{I}$ the collection of edge subsets of E that induce acyclic graphs and w is just the edge weighting. Kruskal's Algorithm is exactly a greedy algorithm. We begin with the complete set of edges and continue adding them to the forest (acyclic subgraph of a given weighted graph (G, w)), each time checking to make sure that the added edge does not induce a cycle (that is, that we have an element of $\mathcal{I}$). We will use this fact to prove Theorem 6.34.

```
Greedy Algorithm
Input: $(E, \mathcal{I}, w)$ a weighted hereditary system
Initizlize: $E' = \emptyset$
Initizlize: $A = E$
    (1) while $A \neq \emptyset$
    (2)         Choose $e \in A$ to minimize $w(e)$
    (3)         $A := A \setminus \{e\}$
    (4)         if $E' \cup \{e\} \in \mathcal{I}$
    (5)                 $E' := E' \cup \{e\}$
    (6)         end if
    (7) end while
Output: $E'$
```

Algorithm 7. Greedy Algorithm (Minimization)

Definition 6.52 (Matroid). Let $M = (E, \mathcal{I})$ be a hereditary system. Then M is a *matroid* if it satisfies the *augmentation property*: If $I, J \in \mathcal{I}$ and $|I| < |J|$ then there is some $e \in E$ so that $e \in J$ and $e \notin I$ and so that $I \cup \{e\} \in \mathcal{I}$.

Remark 6.53. Definition 6.52 essentially says that if there are two independent sets (as an example, you could think acyclic subgraphs) and one has greater cardinality than the other, then there is some element (edge) that can be added to the independent set (acyclic graph) with smaller cardinality so that this new set is still independent (an acyclic graph).

Theorem 6.54. *Let $(E, \mathcal{I}, w)$ be a weighted hereditary system. The structure $M = (E, \mathcal{I})$ is a matroid if and only if the greedy algorithm solves the minimum weight problem associated with M.*

Proof. ($\Rightarrow$) Let $I = \{e_1, \dots, e_n\}$ be the set in $\mathcal{I}$ identified by the greedy algorithm and suppose that $J = \{f_1, \dots, f_m\}$ be any other maximal element of $\mathcal{I}$. Without loss of generality, assume that:

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_n)$$
$$w(f_1) \leq w(f_2) \leq \cdots \leq w(f_m)$$

Assume that $|J| > |I|$; then by the augmentation property there is an element $e \in J$, not in I such that $I \cup \{e\}$ is in $\mathcal{I}$, but this element would have been identified during execution of the greedy algorithm. By the same argument, $|J| \not< |I|$ since again by the augmentation property we could find an element e so that $J \cup \{e\} \in \mathcal{I}$ and thus J is not maximal.

Therefore, $|I| = |J|$ or more specifically $m = n$. Assume $I_k = \{e_1, \dots, e_k\}$ and $J_k = \{f_1, \dots, f_k\}$ for $k = 1, \dots, n$ (thus I_1 and J_1 each have one element, I_2 and J_2 each have two elements etc.) It now suffices to show that if

$$w(I_k) = \sum_{i=1}^{k} w(e_i)$$

then $w(I_k) \leq w(J_k)$ for all $k = 1, \dots, n$. We proceed by induction. Since the Greedy Algorithm selects the element e with smallest weight first, it is clear that $w(I_1) \leq w(J_1)$, thus we have established the base case. Now assume that the statement is true up through some arbitrary $k < n$. By definition, we know that $|J_{k+1}| > |I_k|$ and therefore by the augmentation property there is some $e \in J_{k+1}$ with $e \not\in I_k$ so that $I_k \cup \{e\}$ is an element of $\mathcal{I}$. It follows that $w(e_{k+1}) \leq w(e)$ because otherwise, the Greedy Algorithm would have chosen e instead of e_{k+1}. Furthermore, $w(e) \leq w(f_{k+1})$ since the elements of I and J are listed in ascending order and $e \in J_{k+1}$. Thus, $w(e_{k+1}) \leq w(e) \leq w(f_{k+1})$ and therefore we conclude that $w(I_{k+1}) \leq w(J_{k+1})$. The result follows by induction at once.

($\Leftarrow$) We will proceed by contrapositive to prove that M is a matroid. Suppose that the augmentation property is not satisfied and consider I and J in $\mathcal{I}$ with $|I| < |J|$ so that there is *no* element $e \in J$ with $e \not\in I$ so that $I \cup \{e\}$ is in $\mathcal{I}$. Without loss of generality, assume that $|I| = |J| + 1$. Let $|I| = p$ and consider the following weight function:

$$w(e) = \begin{cases} -p-2 & \text{if } e \in I \\ -p-1 & \text{if } e \in J \setminus I \\ 0 & \text{else} \end{cases}$$

After the Greedy algorithm chooses all the elements of I, it cannot decrease the weight of the independent set because only elements that are not in J will be added. Thus, the total weight will be $-p(p+2) - p^2 - 2p$. However, the set J has weight $-(p+1)(p+1) = -p^2 - 2p - 2$. Thus any set independent set containing J has weight at most $-p^2 - 2p - 2$. Thus, the Greedy Algorithm cannot identify a maximal independent set with minimum weight when the augmentation property is not satisfied. Thus by contrapositive we have shown that if the Greedy Algorithm identifies a maximal independent set with minimal weight, then M must be a matroid. This completes the proof. $\square$

Theorem 6.55. *Let $G = (V, E)$ be a graph. Then the hereditary system $M(G) = (E, \mathcal{I})$ where $\mathcal{I}$ is the collection of subsets that induce acyclic graphs is a matroid.*

Proof. From Proposition 6.46, we know that $(E, \mathcal{I})$ is a hereditary system, we must simply show that it has the augmentation property. To see this, let I and J be two elements of $\mathcal{I}$ with $|I| < |J|$. Let H be the subgraph of G induced from the edge sets $I \cup J$. Let F be a spanning forest of this subgraph H that contains I. We know from Corollary 3.60 that H has a spanning subgraph and we know that we can construct such a graph using the technique from the proof of Theorem 3.59.

Since J is acyclic, F has at least as many edges as J and therefore there exists at least one edge e that is in the forest F but that does not occur in the set I and furthermore, it must be an element of J (by construction of H). Since e is an edge in F, it follows that the subgraph induced by the set $I \cup \{e\}$ is acyclic and therefore, $I \cup \{e\}$ is an element of $\mathcal{I}$. Thus $M(G)$ has the augmentation property and is a matroid. $\square$

Corollary 6.56 (Theorem 6.34). *Let (G, w) be a weighted graph. Then Kruskal's algorithm returns a minimum spanning tree when G is connected.*

Exercise 63. Prove Theorem 6.34.

Remark 6.57. Matroid Theory is a very active and deep area of research in combinatorics and combinatorial optimization theory. A complete study of this field is well outside the scope of this course. The interested reader should consider a text on Matroid Theory like [Oxl92].

CHAPTER 7

A Brief Introduction to Linear Programming

Remark 7.1. It turns out the many graph theoretic problems can be expressed as linear optimization problems. Furthermore, the proofs of some of the most fundamental theorems of graph theory are greatly simplified by the use of a linear optimization formulation.

Even though it seems like we're going to go far off topic, we use this chapter to introduce Linear Optimization and its fundamental results. We will then use these results to prove key results in Graph Theory and thereby illustrate the link between the theory of optimization and the theory of graphs.

1. Linear Programming: Notation

Definition 7.2 (Linear Programming Problem). A *linear programming* problem is an optimization problem of the form:

$$\left\{\begin{aligned} \max\;\; & z(x_1,\dots,x_n) = c_1x_1 + \cdots + c_nx_n \\ s.t.\;\; & a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ & \qquad\vdots \\ & a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \\ & h_{11}x_1 + \cdots + h_{n1}x_n = r_1 \\ & \qquad\vdots \\ & h_{l1}x_1 + \cdots + h_{ln}x_n = r_l \end{aligned}\right. \tag{7.1}$$

Remark 7.3. You will recall from your matrices class (Math 220) that matrices can be used as a short hand way to represent linear equations. Consider the following system of equations:

$$\left\{\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}\right. \tag{7.2}$$

Then we can write this in matrix notation as:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{7.3}$$

where $\mathbf{A}_{ij} = a_{ij}$ for $i = 1,\dots,m$, $j = 1,\dots,n$ and $\mathbf{x}$ is a column vector in $\mathbb{R}^n$ with entries x_j $(j = 1,\dots,n)$ and $\mathbf{b}$ is a column vector in $\mathbb{R}^m$ with entries b_i $(i = 1\dots,m)$. Obviously, if we replace the equalities in Expression 7.2 with inequalities, we can also express systems of inequalities in the form:

$$\mathbf{Ax} \leq \mathbf{b} \tag{7.4}$$

Using this representation, we can write our general linear programming problem using matrix and vector notation. Expression 7.1 can be written as:

$$\begin{cases} \max \quad z(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \qquad\quad s.t. \quad \mathbf{Ax} \leq \mathbf{b} \\ \qquad\qquad\quad \mathbf{Hx} = \mathbf{r} \end{cases} \tag{7.5}$$

Definition 7.4. In Problem 7.5, if we restrict some of the decision variables (the x_i's) to have only integer (or discrete) values, then the problem becomes a mixed integer linear programming problem. If all of the variables are restricted to integer values, the problem is an integer programming problem and if every variable can only take on the values 0 or 1, the program is called a $0-1$ integer programming problem. [WN99] is an excellent reference for Integer Programming.

2. Intuitive Solutions of Linear Programming Problems

Example 7.5. Consider the problem of a toy company that produces toy planes and toy boats. The toy company can sell its planes for \$10 and its boats for \$8 dollars. It costs \$3 in raw materials to make a plane and \$2 in raw materials to make a boat. A plane requires 3 hours to make and 1 hour to finish while a boat requires 1 hour to make and 2 hours to finish. The toy company knows it will not sell anymore than 35 planes per week. Further, given the number of workers, the company cannot spend anymore than 160 hours per week finishing toys and 120 hours per week making toys. The company wishes to maximize the profit it makes by choosing how much of each toy to produce.

We can represent the profit maximization problem of the company as a linear programming problem. Let x_1 be the number of planes the company will produce and let x_2 be the number of boats the company will produce. The profit for each plane is $\$10 - \$3 = \$7$ per plane and the profit for each boat is $\$8 - \$2 = \$6$ per boat. Thus the total profit the company will make is:

$$z(x_1, x_2) = 7x_1 + 6x_2 \tag{7.6}$$

The company can spend no more than 120 hours per week making toys and since a plane takes 3 hours to make and a boat takes 1 hour to make we have:

$$3x_1 + x_2 \leq 120 \tag{7.7}$$

Likewise, the company can spend no more than 160 hours per week finishing toys and since it takes 1 hour to finish a plane and 2 hour to finish a boat we have:

$$x_1 + 2x_2 \leq 160 \tag{7.8}$$

Finally, we know that $x_1 \leq 35$, since the company will make no more than 35 planes per week. Thus the complete linear programming problem is given as:

$$(7.9) \qquad \begin{cases} \max & z(x_1, x_2) = 7x_1 + 6x_2 \\ s.t. & 3x_1 + x_2 \leq 120 \\ & x_1 + 2x_2 \leq 160 \\ & x_1 \leq 35 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{cases}$$

Remark 7.6. Strictly speaking, the linear programming problem in Example 7.5 is not a true linear programming problem because we don't want to manufacture a fractional number of boats or planes and therefore x_1 and x_2 must really be drawn from the *integers* and not the real numbers (a requirement for a linear programming problem). This type of problem is generally called an integer programming problem. However, we will ignore this fact and assume that we can indeed manufacture a fractional number of boats and planes. If you're interested in this distinction, you might consider taking Math 484, where we discuss this issue in depth.

Linear Programs (LP's) with two variables can be solved graphically by plotting the feasible region along with the level curves of the objective function. We will show that we can find a point in the feasible region that maximizes the objective function using the level curves of the objective function. We illustrate the method first using the problem from Example 7.5.

Example 7.7 (Continuation of Example 7.5). Let's continue the example of the Toy Maker begin in Example 7.5. To solve the linear programming problem graphically, begin by drawing the feasible region. This is shown in the blue shaded region of Figure 7.1.

After plotting the feasible region, the next step is to plot the level curves of the objective function. In our problem, the level sets will have the form:

$$7x_1 + 6x_2 = c \implies x_2 = \frac{-7}{6}x_1 + \frac{c}{6}$$

This is a set of parallel lines with slope $-7/6$ and intercept $c/6$ where c can be varied as needed. The level curves for various values of c are parallel lines. In Figure 7.1 they are shown in colors ranging from red to yellow depending upon the value of c. Larger values of c are more yellow.

To solve the linear programming problem, follow the level sets along the gradient (shown as the black arrow) until the last level set (line) intersects the feasible region. If you are doing this by hand, you can draw a single line of the form $7x_1 + 6x_2 = c$ and then simply draw parallel lines in the direction of the gradient $(7, 6)$. At some point, these lines will fail to intersect the feasible region. The last line to intersect the feasible region will do so at a point that maximizes the profit. In this case, the point that maximizes $z(x_1, x_2) = 7x_1 + 6x_2$, subject to the constraints given, is $(x_1^*, x_2^*) = (16, 72)$.

Note the point of optimality $(x_1^*, x_2^*) = (16, 72)$ is at a corner of the feasible region. This corner is formed by the intersection of the two lines: $3x_1 + x_2 = 120$ and $x_1 + 2x_2 = 160$. In

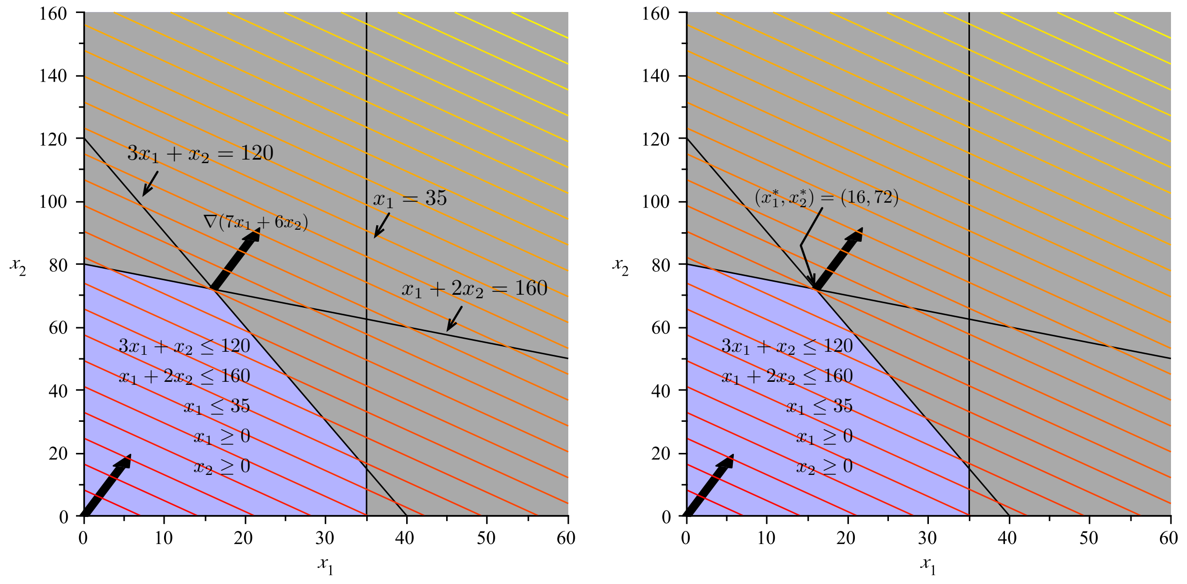


Figure 7.1. Feasible Region and Level Curves of the Objective Function: The shaded region in the plot is the feasible region and represents the intersection of the five inequalities constraining the values of x_1 and x_2. On the right, we see the optimal solution is the "last" point in the feasible region that intersects a level set as we move in the direction of increasing profit.

this case, the constraints

$$3x_1 + x_2 \leq 120$$
$$x_1 + 2x_2 \leq 160$$

are both *binding*, while the other constraints are non-binding. In general, we will see that when an optimal solution to a linear programming problem exists, it will always be at the intersection of several binding constraints; that is, it will occur at a corner of a higher-dimensional polyhedron.

Remark 7.8. It can sometimes happen that a linear programming problem has an infinite number of alternative optimal solutions. We illustrate this in the next example.

Example 7.9. Suppose the toy maker in Example 7.5 finds that it can sell planes for a profit of \$18 each instead of \$7 each. The new linear programming problem becomes:

$$(7.10)\quad \begin{cases} \max & z(x_1, x_2) = 18x_1 + 6x_2 \\ s.t. & 3x_1 + x_2 \leq 120 \\ & x_1 + 2x_2 \leq 160 \\ & x_1 \leq 35 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{cases}$$

Applying our graphical method for finding optimal solutions to linear programming problems yields the plot shown in Figure 7.2. The level curves for the function $z(x_1, x_2) = 18x_1 + 6x_2$

are *parallel* to one face of the polygon boundary of the feasible region. Hence, as we move further up and to the right in the direction of the gradient (corresponding to larger and larger values of $z(x_1, x_2)$) we see that there is not *one* point on the boundary of the feasible region that intersects that level set with greatest value, but instead a side of the polygon boundary described by the line $3x_1 + x_2 = 120$ where $x_1 \in [16, 35]$. Let:

$$S = \{(x_1, x_2 | 3x_1 + x_2 \leq 120,\ x_1 + 2x_2 \leq 160,\ x_1 \leq 35,\ x_1, x_2 \geq 0\}$$

that is, S is the feasible region of the problem. Then for any value of $x_1^* \in [16, 35]$ and any value x_2^* so that $3x_1^* + x_2^* = 120$, we will have $z(x_1^*, x_2^*) \geq z(x_1, x_2)$ for all $(x_1, x_2) \in S$. Since there are infinitely many values that x_1 and x_2 may take on, we see this problem has an infinite number of alternative optimal solutions.

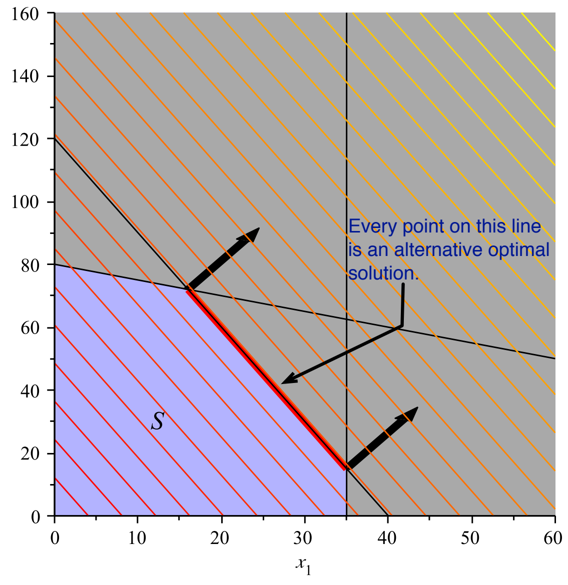


Figure 7.2. An example of infinitely many alternative optimal solutions in a linear programming problem. The level curves for $z(x_1, x_2) = 18x_1 + 6x_2$ are *parallel* to one face of the polygon boundary of the feasible region. Moreover, this side contains the points of greatest value for $z(x_1, x_2)$ inside the feasible region. Any combination of (x_1, x_2) on the line $3x_1 + x_2 = 120$ for $x_1 \in [16, 35]$ will provide the largest possible value $z(x_1, x_2)$ can take in the feasible region S.

3. Some Basic Facts about Linear Programming Problems

Definition 7.10 (Canonical Form). A maximization linear programming problem is in *canonical form* if it is written as:

$$(7.11)\quad \begin{cases} \max \quad z(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \qquad s.t.\ \ \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ \qquad\qquad \mathbf{x} \geq 0 \end{cases}$$

A minimization linear programming problem is in *canonical form* if it is written as:

$$\left\{\begin{aligned} \min \quad z(\mathbf{x}) = &\mathbf{c}^T\mathbf{x} \\ s.t. \quad &\mathbf{A}\mathbf{x} \geq \mathbf{b} \\ &\mathbf{x} \geq 0 \end{aligned}\right. \tag{7.12}$$

Definition 7.11 (Standard Form). A linear programming problem is in *standard form* if it is written as:

$$\left\{\begin{aligned} \max \quad z(\mathbf{x}) = &\mathbf{c}^T\mathbf{x} \\ s.t. \quad &\mathbf{A}\mathbf{x} = \mathbf{b} \\ &\mathbf{x} \geq 0 \end{aligned}\right. \tag{7.13}$$

Remark 7.12. The following theorem is outside the scope of the course. You may cover it in a Math 484 [Gri11b].

Theorem 7.13. *Every linear programming problem in canonical form can be put into standard form.* □

Exercise 64. Show that a minimization linear programming problem in canonical form can be rephrased as a maximization linear programming problem in canonical form. [Hint: Multiply the objective and constraints -1. Define new matrices.]

Remark 7.14. To illustrate Theorem 7.13, we note that it is relatively easy to convert any inequality constraint into an equality constraint. Consider the inequality constraint:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i \tag{7.14}$$

We can add a new *slack variable* s_i to this constraint to obtain:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + s_i = b_i$$

Obviously this slack variable $s_i \geq 0$. The slack variable then becomes just another variable whose value we must discover as we solve the linear program for which Expression 7.14 is a constraint.

We can deal with constraints of the form:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i \tag{7.15}$$

in a similar way. In this case we subtract a surplus variable s_i to obtain:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - s_i = b_i$$

Again, we must have $s_i \geq 0$.

Example 7.15. Consider the linear programming problem:

$$\left\{\begin{aligned} \max \quad & z(x_1, x_2) = 2x_1 - x_2 \\ s.t. \quad & x_1 - x_2 \leq 1 \\ & 2x_1 + x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{aligned}\right.$$

This linear programming problem can be put into standard form by using both a slack and surplus variable.

$$\begin{cases} \max \;\; z(x_1, x_2) = 2x_1 - x_2 \\ \;\; s.t. \;\; x_1 - x_2 + s_1 = 1 \\ \qquad\;\; 2x_1 + x_2 - s_2 = 6 \\ \qquad\;\; x_1, x_2, s_1, s_2 \geq 0 \end{cases}$$

Definition 7.16 (Row Rank). Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. The *row rank* of $\mathbf{A}$ is the size of the largest set of row (vectors) from $\mathbf{A}$ that are linearly independent.

Example 7.17. The row rank of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

is 2. To see this note that:

$$\begin{bmatrix} 7 & 8 & 9 \end{bmatrix} = -\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} + 2\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

It is also clear that [1 2 3] and [4 5 6] are linearly independent. Thus showing that the row rank of $\mathbf{A}$ is 2.

Remark 7.18. The column rank of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is defined analogously on columns rather than rows. The following theorem relates the row and column rank. It's proof is outside the scope of the course.

Theorem 7.19. *If $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a matrix, then the row rank of $\mathbf{A}$ is equal to the column rank of $\mathbf{A}$. Further,* $\text{rank}(\mathbf{A}) \leq \min\{m, n\}$. □

Definition 7.20. Suppose that $\mathbf{A} \in \mathbb{R}^{m \times n}$ and let $m \leq n$. Then $\mathbf{A}$ has *full row rank* if $\text{rank}(\mathbf{A}) = m$.

Remark 7.21. We will assume, when dealing with Linear Programming Problems in standard or canonical form that the matrix $\mathbf{A}$ has full row rank and if not, we will adjust it so this is true. The following theorem tells us what can happen in a Linear Programming Problem.

Theorem 7.22. *Consider any linear programming problem:*

$$P \begin{cases} \max \;\; z(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \qquad s.t. \;\; \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ \qquad\qquad \mathbf{x} \geq 0 \end{cases}$$

Then there are exactly four possibilities:

(1) *There is a unique solution to problem P denoted $\mathbf{x}^*$.*
(2) *There are an infinite number of alternative optimal solutions to P.*
(3) *There is no solution to P because there is no $\mathbf{x}$ that satisfies $\mathbf{A}\mathbf{x} = \mathbf{b}$.*
(4) *There is no solution to P because the problem is unbounded. That is for any $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{b}$ there is another $\mathbf{x}' \neq \mathbf{x}$ so that $\mathbf{A}\mathbf{x}' = \mathbf{b}$ and $\mathbf{c}^T\mathbf{x} < \mathbf{c}^T\mathbf{x}'$.*

4. Solving Linear Programming Problems with a Computer

Remark 7.23. There are a few ways to solve Linear Programming problems. The most common approach is called the *Simplex Algorithm*. Unfortunately, we will not have time to cover the Simplex Algorithm in this class. This is covered in IE 405 and Math 484, for those interested [Gri11b].

Remark 7.24. We'll show how to solve Linear Programs using Matlab. Matlab assumes that all linear programs are input in the following form:

$$\left\{\begin{aligned} \min \quad z(\mathbf{x}) = & \mathbf{c}^T\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{H}\mathbf{x} = \mathbf{r} \\ & \mathbf{x} \geq \mathbf{l} \\ & \mathbf{x} \leq \mathbf{u} \end{aligned}\right. \tag{7.16}$$

Here $\mathbf{c} \in \mathbb{R}^{n\times 1}$, so there are n variables in the vector $\mathbf{x}$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^{m\times 1}$, $\mathbf{H} \in \mathbb{R}^{l\times n}$ and $\mathbf{r} \in \mathbb{R}^{l\times 1}$. The vectors $\mathbf{l}$ and $\mathbf{u}$ are lower and upper bounds respectively on the decision variables in the vector $\mathbf{x}$.

The Matlab command for solving linear programs is `linprog` and it takes the parameters:

1. $\mathbf{c}$,
2. $\mathbf{A}$,
3. $\mathbf{b}$,
4. $\mathbf{H}$,
5. $\mathbf{r}$,
6. $\mathbf{l}$,
7. $\mathbf{u}$

If there are no inequality constraints, then we set $\mathbf{A} = []$ and $\mathbf{b} = []$ in Matlab; i.e., $\mathbf{A}$ and $\mathbf{b}$ are set as the *empty matrices*. A similar requirement holds on $\mathbf{H}$ and $\mathbf{r}$ if there are no equality constraints. If some decision variables have lower bounds and others don't, the term `-inf` can be used to set a lower bound at $-\infty$ (in $\mathbf{l}$). Similarly, the term `inf` can be used if the upper bound on a variable (in $\mathbf{u}$) is infinity. The easiest way to understand how to use Matlab is to use it on an example.

Example 7.25. Suppose I wish to design a diet consisting of Raman noodles and ice cream. I'm interested in spending as little money as possible but I want to ensure that I eat at least 1200 calories per day and that I get at least 20 grams of protein per day. Assume that each serving of Raman costs \$1 and contains 100 calories and 2 grams of protein. Assume that each serving of ice cream costs \$1.50 and contains 200 calories and 3 grams of protein.

We can construct a linear programming problem out of this scenario. Let x_1 be the amount of Raman we consume and let x_2 be the amount of ice cream we consume. Our objective function is our cost:

$$x_1 + 1.5x_2 \tag{7.17}$$

Our constraints describe our protein requirements:

$$2x_1 + 3x_2 \geq 20 \tag{7.18}$$

and our calorie requirements (expressed in terms of 100's of calories):

$$x_1 + 2x_2 \geq 12 \tag{7.19}$$

This leads to the following linear programming problem:

$$\begin{cases} \min & x_1 + 1.5x_2 \\ s.t. & 2x_1 + 3x_2 \geq 20 \\ & x_1 + 2x_2 \geq 12 \\ & x_1, x_2 \geq 0 \end{cases} \tag{7.20}$$

Let's use Matlab to solve this problem. Our original problem is:

$$\begin{cases} \min & x_1 + 1.5x_2 \\ s.t. & 2x_1 + 3x_2 \geq 20 \\ & x_1 + 2x_2 \geq 12 \\ & x_1, x_2 \geq 0 \end{cases}$$

This is not in a form Matlab likes, so we change it by multiplying the constraints by -1 on both sides to obtain:

$$\begin{cases} \min & x_1 + 1.5x_2 \\ s.t. & -2x_1 - 3x_2 \leq -20 \\ & -x_1 - 2x_2 \leq -12 \\ & x_1, x_2 \geq 0 \end{cases}$$

Then we have:

$$\mathbf{c} = \begin{bmatrix} 1 \\ 1.5 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} -2 & -3 \\ -1 & -2 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} -20 \\ -12 \end{bmatrix}$$

$$\mathbf{H} = \mathbf{r} = []$$

$$\mathbf{l} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \mathbf{u} = []$$

The Matlab code to solve this problem is shown in Figure 7.3 The solution Matlab returns in the $\mathbf{x}$ variable is $x_1 = 3.7184$ and $x_2 = 4.1877$. It turns out there are actually an infinite number of alternative optimal solutions to this problem. You could draw a picture of this scenario to see if you can figure out why.

EXERCISE 65. In previous example, you could also have just used the problem in standard form with the surplus variables and had $\mathbf{A} = \mathbf{b} = []$ and defined $\mathbf{H}$ and $\mathbf{r}$ instead. Use Matlab to solve the diet problem in *standard form.* Compare your results to Example 7.25

```
%%Solve the Diet Linear Programming Problem
c = [1 1.5]';
A = [[-2 -3];...
     [-1 -2]];
b = [-20 -12]';
H = [];
r = [];
l = [0 0]';
u = [];
[x obj] = linprog(c,A,b,H,r,l,u);
```

Figure 7.3. Matlab input for solving the diet problem. Note that we are solving a *minimization* problem. Matlab assumes all problems are *mnimization* problems, so we don't need to multiply the objective by -1 like we would if we started with a maximization problem.

5. Kurush-Kuhn-Tucker (KKT) Conditions

REMARK 7.26. The single most important thing to learn about Linear Programming (or optimization in general) is the Kurush-Kuhn-Tucker optimality conditions. These conditions provide necessary and sufficient conditions for a point $\mathbf{x} \in \mathbb{R}^n$ to be an optimal solution to a Linear Programming problem. We state the Kurush-Kuhn-Tucker theorem, but do not prove it. A proof can be found in Chapter 8 of [Gri11b].

THEOREM 7.27. *Consider the linear programming problem:*

$$P\begin{cases}\max & \mathbf{cx}\\ s.t. & \mathbf{Ax} \leq \mathbf{b}\\ & \mathbf{x} \geq \mathbf{0}\end{cases} \tag{7.21}$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ *and (row vector)* $\mathbf{c} \in \mathbb{R}^n$. *Then* $\mathbf{x}^* \in \mathbb{R}^n$ *if and only if there exists (row) vectors* $\mathbf{w}^* \in \mathbb{R}^m$ *and* $\mathbf{v}^* \in \mathbb{R}^n$ *and a slack variable vector* $\mathbf{s}^* \in \mathbf{R}^m$ *so that:*

$$\text{Primal Feasibility}\begin{cases}\mathbf{Ax}^* + \mathbf{s}^* = \mathbf{b}\\ \mathbf{x}^* \geq \mathbf{0}\end{cases} \tag{7.22}$$

$$\text{Dual Feasibility}\begin{cases}\mathbf{w}^*\mathbf{A} - \mathbf{v}^* = \mathbf{c}\\ \mathbf{w}^* \geq \mathbf{0}\\ \mathbf{v}^* \geq \mathbf{0}\end{cases} \tag{7.23}$$

$$\text{Complementary Slackness}\begin{cases}\mathbf{w}^*\left(\mathbf{Ax}^* - \mathbf{b}\right) = 0\\ \mathbf{v}^*\mathbf{x}^* = 0\end{cases} \tag{7.24}$$

REMARK 7.28. The vectors $\mathbf{w}^*$ and $\mathbf{v}^*$ are sometimes called *dual variables* for reasons that will be clear in the next chapter. They are also sometimes called *Lagrange Multipliers.* You may have encountered Lagrange Multipliers in your Math 230 or Math 231 class. These are the same kind of variables except applied to linear optimization problems. There is one element in the dual variable vector $\mathbf{w}^*$ for each constraint of the form $\mathbf{Ax} \leq \mathbf{b}$ and one element in the dual variable vector $\mathbf{v}^*$ for each constraint of the form $\mathbf{x} \geq \mathbf{0}$.

Example 7.29. Consider the Toy Maker Problem (Equation 7.9) with Dual Variables (Lagrange Multipliers) listed next to their corresponding constraints:

$$\begin{cases} \max & z(x_1, x_2) = 7x_1 + 6x_2 & \textbf{Dual Variable} \\ s.t. & 3x_1 + x_2 \leq 120 & (w_1) \\ & x_1 + 2x_2 \leq 160 & (w_1) \\ & x_1 \leq 35 & (w_3) \\ & x_1 \geq 0 & (v_1) \\ & x_2 \geq 0 & (v_2) \end{cases}$$

In this problem we have:

$$\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 7 & 6 \end{bmatrix}$$

Then the KKT conditions can be written as:

$$\text{Primal Feasibility} \begin{cases} \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{cases}$$

$$\text{Dual Feasibility} \begin{cases} \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} v_1 & v_2 \end{bmatrix} = \begin{bmatrix} 7 & 6 \end{bmatrix} \\ \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \geq \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} v_1 & v_2 \end{bmatrix} \geq \begin{bmatrix} 0 & 0 \end{bmatrix} \end{cases}$$

$$\text{Complementary Slackness} \begin{cases} \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \left(\begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} - \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix} \right) \\ \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} = 0 \end{cases}$$

Recall that at optimality, we had $x_1 = 16$ and $x_2 = 72$. The binding constraints in this case where

$$3x_1 + x_2 \leq 120$$
$$x_1 + 2x_2 \leq 160$$

To see this note that if $3(16) + 72 = 120$ and $16 + 2(72) = 160$. Then we should be able to express $\mathbf{c} = [7 \quad 6]$ (the vector of coefficients of the objective function) as a positive combination of the gradients of the binding constraints:

$$\nabla(7x_1 + 6x_2) = \begin{bmatrix} 7 & 6 \end{bmatrix}$$
$$\nabla(3x_1 + x_2) = \begin{bmatrix} 3 & 1 \end{bmatrix}$$
$$\nabla(x_1 + 2x_2) = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

That is, we wish to solve the linear equation:

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 6 \end{bmatrix} \tag{7.25}$$

The result is the system of equations:

$$\begin{aligned} 3w_1 + w_2 &= 7 \\ w_1 + 2w_2 &= 6 \end{aligned}$$

A solution to this system is $w_1 = \frac{8}{5}$ and $w_2 = \frac{11}{5}$. This fact is illustrated in Figure 7.4.

Figure 7.4 shows the gradient cone formed by the binding constraints at the optimal point for the toy maker problem. Since $x_1, x_2 > 0$, we must have $v_1 = v_2 = 0$. Moreover,

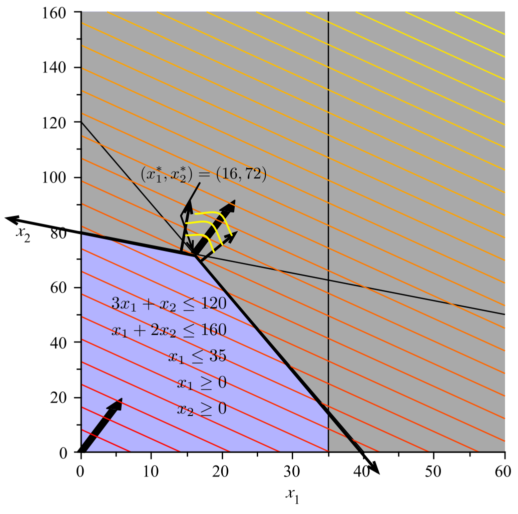


Figure 7.4. The Gradient Cone: At optimality, the cost vector **c** is obtuse with respect to the directions formed by the binding constraints. It is also contained inside the cone of the gradients of the binding constraints, which we will discuss at length later.

since $x_1 < 35$, we know that $x_1 \leq 35$ is not a binding constraint and thus its dual variable w_3 is also zero. This leads to the conclusion:

$$\begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} 16 \\ 72 \end{bmatrix} \quad \begin{bmatrix} w_1^* & w_2^* & w_3^* \end{bmatrix} = \begin{bmatrix} 8/5 & 11/5 & 0 \end{bmatrix} \quad \begin{bmatrix} v_1^* & v_2^* \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

and the KKT conditions are satisfied.

Exercise 66. Consider the problem:

$$\begin{aligned} \max \quad & x_1 + x_2 \\ s.t. \quad & 2x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Write the KKT conditions for an optimal point for this problem. (You will have a vector $\mathbf{w} = [w_1 \quad w_2]$ and a vector $\mathbf{v} = [v_1 \quad v_2]$).

Draw the feasible region of the problem and use Matlab to solve the problem. At the point of optimality, identify the binding constraints and draw their gradients. Show that the KKT conditions hold. (Specifically find $\mathbf{w}$ and $\mathbf{v}$.)

EXERCISE 67. Find the KKT conditions for the problem:

$$\begin{cases} \min & \mathbf{cx} \\ s.t. & \mathbf{Ax} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{cases} \tag{7.26}$$

[Hint: Remember, every minimization problem can be converted to a maximization problem by multiplying the objective function by -1 and the constraints $\mathbf{Ax} \geq \mathbf{b}$ are equivalent to the constraints $-\mathbf{Ax} \leq -\mathbf{b}$.

6. Duality

REMARK 7.30. In this section, we show that to each linear programming problem (the primal problem) we may associate another linear programming problem (the dual linear programming problem). These two problems are closely related to each other and an analysis of the dual problem can provide deep insight into the primal problem.

Consider the linear programming problem

$$P \begin{cases} \max & \mathbf{c}^T\mathbf{x} \\ s.t. & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{cases} \tag{7.27}$$

Then the dual problem for Problem P is:

$$D \begin{cases} \min & \mathbf{wb} \\ s.t. & \mathbf{wA} \geq \mathbf{c} \\ & \mathbf{w} \geq \mathbf{0} \end{cases} \tag{7.28}$$

REMARK 7.31. Let $\mathbf{v}$ be a vector of *surplus* variables. Then we can transform Problem D into standard form as:

$$D_S \begin{cases} \min & \mathbf{wb} \\ s.t. & \mathbf{wA} - \mathbf{v} = \mathbf{c} \\ & \mathbf{w} \geq \mathbf{0} \\ & \mathbf{v} \geq \mathbf{0} \end{cases} \tag{7.29}$$

Thus we already see an intimate relationship between duality and the KKT conditions. The feasible region of the dual problem (in standard form) is precisely the the dual feasibility constraints of the KKT conditions for the primal problem.

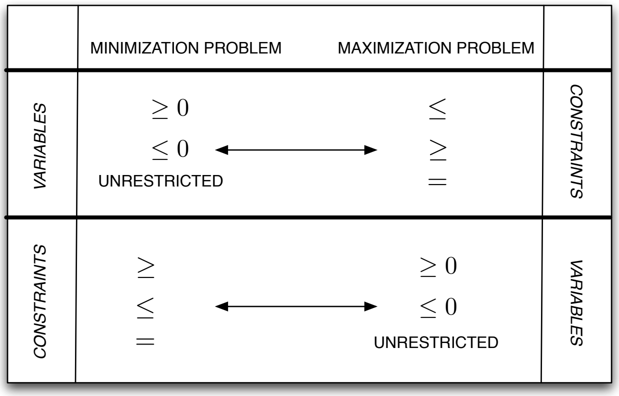
In this formulation, we see that we have assigned a dual variable w_i ($i = 1, \ldots, m$) to each constraint in the system of equations $\mathbf{Ax} \leq \mathbf{b}$ of the primal problem. Likewise dual variables $\mathbf{v}$ can be thought of as corresponding to the constraints in $\mathbf{x} \geq \mathbf{0}$.

REMARK 7.32. The proof of the following lemma is outside the scope of the class, but it establishes an important fact about duality.

Lemma 7.33. *The dual of the dual problem is the primal problem.* □

Remark 7.34. Lemma 7.33 shows that the notion of dual and primal can be exchanged and that it is simply a matter of perspective which problem is the dual problem and which is the primal problem. Likewise, by transforming problems into canonical form, we can develop dual problems for any linear programming problem.

The process of developing these formulations can be exceptionally tedious, as it requires enumeration of all the possible combinations of various linear and variable constraints. The following table summarizes the process of converting an arbitrary primal problem into its dual. This table can be found in Chapter 6 of [BJS04].



	MINIMIZATION PROBLEM	MAXIMIZATION PROBLEM	
VARIABLES	≥ 0	$\leq$	CONSTRAINTS
	≤ 0	$\geq$	
	UNRESTRICTED	$=$	
CONSTRAINTS	$\geq$	≥ 0	VARIABLES
	$\leq$	≤ 0	
	$=$	UNRESTRICTED	

Table 1. Table of Dual Conversions: To create a dual problem, assign a dual variable to each constraint of the form $\mathbf{Ax} \circ \mathbf{b}$, where $\circ$ represents a binary relation. Then use the table to determine the appropriate sign of the inequality in the dual problem as well as the nature of the dual variables.

Example 7.35. Consider the problem of finding the dual problem for the Toy Maker Problem (Example 7.5) in standard form. The primal problem is:

$$\begin{aligned}
\max\;\; & 7x_1 + 6x_2 & \\
s.t.\;\; & 3x_1 + x_2 + s_1 = 120 & \quad (w_1) \\
& x_1 + 2x_2 + s_2 = 160 & \quad (w_2) \\
& x_1 + s_3 = 35 & \quad (w_3) \\
& x_1, x_2, s_1, s_2, s_3 \geq 0 &
\end{aligned}$$

Here we have placed dual variable names (w_1, w_2 and w_3) next to the constraints to which they correspond.

The primal problem variables in this case are all positive, so using Table 1 we know that the constraints of the dual problem will be greater-than-or-equal-to constraints. Likewise, we know that the dual variables will be unrestricted in sign since the primal problem constraints are all equality constraints.

The coefficient matrix is:

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

Clearly we have:

$$\mathbf{c} = \begin{bmatrix} 7 & 6 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{b} = \begin{bmatrix} 120 \\ 160 \\ 35 \end{bmatrix}$$

Since $\mathbf{w} = [w_1 \;\; w_2 \;\; w_3]$, we know that $\mathbf{wA}$ will be:

$$\mathbf{wA} = \begin{bmatrix} 3w_1 + w_2 + w_3 & w_1 + 2w_2 & w_1 & w_2 & w_3 \end{bmatrix}$$

This vector will be related to $\mathbf{c}$ in the constraints of the dual problem. **Remember, in this case, all variables in the primal problem are greater-than-or-equal-to zero**. Thus we see that the constraints of the dual problem are:

$$\begin{aligned} 3w_1 + w_2 + w_3 &\geq 7 \\ w_1 + 2w_2 &\geq 6 \\ w_1 &\geq 0 \\ w_2 &\geq 0 \\ w_3 &\geq 0 \end{aligned}$$

We also have the redundant set of constraints that tell us $\mathbf{w}$ is unrestricted because the primal problem had equality constraints. This *will always* happen in cases when you've introduced slack variables into a problem to put it in standard form. This should be clear from the definition of the dual problem for a maximization problem in canonical form.

Thus the whole dual problem becomes:

$$(7.30) \qquad \begin{aligned} \min \;\; & 120w_1 + 160w_2 + 35w_3 \\ s.t. \;\; & 3w_1 + w_2 + w_3 \geq 7 \\ & w_1 + 2w_2 \geq 6 \\ & w_1 \geq 0 \\ & w_2 \geq 0 \\ & w_3 \geq 0 \\ & \mathbf{w} \quad \text{unrestricted} \end{aligned}$$

Again, note that in reality, the constraints we derived from the $\mathbf{wA} \geq \mathbf{c}$ part of the dual problem make the constraints "$\mathbf{w}$ unrestricted" redundant, for in fact $\mathbf{w} \geq \mathbf{0}$ just as we would expect it to be if we'd found the dual of the Toy Maker problem given in canonical form.

EXERCISE 68. Identify the dual problem for:

$$\begin{aligned} \max\ & x_1 + x_2 \\ s.t.\ & 2x_1 + x_2 \geq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

EXERCISE 69. Use the table or the definition of duality to determine the dual for the problem:

$$\begin{cases} \min\ \mathbf{cx} \\ s.t.\ \mathbf{Ax} \geq \mathbf{b} \\ \quad\ \ \mathbf{x} \geq \mathbf{0} \end{cases} \tag{7.31}$$

REMARK 7.36. The following theorems are outside the scope of this course, but they can be useful to know and will help cement your understanding of the true nature of duality.

THEOREM 7.37 (Strong Duality Theorem). *Consider Problem P and Problem D. Then*

(Weak Duality): $\mathbf{cx}^* \leq \mathbf{w}^*\mathbf{b}$, *thus every feasible solution to the primal problem provides a lower bound for the dual and every feasible solution to the dual problem provides an upper bound to the primal problem.*

Furthermore exactly one of the following statements is true:

(1) *Both Problem P and Problem D possess optimal solutions $\mathbf{x}^*$ and $\mathbf{w}^*$ respectively and $\mathbf{cx}^* = \mathbf{w}^*\mathbf{b}$.*
(2) *Problem P is unbounded and Problem D is infeasible.*
(3) *Problem D is unbounded and Problem P is infeasible.*
(4) *Both problems are infeasible.*

THEOREM 7.38. *Problem D has an optimal solution $\mathbf{w}^* \in \mathbb{R}^m$ if and only if there exists vectors $\mathbf{x}^* \in \mathbb{R}^n$ and $\mathbf{s}^* \in \mathbb{R}^m$ and a vector of surplus variables $\mathbf{v}^* \in \mathbb{R}^n$ such that:*

$$\textit{Primal Feasibility} \begin{cases} \mathbf{w}^*\mathbf{A} \geq \mathbf{c} \\ \mathbf{w}^* \geq 0 \end{cases} \tag{7.32}$$

$$\textit{Dual Feasibility} \begin{cases} \mathbf{Ax}^* + \mathbf{s}^* = \mathbf{b} \\ \mathbf{x}^* \geq 0 \\ \mathbf{s}^* \geq 0 \end{cases} \tag{7.33}$$

$$\textit{Complementary Slackness} \begin{cases} (\mathbf{w}^*\mathbf{A} - \mathbf{c})\,\mathbf{x}^* = 0 \\ \mathbf{w}^*\mathbf{s}^* = 0 \end{cases} \tag{7.34}$$

Furthermore, these KKT conditions are equivalent to the KKT conditions for the primal problem.

REMARK 7.39. The final theorem illustrates the true nature of duality. Two linear programming problems are dual if they share KKT conditions. That is, if they share conditions for optimality.

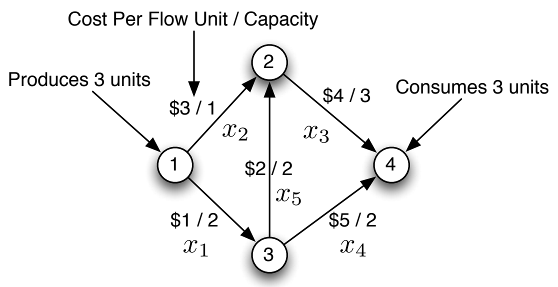


Figure 7.5. In this problem, it costs a certain amount to ship a commodity along each edge and each edge has a capacity. The objective is to find an allocation of capacity to each edge so that the total cost of shipping three units of this commodity from Vertex 1 to Vertex 4 is minimized.

Exercise 70. Consider the directed graph shown in Figure 7.5 The amount of flow along each edge is given by the variables $x_1, \dots, x_5$. The total cost of shipping flow from Vertex 1 to Vertex 5 is

$$\sum_{i=1}^{5} c_i x_i \tag{7.35}$$

where c_i is the cost associated to the flow in each edge. For each edge, we know that $x_i \geq 0$ and $x_i \leq u_i$ where u_i is the capacity on each edge. Finally, we must be able to assert that commodities are neither created nor destroyed (except at Vertex 1, where 3 units of commodity are created and at Vertex 4 where 3 units of commodity are consumed). Thus we have constraints of the form:

$$x_1 + x_2 = 3$$
$$x_1 = x_4 + x_5$$
$$x_2 + x_5 = x_3$$
$$x_3 + x_4 = 3$$

(1) Put all these constraints together to form a linear programming problem whose solution yields a minimal cost assignment of flow to the edges.
(2) Use Matlab to find an optimal flow.
(3) Notice that each equation in the equality constraints represents the balance of flow into and out of a vertex. Rewrite each equation so that is has the form

flow-out − flow-in = flow produced at vertex − flow consumed at vertex

(4) Compute the **A** matrix that results from the equations you just constructed and compare it to the incidence matrix of the directed graph. [Hint: They should be the same.]

CHAPTER 8

An Introduction to Network Flows and Combinatorial Optimization

Remark 8.1. For the remainder of this chapter, we will consider directed graphs with *no* isolated vertices and no self-loops. That is, we will only consider those graphs whose incident matrices do not have any zero rows. These graphs will be connected and furthermore will have two special vertices v_1 and v_m and we will assume that there is at least one directed path from v_1 to v_m.

1. The Maximum Flow Problem

Definition 8.2 (Flow). Let $G = (V, E)$ be a digraph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. If $e_k = (v_i, v_j)$ is an edge, then a *flow* on e_k is a value $x_k \in \mathbb{R}_+$ that determines that amount of some quantity (of a commodity) that will leave v_i and flow along e_k to v_j.

Definition 8.3 (Vertex Supply and Demand). Let $G = (V, E)$ be a digraph and suppose $V = \{v_1, \dots, v_m\}$. The *flow supply* for vertex v_i is a real value b_i assigned to v_i that quantifies that amount of flow produced at vertex v_i. If $b_i < 0$, then vertex v_i consumes flow (rather than producing it).

Definition 8.4 (Flow Conservation Constraint). Let $G = (V, E)$ be a digraph with no self-lops and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Let $I(i)$ be the set of edges with destination vertex v_i and $O(i)$ be the set of edges with source v_i. Then the flow conservation constraint associated to vertex v_i is:

$$\sum_{k \in O(i)} x_k - \sum_{k \in I(i)} x_k = b_i \quad \forall i \tag{8.1}$$

Remark 8.5. Equation 8.1 states that the total flow out of vertex v_i minus the total flow into v_i must be equal to the total flow produced at v_i. Or put more simply, excess flow is neither created nor destroyed.

Definition 8.6 (Edge Capacity). Let $G = (V, E)$ be a digraph with no self-lops and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. If $e_k \in E$ then its capacity is a positive real value u_k that determines the maximum amount of flow the edge may be assigned. We can think of $(G, \mathbf{u})$ as being a weighed graph where the weights are the edge capacities.

Proposition 8.7. *Let $G = (V, E)$ be a digraph with no self-lops and suppose $V = \{v_1, \dots, v_n\}$ and let $\mathbf{A}$ be the incidence matrix of G (see Definition 4.63). Then Equation 8.1 can be written as:*

$$\mathbf{A}_{i\cdot}\mathbf{x} = b_i \tag{8.2}$$

where $\mathbf{x}$ *is a vector of variables of the form* x_k *taken in the order the edges are represented in* $\mathbf{A}$*.*

PROOF. From Definition 4.63 we know that:

$$\mathbf{A}_{ik} = \begin{cases} 0 & \text{if } v_i \text{ is not in } e_k \\ 1 & \text{if } v_i \text{ is the source of } e_k \\ -1 & \text{if } v_i \text{ is the destination of } e_k \end{cases} \tag{8.3}$$

The equivalence between Equation 8.2 and Equation 8.1 follows at once from this fact. □

REMARK 8.8. For the remainder of this chapter, Let $\mathbf{e}_1 \in \mathbb{R}^{m\times 1}$ be the vector with a 1 at position 1 and 0 everywhere else. Define $\mathbf{e}_m$ similarly.

DEFINITION 8.9 (Maximum Flow Problem). Let $G = (V, E)$ be a digraph with no self-loops and suppose that $V = \{v_1, \dots, v_m\}$. Without loss of generality, suppose that there is no edge connecting v_m to v_1. The *maximum flow problem* for G is the linear programming problem:

$$\begin{cases} \max\ f \\ s.t.\ \ (\mathbf{e}_m - \mathbf{e}_1) f + \mathbf{A}\mathbf{x} = \mathbf{0} \\ \qquad \mathbf{x} \leq \mathbf{u} \\ \qquad \mathbf{x} \geq \mathbf{0} \\ \qquad f \text{ unrestricted} \end{cases} \tag{8.4}$$

Here $\mathbf{u}$ is a vector of edge flow capacity values.

REMARK 8.10. The constraints $(\mathbf{e}_m - \mathbf{e}_1) f + \mathbf{A}\mathbf{x} = \mathbf{0}$ are flow conservation constraints when we assume that there is an (imaginary) flow backwards from v_m to v_1 along an edge (v_m, v_1) and that no flow is produced in the graph. That is, we assume all flow is circulating within the graph. The value f determines the amount of flow that circulates back to vertex v_1 from v_m under this assumption. Since all flows are circulating and excess flow is neither created nor destroyed, the value of f is then the total flow that flows from v_1 to v_m. By maximizing f, Problem 8.4 is exactly computing the maximum amount of flow that can go from vertex v_1 to v_m under the assumptions that flows are constrained by edge capacities ($\mathbf{x} \leq \mathbf{u}$), flows are non-negative ($\mathbf{x} \geq \mathbf{0}$) and flows are neither created nor destroyed in the graph.

2. The Dual of the Flow Maximization Problem

THEOREM 8.11. *The dual linear programming problem for Problem 8.4 is:*

$$\begin{cases} \min\ \sum_{k=1}^{n} u_k h_k \\ s.t.\ \ w_m - w_1 = 1 \\ \qquad w_i - w_j + h_k \geq 0 \quad \forall\, e_k = (v_i, v_j) \in E \\ \qquad h_k \geq 0 \quad \forall\, (v_i, v_j) \in E \\ \qquad w_i \textit{ unrestricted} \quad \forall i \in \{1, \dots, m\} \end{cases} \tag{8.5}$$

Proof. Consider the constraints of Problem 8.4 and suppose that the imaginary edge from v_m to v_1 is edge e_0. We first add slack variables to constraints of the form $x_k \leq u_k$ to obtain:

$$x_k + s_k = u_k \quad \forall k \in \{1, \dots, n\}$$

The constraints (other than $\mathbf{x} \geq \mathbf{0}$ and f unrestricted) can be rewritten in matrix form as:

$$(8.6) \qquad \begin{bmatrix} -1 & a_{11} & a_{12} & \cdots & a_{1n} & 0 & 0 & \cdots & 0 \\ 0 & a_{12} & a_{22} & \cdots & a_{2n} & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & a_{m1} & a_{m2} & \cdots & a_{mn} & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} f \\ x_1 \\ x_2 \\ \vdots \\ x_n \\ s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

or more simply as:

$$(8.7) \qquad \begin{bmatrix} \mathbf{e}_m - \mathbf{e}_1 & \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_n & \mathbf{I}_n \end{bmatrix} \begin{bmatrix} f \\ \mathbf{x} \\ \mathbf{s} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{u} \end{bmatrix}$$

where all elements written $\mathbf{0}$ are zero matrices or vectors of appropriate dimension. This matrix has $2n + 1$ columns and $m + n$ rows. To the first m rows, we associate the dual variables $w_1, \dots, w_m$. To the next n rows we associate the dual variables $h_1, \dots, h_n$. Our dual variable vector is then:

$$\mathbf{y} = [w_1, \dots, w_m, h_1, \dots, h_n]$$

Since the constraints in Expression 8.6 are all equality constraints, we know that there dual variables are unrestricted. Further from Expression 8.6 we know that the objective function vector is:

$$\mathbf{c} = [1, 0, \dots, 0]$$

We can now compute our dual constraints. The first constraint is computed from $\mathbf{y}$ multiplied by the first column of the matrix in Expression 8.6 and gives:

$$(8.8) \qquad -w_1 + w_m = 1$$

Since this dual constraint corresponds to the variable f in the primal problem we know it will be an equality constraint (f is unrestricted) and that its right hand side will be 1 the coefficient of f in the primal problem. The next n constraints are derived by multiplying $\mathbf{y}$ by the next n columns of the matrix in Expression 8.6 and will have the form:

$$(8.9) \qquad w_i - w_j + h_k \geq 0$$

This follows since there will be a -1 in the matrix whenever edge e_k has as destination vertex v_j and a $+1$ in the matrix whenever edge e_k has source at vertex v_i. Clearly there is a 1 in the k^{th} row of the identity matrix below the $\mathbf{A}$ matrix in Expression 8.6 thus yielding the $+h_k$

term. These constraints correspond to the variables $x_1, \dots, x_n$, which are all non-negative and thus the constraints are non-negative. The final n constraints have form:

$$h_k \geq 0 \tag{8.10}$$

and are derived by multiplying $\mathbf{y}$ by the last n columns of the matrix in Expression 8.6. These constraints correspond to the variables $s_1, \dots, s_n$ which are non-negative and thus the constraints are non-negative. The objective function of the dual problem is computed by multiplying $\mathbf{y}$ by the right-hand-side of Expression 8.6. This yields:

$$\sum_{k=1}^{n} u_k h_k \tag{8.11}$$

Problem 8.5 follows at once. This completes the proof. □

3. The Max-Flow / Min-Cut Theorem

Remark 8.12. Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Recall we defined an edge cut in Definition 3.39. Let V_1 be any set of vertices containing v_1 and not containing v_m and let $V_2 = V \setminus v_1$. Immediately we see $v_m \in V_2$. Clearly, the edges connecting vertices in V_1 with vertices in V_2 form an edge cut and moreover any edge cut that divides G into two components, one containing v_1 and the other containing v_m corresponds to some sets V_1 and V_2. Thus, we will refer to all such edge cuts by these generated sets; that is, (V_1, V_2) corresponds to the edge cut defined when $v_1 \in V_1$ and $v_m \in V_2$ and $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$. For the remainder of this chapter, a cut refers to a cut of this type.

Definition 8.13 (Cut Capacity). Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Let (V_1, V_2) be a cut separating v_1 from v_m containing edges $e_{s_1}, \dots, e_{s_l}$ with sources in V_1 and destinations in V_2, where $s_1, \dots, s_l$ is a subset of the edge indexes $1, \dots, n$. Then the *capacity of the cut* (V_1, V_2) is:

$$C(V_1, V_2) = \sum_{k=1}^{l} u_{s_k} \tag{8.12}$$

Lemma 8.14. *Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. The solution to the maximum flow problem is bounded above by the minimal cut capacity.*

Proof. Let (V_1, V_2) be the cut with minimal capacity. Consider the following solution to the dual problem:

$$w_i^* = \begin{cases} 0 & v_i \in V_1 \\ 1 & v_i \in V_2 \end{cases} \tag{8.13}$$

and

$$h_k^* = \begin{cases} 1 & e_k = (v_i, v_j) \text{ and } v_i \in V_1 \text{ and } v_j \in V_2 \\ 0 & \text{else} \end{cases} \tag{8.14}$$

It is clear this represents a feasible solution to the dual problem. Thus by the strong duality theorem (Theorem 7.37) the objective function value:

$$\sum_k u_k h_k \tag{8.15}$$

is an upper bound for the primal problem. But this is just the capacity of the cut with the smallest capacity. This completes the proof. □

LEMMA 8.15. *In any optimal solution to Problem 8.4 every directed path from v_1 to v_m must have at least one edge at capacity.*

PROOF. Note first, problem 8.4 is bounded above by the capacity of the minimal cut as shown in Lemma 8.14 and since the zero flow is a feasible solution, we know from Theorem 7.22 there is at least one optimal solution to Problem 8.4 because the problem can neither be unbounded nor infeasible.

Consider any optimal solution to Problem 8.5. Then it corresponds to some optimal solution to the primal problem and these solutions satisfy the Kurush-Kuhn-Tucker conditions. We show that in this primal solution, along each path from v_1 to v_m in G at least one edge must have flow equal to its capacity. To see this note that for any edge that does not carry its capacity (that is $x_k < u_k$) we must have $h_k = 0$ (to ensure complementary slackness). Suppose this path has vertices $(u_1, v_2, \dots, u_s)$ with $v_1 = u_1$ and $v_m = u_s$. If there is some path from v_1 to v_m that does not carry its capacity, then we have the following requirements:

$$\begin{aligned} & w_s > w_1 \\ & w_1 \geq w_2 \\ & \vdots \\ & w_{s-1} \geq w_s \end{aligned}$$

But this implies that $w_s > w_1 \geq w_2 \geq \cdots \geq w_s$, which is a contradiction. Therefore every path from v_1 to v_m has at least one edge at capacity. □

THEOREM 8.16. *Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. There is at least one cut (V_1, V_2) so that the flow from v_1 to v_m is equal to the capacity of the cut (V_1, V_2).*

PROOF. Denote one such solution to Problem 8.4 as $(\mathbf{x}^*, f^*)$; we now know such a solution exists. By Lemma 8.15, we know that in this solution every directed path from v_1 to v_m must have at least one edge at capacity. From each path from v_1 to v_m select an edge that is at capacity in such a way that we minimize the total sum of the capacities of the chosen edges. Denote this set of edges E'.

If E' is not yet an edge cut in the underlying graph of G, then there are some paths from v_1 to v_m in the underlying graph of G that are *not* directed paths from v_1 to v_m. In each such path, there is at least one edge directed toward v_m from v_1. Choose one edge from each of these paths directed from v_m to v_1 to minimize the total cardinality of edges chosen and add these edges to E'. (See Figure 8.1).

Define V_1 and V_2 as follows:

$$V_1 = \{v : \text{there is a simple path from } v_1 \text{ to } v \text{ in the underlying graph of } G - E'\}$$

$$V_2 = \{v : \text{there is a path from } v \text{ to } v_m \text{ in the underlying graph of } G - E'\}$$

This construction is illustrated in Figure 8.1.

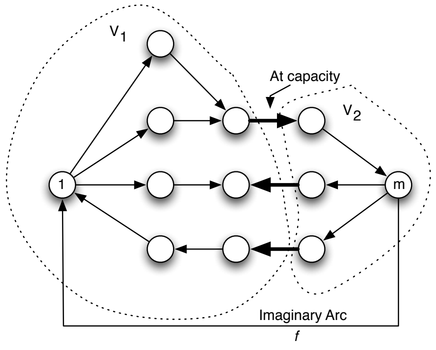


Figure 8.1. A cut is defined as follows: in each directed path from v_1 to v_m, we choose an edge at capacity so that the collection of chosen edges has minimum capacity (and flow). If this set of edges is not an edge cut of the underlying graph, we add edges that are directed from v_m to v_1 in a simple path from v_1 to v_m in the underlying graph of G.

Claim 1. *Every vertex is either in V_1 or V_2 using the definition of E' and thus the set $E' = (V_1, V_2)$ is an edge cut in the underlying graph of G.*

Proof. See Exercise 71. □

Suppose $E' = \{e_{s_1}, \dots, e_{s_l}\}$.

Claim 2. *If there is some edge e_k with source in V_2 and destination in V_1, then $x_k = 0$.*

Proof. If $x_k \neq 0$, we could reduce this flow to zero and increase the net flow from v_1 to v_m by adding this flow to f. If flow cannot reach x_1 along e_k (illustrated by the middle path in Figure 8.1, then flow conservation ensures it must be equal to zero. □

Claim 3. *The total flow from v_1 to v_m must be equal to the capacity of the edges in E' that have source in V_1 and destination in V_2.*

Proof. We've established that if there is some edge e_k with source in V_2 and destination in V_1, then $x_k = 0$. Thus, the flow from v_1 to v_m must all traverse edges leaving V_1 and entering V_2. Thus, the flow from v_1 to v_m must be equal to the capacity of the cut $E' = (V_1, V_2)$. □

Claim 3 establishes that the flow f^* must be equal to the capacity of a cut (V_1, V_2). This completes the proof of the theorem. □

Corollary 8.17 (Max Flow / Min Cut Theorem). *Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Then the maximum flow from v_1 to v_m is equal to the capacity of the minimum cut separating v_1 from v_m.*

Proof. By Theorem 8.16, if $(\mathbf{x}^*, f^*)$ is a maximum flow in G from v_1 to v_m, then there is a cut (V_1, V_2) so that the capacity of this cut is equal to f^*. Since f^* is bounded above by the capacity of the minimal cut separating v_1 from v_m, the cut constructed in the proof of Theorem 8.16 *must* be a minimal capacity cut. Thus, the maximum flow from v_1 to v_m is equal to the capacity of the minimum cut separating v_1 from v_m. □

Exercise 71. Prove Claim 1 in the proof of the Theorem 8.16.

4. An Algorithm for Finding Optimal Flow

Remark 8.18. The proof of the Max Flow / Min Cut theorem we presented is a very non-standard proof technique. Most techniques are constructive; that is, they specify an algorithm for generating a maximum flow and then show that this maximum flow must be equal to the capacity of the minimal cut. In this section, we'll develop this algorithm and show that it generates a maximum flow and then (as a result of the Max Flow / Min Cut theorem this maximum flow must be equal to the capacity of the minimum cut.)

Definition 8.19 (Augment). Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$ and let $\mathbf{x}$ be a feasible flow in G. Consider a *simple path* $p = (v_1, e_1, \dots, e_l, v_m)$ in the underlying graph of G from v_1 to v_m. The augment of p is the quantity:

$$\min_{k \in \{1,\dots,l\}} \begin{cases} u_k - x_k & \text{if the edge } e_k \text{ is directed toward } v_m \\ x_k & \text{else} \end{cases} \tag{8.16}$$

Definition 8.20 (Augmenting Path). Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$ and let $\mathbf{x}$ be a feasible flow in G. A simple path p in in the underlying graph of G from v_1 to v_m is an *augmenting path* if its augment is non-zero. In this case we say that flow $\mathbf{x}$ has an augmenting path.

Example 8.21. An example of augmenting paths is shown in Figure 8.2. An augmenting

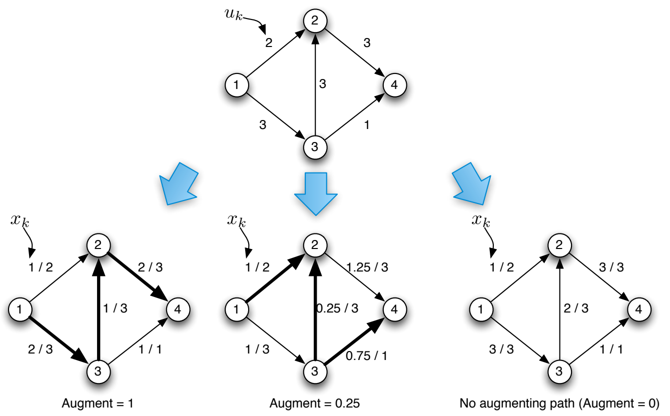


Figure 8.2. Two flows with augmenting paths and one with no augmenting paths are illustrated.

path is simply an indicator that more flow can be pushed from vertex v_1 to vertex v_m. For example, in the flow on the bottom left of Figure 8.2 we could add an additional unit of flow on the edge (v_1, v_3). This one unit could flow along edge (v_3, v_2) and then along edge (v_2, v_4). Augmenting paths that are augmenting soley because of a backward flow away from v_1 to v_m can also be used to increase the net flow from v_1 to v_m by removing flow along the backward edge.

Definition 8.22 (Path Augment). If p is an augmenting path in G with augment Δ, then by *augmenting* p by Δ we mean add Δ to the flow in each edge directed from v_1 toward v_m and subtract Δ from the flow in each edge directed from v_m to v_1.

Example 8.23. If we augment the augmenting paths illustrated in Example 8.21, the resulting flows are illustrated in Figure 8.3.

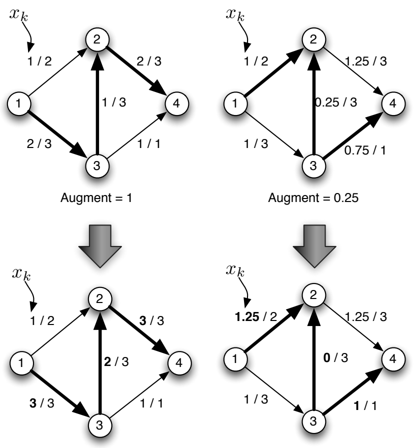


Figure 8.3. The result of augmenting the flows shown in Figure 8.2.

Remark 8.24. The next algorithm, sometimes called the Edmonds-Karp Algorithm will find a maximum flow in a network by discovering and removing all augmenting paths.

```
Maximum Flow Algorithm
Input: $(G, u)$ a weighted directed graph with $G = (V, E)$, $V = \{v_1, \dots, v_m\}$, $E = \{e_1, \dots, e_n\}$
Initizlize: $\mathbf{x} = \mathbf{0}$ {Initialize all flow variables to zero.}
    (1) Find the shortest augmenting path $p$ in $G$ using the current flow $\mathbf{x}$.
    (2) if no augmenting path exists then STOP
    (3) else augment the flow along path $p$ to produce a new flow $\mathbf{x}$
    (4) end if
    (5) GOTO (1)
Output: $\mathbf{x}^*$
```

Algorithm 8. Maximum Flow Algorithm

Example 8.25. We illustrate an example of the Edmonds-Karp algorithm in Figure 8.4. Notice that the capacity of the minimum cut is equal to the total flow leaving Vertex 1 and flowing to Vertex 4 at the completion of the algorithm.

Remark 8.26. The Edmonds-Karp Algorithm is a specialization (and correction) to the Ford-Fulkerson algorithm, which does not specify how the augmenting paths in Line 1 are chosen.

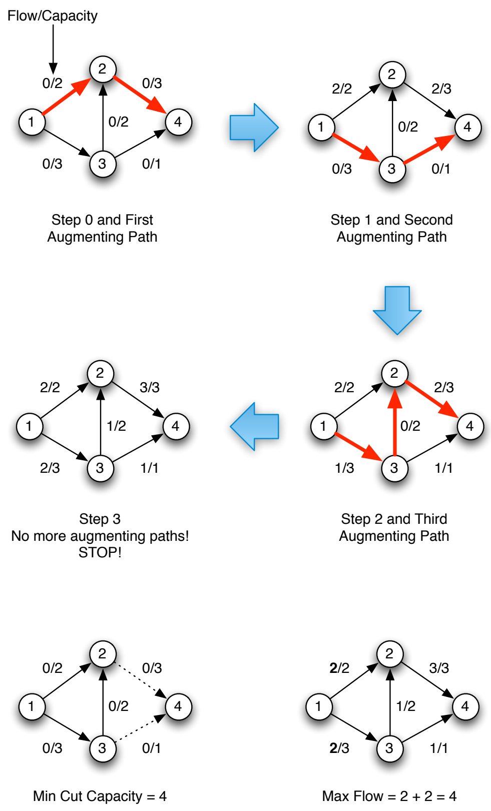


Figure 8.4. The Edmonds-Karp algorithm iteratively augments flow on a graph until no augmenting paths can be found. An initial zero-feasible flow is used to start the algorithm. Notice that the capacity of the minimum cut is equal to the total flow leaving Vertex 1 and flowing to Vertex 4.

Lemma 8.27. *Let $G = (V, E)$ be a directed graph and suppose $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$ and let $\mathbf{x}^*$ is optimal if and only if $\mathbf{x}^*$ does not have an augmenting path.*

Proof. Our proof is by abstract example. Without loss of generality, consider Figure 8.5. Suppose there is an augmenting path (as illustrated in Figure 8.5). If the flow f_1 is

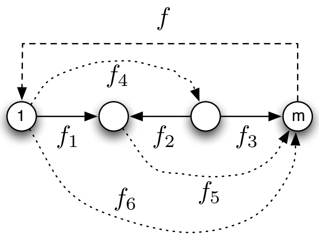


Figure 8.5. Illustration of the impact of an augmenting path on the flow from v_1 to v_m.

below capacity c_1, and this is the augment. Then we can increase the total flow along this path by increasing the flow on each edge in the direction of v_m (from v_1) by $\Delta = c_1 - f_1$ and decreasing the flow on each edge in the direction of v_1 (from v_m) by Δ. Flow conservation is preserved since we see that:

$$f_1 + f_2 - f_5 = 0 \implies (f_1 + \Delta) + (f_2 - \Delta) - f_5 = 0 \tag{8.17}$$
$$f_3 + f_2 - f_4 = 0 \implies (f_3 + \Delta) + (f_2 - \Delta) - f_4 = 0 \tag{8.18}$$

and:

$$f = f_5 + f_6 + f_3 \implies f + \Delta = f_5 + f_6 + (f_3 + \Delta) \tag{8.19}$$
$$f = f_4 + f_6 + f_1 \implies f + \Delta = f_4 + f_6 + (f_1 + \Delta) \tag{8.20}$$

The same is true if the flow on $f_2 > 0$ and this is the augment. In this case, we can increase the total flow by decreasing the flow on each edge in the direction of v_1 (from v_m) by $\Delta = f_2$ and increasing the flow on each edge in the direction of v_m (from v_1) by Δ. Thus if an augmenting path exists, the flow cannot be maximal.

Conversely, suppose we do not have a maximal flow. Then by the Max Flow / Min Cut theorem, the flow across the minimal edge cut is not equal to its capacity. Thus there is some edge in the minimal edge cut whose flow can be increased. Thus, there must be an augmenting path. This completes the proof. □

Remark 8.28. The proof of Lemma 8.27 also illustrates that Algorithm 8 maintains flow feasibility as it is executed.

Remark 8.29. The proof of the completeness of Algorithm 8 is a bit complicated. Therefore, we simply state it without offering formal proof. The interested reader should consult [**KV08**].

Theorem 8.30. *Algorithm 8 terminates in $O(mn^2)$ time.*

Theorem 8.31. *At the completion of Algorithm 8, there are no augmenting paths and the flow $\mathbf{x}^*$ is feasible.*

Proof. To see that $\mathbf{x}^*$ is feasible, note that we never increase the flow along any path by more than the maximum amount possible to ensure feasibility in all flows and a flow is never decreased beyond zero. This is ensured in our definition of augment.

To prove optimality, suppose at the completion of Algorithm 8 there was an augmenting path p. If we execute Line 1 of the algorithm, we will detect that augmenting path. Thus, no augmenting path exists at the conclusion of Algorithm 8 and by Lemma 8.27 $\mathbf{x}^*$ is optimal. □

Corollary 8.32 (Integral Flow Theorem). *If the capacities of a network are all integers, then there exists an integral maximum flow.*

Remark 8.33. It is worth noting that the original form of Algorithm 8 did not specify which augmenting path to find. This leads to a pathological condition in which the algorithm occasionally will not terminate. This is detailed in Ford and Fulkerson's original paper and more recently in [**Zwi95**]. The shortest augmenting path can be found using a breadth first search on the underlying graph. This breadth first search is what leads to the proof of Theorem 8.30.

Exercise 72. Prove the Integral Flow Theorem.

REMARK 8.34. The problem of finding a maximum flow (or minimum cut) is still very much an area of interest for researchers with new results being published as recently as the late 90's. See [**KV08**] for details.

5. Applications of the Max Flow / Min Cut Theorem

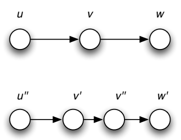
THEOREM 8.35 (Menger's First Theorem). *Let G be an (undirected) graph with $V = \{v_1, \dots, v_m\}$. Then the number of edge disjoint paths from v_1 to v_m is equal to the size of the smallest edge cut separating v_1 from v_m.*

EXERCISE 73. Prove Menger's Theorem. [Hint: Enumerate all edge disjoint paths from v_1 to v_m and replace them with directed paths from v_1 to v_m. If any edges remain un-directed, then give them arbitrary direction. Assign each arc a flow capacity of 1.]

REMARK 8.36. The next two proofs of classical theorems are based on the same results in [**KV08**].

THEOREM 8.37 (Menger's Second Theorem). *Let $G = (V, E)$ be a directed graph. Let v_1 and v_2 be two non-adjacent and distinct vertices in V. The maximum number of vertex disjoint directed paths from v_1 to v_2 is equal to the minimum number of vertices (excluded v_1 and v_2) whose deletion will destroy all directed paths from v_1 to v_2.*

PROOF. We construct a new graph by replace each vertex v in G by two vertices v' and v'' and an edge (v', v''). Each edge (v, w) is replaced by the edge (v'', w') while each edge (u, v) is replaced by (u'', v'), illustrated below.



Note that each arc of the form (v', v'') corresponds to a vertex in G. Thus edge disjoint paths in the constructed graph correspond to vertex disjoint graphs in the original graph. The result follows from Menger's First Theorem. □

DEFINITION 8.38 (Matching). A matching in a graph $G = (V, E)$ is a subset M of E such that no two edges in M share a vertex in common. A matching is *maximal* if there is no other matching in G containing it. A maximal matching is *perfect* if every vertex is adjacent to an edge in the matching.

EXAMPLE 8.39. We illustrate a maximal matching and a perfect matching in Figure 8.6.

DEFINITION 8.40. Let $G = (V, E)$ be a graph and let M be a matching in G. A vertex $v \in V$ is M-saturated if there is an edge in M adjacent to v.

REMARK 8.41. Let $G = (V, E)$ be a graph. Recall from Definition 2.66 that a vertex cover is a set of vertices $S \subseteq V$ so that every edge in E is adjacent to at least one vertex in S.

DEFINITION 8.42 (Minimal Cover). Let $G = (V, E)$ be a graph. A vertex cover is S is minimal if there is no other vertex cover S' with smaller cardinality.

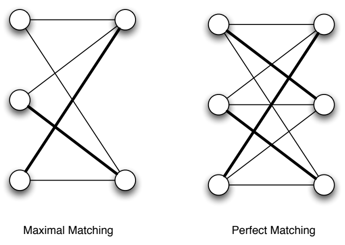
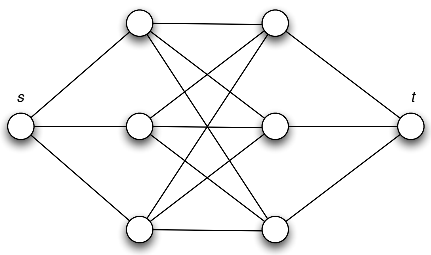


Figure 8.6. A maximal matching and a perfect matching. Note no other edges can be added to the maximal matching and the graph on the left cannot have a perfect matching.

THEOREM 8.43 (König's Theorem). *In a bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimal covering*[1]

PROOF. Let $G = (V, E)$ be the bipartite graph with $V = V_1 \cup V_2$. Let M^* be a maximal matching for G and let C^* be a minimal covering. First note that $|M^*| \leq |C^*|$. To see this note that if a matching contains every vertex (e.g. it is perfect) then a covering of G can be constructed by simply choosing (at random) one vertex from each match. The fact that G is bipartite means that each edge in G is now adjacent to some vertex in the cover constructed from M^*.

Construct a new graph N from G by introducing new vertices s and t so that s is adjacent to all vertices in V_1 and t is adjacent to all vertices in V_2. This is illustrated below.



In the remainder of the proof, s will be our source (v_1) and t will be our sink (v_m). Consider a maximal (in cardinality) set P of vertex disjoint paths from s to t (we may think of G being directed from vertices in V_1 toward vertices in V_2). Each path $p \in P$ has the form $(s, e_1, v_1, e_2, v_2, e_3, t)$ with $v_1 \in V_1$ and $v_2 \in V_2$. It is easy to see that we can construct a matching $M(P)$ from P so for path p we introduce the edge $e_2 = \{v_1, v_2\}$ into our matching $M(P)$. The fact that the paths in P are vertex disjoint implies there is a one-to-one correspondence between elements in $M(P)$ and elements in P. Thus, $|P| \leq |M^*|$ since we assumed that M^* was a maximum cardinality matching.

Now consider the smallest set $J \subset V$ whose deletion destroys all paths from s to t in N. By way of contradiction, suppose that $|J| < |C^*|$. Since we assumed that C^* was a minimal vertex cover, it follows that J is not itself a vertex cover of G and thus $G - J$ leaves at least one edge in G. But this edge must connect a vertex in V_1 to a vertex in V_2 because G is bipartite. Thus, $N - J$ has a path from s to t, which is a contradiction. Thus, $|C^*| \leq |J|$. Thus we have inequalities: $|P| \leq |M^*| \leq |C^*| \leq |J|$. But by Menger's Second

[1]Thanks to S. Shekhar for pointing out a weakness in the original proof I used.

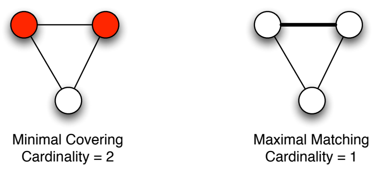


Figure 8.7. In general, the cardinality of a maximal matching is not the same as the cardinality of a minimal vertex covering, though the inequality that the cardinality of the maximal matching is at most the cardinality of the minimal covering does hold.

Theorem, minimizing $|J|$ and maximizing $|P|$ implies that $|J| = |P|$ and thus $|M^*| = |C^*|$. This completes the proof. $\square$

Remark 8.44. It is clear that K onig's Theorem does not hold in general. To see this, consider K_3 (see Figure 8.7). In this case, the general inequality that that the cardinality of the maximal matching is at most the cardinality of the minimal covering does hold (and this will always hold), but we do not have equality.

Remark 8.45. Let $G = (V, E)$ be a (bipartite) graph with $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. The minimal vertex covering problem for G can be written as the integer programming problem:

$$(8.21)\quad \begin{cases} \min & x_1 + \cdots + x_m \\ s.t. & x_i + x_j \geq 1 \quad \forall \{v_i, v_j\} \in E \\ & x_i \in \{0,1\} \quad \forall i = 1, \dots, m \end{cases}$$

If $\mathbf{A}$ is the incidence matrix for G, then this problem can be written in matrix notation as:

$$(8.22)\quad \begin{cases} \min & \mathbf{1}^T\mathbf{x} \\ s.t. & \mathbf{A}^T\mathbf{x} \geq \mathbf{1} \\ & \mathbf{x} \in \{0,1\}^m \end{cases}$$

where $\mathbf{1}$ is a vector consisting of only ones of appropriate length.

Exercise 74. Consider the *relaxation* of Problem 8.21:

$$(8.23)\quad \begin{cases} \min & x_1 + \cdots + x_m \\ s.t. & x_i + x_j \geq 1 \quad \forall \{v_i, v_j\} \in E \\ & x_i \leq 1 \quad \forall i = 1, \dots, m \\ & x_i \geq 0 \quad \forall i = 1, \dots, m \end{cases}$$

Compute the dual problem for Problem 8.21. [Hint: Use slack and surplus variables to construct a problem with only equality constraints (except for $\mathbf{x} \geq \mathbf{0}$). Compute the dual problem. You will obtain an objective function that looks like $\max w_1 + \cdots + w_n + u_1 + u_2 + \cdots + u_m$ and some constraints.]

Exercise 75. Use the dual problem you constructed in the Exercise 74 along with a restriction to $0-1$ variables to show that to show that for K_3 the cardinality of the maximum matching cannot be greater than 1 while the cardinality of the minimum covering can be 2. Thus, illustrate that strong duality does not hold for integer programming problems.

Exercise 76. When G is a bipartite graph, argue that there is an integer solution of the dual problem you found in Exercise 74 (even if you don't force it to have $0-1$ variables). Finally prove that under the assumptions we've given in this chapter (i.e., that no isolated vertex exists in a bipartite graph G) the cardinality of a minimal vertex covering is equal to the cardinality of a maximal matching. [Hint: Argue an optimal integer solution exists to the relaxation of both problems. The result follows from strong duality.]

Remark 8.46. There are many other applications of Linear Programming (and Integer Programming) to the study of graphs. Please consult [**BJS04**, **KV08**, **PS98**, **WN99**] for details.

CHAPTER 9

A Short Introduction to Random Graphs

The study of random graphs presupposes a little knowledge of probability theory. For those readers not familiar with probability at all, Chapter 1 of [Gri11a] contains a small introduction to discrete probability spaces, which should provide sufficient background for the material on Random Graphs. The material can be obtained from http://www.personal.psu.edu/cxg286/Math486.pdf. We provide one definition that is not contained in Chapter 1 of [Gri11a] that is critical for the rest of this chapter.

Definition 9.1 (Bernoulli Random Variable). A *Bernoulli Random Variable* is a random variable underlying a probability space with exactly 2 outcomes 0 and 1 and for which the probability of 1 occurring is p.

Remark 9.2. It is easy to see that if X is a Bernoulli random variable, then (the expected value of X) $\mathbb{E}(X) = p = \Pr(X = 1)$. Furthermore, given a set of n Bernoulli Random variables, the expected number of these events that will be 1 is np.

Remark 9.3. Random graphs are misnamed, as it makes one think of a graph that is somehow random. In reality, the term random graph usually refers to a family of graphs, which serves as a discrete probability space on which we derive a probability measure that assigns to each graph some probability that it occurs.

1. Bernoulli Random Graphs

Definition 9.4 (Bernoulli Random Graph). Let $n \in \mathbb{Z}_+$ and let $p \in (0, 1)$ Then $\mathcal{G}(n, p)$ is *Bernoulli Family of Random Graphs*, which is the discrete probability space so that:

(1) The sample space is the set of all graphs with n vertices
(2) The probability that any two vertices chosen at random has an edge between them is p and this probability is independent of all other edges.
(3) Therefore, for any given graph $G = (V, E)$ in $\mathcal{G}(n, p)$ with $|E| = m$, the probability assigned to G is:

$$p^m(1-p)^{\binom{n}{2}-m}$$

Remark 9.5. Even though they are called Bernoulli Random Graphs, they were invented by E. N. Gilbert [Gil59]. Gilbert went on to look at other (more useful) classes of random graphs that are useful for modeling more realistic phenomena in communications [Gil61].

Example 9.6. We illustrate three graphs generated randomly where $n = 10$ and $p = 0.5$. That means that any two edges have a 50% chance of having an edge between them. The first two graphs each have 21 edges and therefore the probability of these graphs is $0.5^21 \times 0.5^24$, while the probability of the third graph is $0.5^24 \times 0.5^21$ because it has 24 edges. Of course these two values are identical.

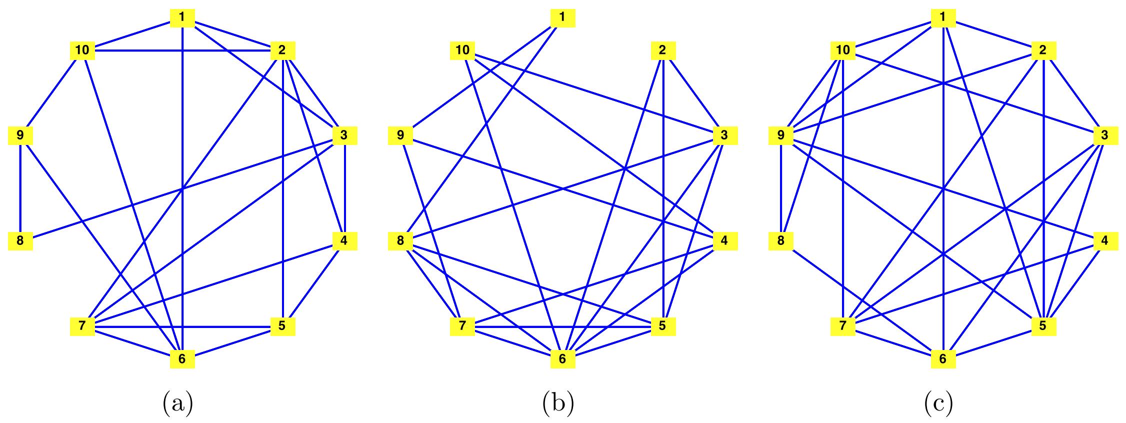


Figure 9.1. Three random graphs in the same random graph family $\mathcal{G}\left(10, \frac{1}{2}\right)$. The first two graphs, which have 21 edges, have probability $0.5^21 \times 0.5^24$. The third graph, which has 24 edges, has probability $0.5^24 \times 0.5^21$.

Theorem 9.7. *Let $2 \geq k \geq n$. Then the probability that a graph $G \in \mathcal{G}(n,p)$ has a set of k independent vertices is at most:*

$$\binom{n}{k}(1-p)^{\binom{k}{2}} \tag{9.1}$$

Proof. For any set of k vertices, the probability that they are independent is simply the probability that none of the pairs of vertices are connected by an edge. There are $\binom{k}{2}$ such pairs and each has a probability of $(1-p)$ of *not* being connected by an edge. There are $\binom{n}{k}$ subsets of the n vertices each containing k elements. Therefore, the probability that any of these sets is an independent set is:

$$\binom{n}{k}(1-p)^{\binom{k}{2}}$$

Thus we have proved:

$$\Pr(\alpha(G) \geq k) \leq \binom{n}{k}(1-p)^{\binom{k}{2}} \tag{9.2}$$

where α is the independence number of the graph G. □

Remark 9.8. Observe that Equation 9.2 is a weak bound in that if we choose k much smaller than n and fix p, then the bound exceeds 1. This is because, while it is true that each edge's existence in the set U in the proof is independent, not all the $\binom{n}{k}$ possible sets U are independent (they will share vertices). Thus we are over estimating the probability when we sum them in the proof.

Exercise 77. Find a limiting expression for the probability that $G \in \mathcal{G}(n,p)$ has a clique of size k with $2 \geq k \geq n$. [Hint: Remember, a clique is the exact opposite of an independent set.]

Definition 9.9 (Almost Sure Properties). Let P be a statement about a graph G. (For example, P might be the statement "G is a tree.") A property P is said to hold *almost surely* (abbreviated a.s) for graphs in the family $\mathcal{G}(n, p)$ if:

$$\lim_{n\to\infty} \Pr\left(P \text{ holds for an arbitrary graph } G \in \mathcal{G}(n,p)\right) = 1$$

Remark 9.10. This notion of almost surely is a funny one, but there's any easy way to get used to it. As n grows large, we are really absorbing more and more of the graphs that are possible (that is structures $G = (V, E)$ with $|V|$ some finite integer). If a property holds with probability 1 as n goes to infinity, it means that for almost every graph that property must hold because no matter how large an n we choose, there are always more graphs with more than n vertices than there are with fewer than n vertices. Thus, almost surely should be interpreted over the set of all graphs with a finite (but unbounded) number of vertices.

Lemma 9.11. *Let $p \in (0, 1)$ and let G be a graph in $\mathcal{G}(n, p)$. Then almost surely every pair of vertices v, u in G is connected by a path of length 2.*

Proof. Let $G = (V, E)$ and consider $w \in V\{v, u\}$. The probability that both edges $\{v, w\}$ and $\{u, w\}$ are in G is p^2. Thus the probability that at least one of these edges is absent is $1 - p^2$. Over all $n - 2$ possible choices for w, the probability that this occurs each time, therefore is:

$$\left(1 - p^2\right)^{n-2}$$

This quantity approaches 0 as n approaches infinity and thus, a.s. there is a path of length 2 connecting u and v. □

Theorem 9.12. *A graph $G \in \mathcal{G}(n, p)$ is almost surely connected.*

Proof. This proof is immediate from the previous lemma. □

Theorem 9.13. *Let $p \in (0, 1)$ and let G be a graph in $\mathcal{G}(n, p)$ and let $H = (V, E)$ be an arbitrary fixed graph. Then the property G has H as a subgraph holds a.s.*

Proof. Suppose that H contains m vertices. Then partition the n vertices available in graphs in $\mathcal{G}(n, p)$ into m sets each with size $k = \lfloor n/m \rfloor$. If there are vertices remaining, they may be added to an $m + 1^{\text{st}}$ partition and ignored. Suppose that H contains s edges. We'll order the vertices in each partition and choose the i^{th} element from each of these m partitions to be the vertices of H. (We will see that i will run from 1 to k.) With s edges, the probability that the edges of the graph H are present is exactly p^s and therefore the probability these edges are not present is $1-p^s$. Each of the k vertex m-tuples is independent of all the others because (by ordering the partitions) we do not repeat any pair of vertices that might form an edge. Thus, the probability that H is not spanned by *any* of these k m-tuples of vertices is:

$$(1 - p^s)^k = (1 - p^s)^{\lfloor n/m \rfloor}$$

Since $p \in (0, 1)$ we have:

$$\lim_{n\to\infty} (1 - p^s)^{\lfloor n/m \rfloor} = 0 \tag{9.3}$$

Thus, the probability that H is a subgraph approaches 1 $(1 = 1 - 0)$ as n approaches infinity. This completes the proof. □

Remark 9.14. Our use of independence in the proof of Theorem 9.13 is a little subtle. Clearly, when we are dealing with the m-tuple of vertices, the existence of the s edges between each of those m vertices is independent by the definition of the Bernoulli Random Graph Family. By ordering the elements in the m partitions of the vertices and choosing only m-tuples that never repeat vertices, we can be sure that we never repeat edges and thus the $\lfloor n/m \rfloor$ trials we take to find H as a subgraph are independent. This is how we can obtain Equation 9.3. Compare this to the proof of Theorem 9.7, in which we only seek a bound.

Exercise 78. Let G be a graph in $\mathcal{G}(n, p)$ for $p \in (0, 1)$. Show that the property G is *not* a forest holds a.s.

2. First Order Graph Language and $0-1$ properties

Remark 9.15. Theorems like Theorem 9.13 are called $0-1$ properties. In turns out that a great deal of these types of properties exist for simple graphs, as we prove.

Definition 9.16 (First Order Graph Language and the First Order Theory of Graphs). The *first order graph language* consists of a single relation (verb) E in which we write x and y are connected by an edge as $\{x, y\} \in E$; here x and y are *variables* (vertex placeholders).

Remark 9.17. Sentences in first order graph language are formed from the connectives *and* ($\wedge$), *or* ($\vee$), *implies* ($\implies$), *not* ($\neg$) and the quantifiers there *exists* ($\exists$) and *for all* ($\forall$).

Definition 9.18 (First Order Theory of Graphs). In *the first order theory of graphs*, we assume that

(1) for all x and y, $\{x, y\} \in E \iff \{y, x\} \in E$ and
(2) $\{x, x\} \notin E$ for any x.

We also assume that there is an equality operation so that we can determine whether to variables are equal.

Example 9.19. A very simple statement in first order graph language is "There is a path of length 3." In this case, the statement would read:

$$\exists v_1 \exists v_2 \exists v_3 \exists v_4 (\{v_1, v_2\} \in E \wedge \{v_2, v_3\} \in E \wedge \{v_3, v_4\} \in E \\ \wedge v_1 \neq v_2 \wedge v_1 \neq v_3 \wedge v_1 \neq v_4 \wedge v_2 \neq v_3 \wedge v_2 \neq v_4 \wedge v_3 \neq v_4) \tag{9.4}$$

This simply says, there exists four distinct vertices v_1, v_2, v_3 and v_4 and v_1 and v_2 are adjacent, v_2 and v_3 are adjacent, and v_3 and v_4 are adjacent. Other statements become harder to write, but many graph properties can be expressed in first order graph language.

Theorem 9.20. *For every property $\mathcal{P}$ written in first order graph language either $\mathcal{P}$ holds a.s. or $\mathcal{P}$ fails to holds a.s.* □

Remark 9.21. The interested reader should see [Bol01] (Chapter 2) or (for the logicians point of view) see [Mar00] (Chapter 2) for a proof of this theorem. This tells us that any property that can be expressed in first order graph language is no particularly interesting in that either it will hold for almost every graph *or* fail to hold for almost every graph.

3. Erdös-Rényi Random Graphs

Definition 9.22 (Erdös-Rényi Family of Random Graphs). Let $n \in \mathbb{Z}_+$ and let m be an integer between 0 and $\binom{n}{2}$ (see Corollary 2.45). Then $\mathcal{G}(n, m)$ is *Erdös-Rényi Family of Random Graphs*, which is the discrete probability space so that:

(1) The sample space is the set of all graphs with n vertices and m edges and
(2) The probability function assigns a uniform probability:

$$p(G) = \binom{\binom{n}{2}}{m}^{-1}$$

to each graph in the sample space.

Remark 9.23. Erdös-Rényi graphs were developed by P. Erdös and A. Rényi in 1959 while studying the interactions of probability theory and discrete mathematics [**ER60, ER59**]. Before giving an example of an Erdös-Rényi Family of Random Graphs, we require three lemmas, which will be useful.

Lemma 9.24. *Consider the Path Graph P_n; that is the graph on n vertices that is itself a path. The isomorphism type of P_n contains exactly $n!/2$ distinct graphs.*

Proof. Consider, without loss of generality the example graph shown in Figure 9.2 For

Figure 9.2. A path graph with 4 vertices has exactly $4!/2 = 12$ isomorphic graphs obtained by rearranging the order in which the vertices are connected.

n vertices, we could arrange these vertices in any of $n!$ ways. However, (by way of example) the graph in which the vertices are arranged in order 1 to n with vertex i adjacent to vertex $i-1$ and vertex $i+1$ (save for vertices 1 and n, which are only adjacent to vertices 2 and $n-1$ respectively) has an identical edge set to the graph with the vertices arranged in the order n to 1. Thus, to obtain the size of the isomorphism type of P_n, we must divide $n!$ by 2 to remove duplicates. Thus the size of the isomorphism type of P_n is $n!/2$. This completes the proof. $\square$

Lemma 9.25. *There are exactly $n+1$ distinct graphs in the isomorphism type of S_n (the star graph on n vertices).*

Proof. By way of example, consider the graph S_3 shown in Figure 9.3. We can choose any of the 4 vertices to be the center of the star and obtain a different edge relation. For S_n, which contains $n+1$ vertices, there are $n+1$ possible vertices that could act as the center of the star. Thus there are $n+1$ distinct graphs in the isomorphism class. This completes the proof. $\square$

Remark 9.26. In the previous two lemmas, we were interested in the number of distinct graphs in the isomorphism class. Here distinct means that the edge sets are different. In the case of the star graph shown in Figure 9.3 this means that the graph in which vertex 1 is

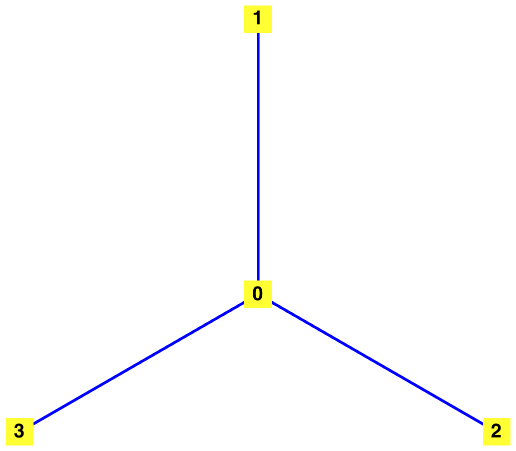


Figure 9.3. There are 4 graphs in the isomorphism class of S_3, one for each possible center of the star.

the center is distinct from the star graph with vertex 4 at the center because the edge sets in these two instances would be:

$$E = \{\{0,1\},\{0,2\},\{0,3\}\}\; E' = \{\{3,0\},\{3,1\},\{3,2\}\}$$

Note this is distinct from the number of automorphisms, which might be quite a bit higher because two distinct automorphism might create the same edge set. (For example, in the figure if we map the vertex 1 to 3 and leave all other vertices unchanged, this is an automorphism, but the edge structure has not changed.)

Lemma 9.27. *The number of distinct graphs in the isomorphism class of K_n is 1.* □

Exercise 79. Prove Lemma 9.27.

Example 9.28. This example appears in Chapter 11.3 of [GY05]. We just explain it a little more completely. Consider the random graph family $\mathcal{G}(5,3)$. This family of graphs contains

$$\binom{\binom{5}{2}}{3} = \binom{10}{3} = 120$$

Some of these graphs are isomorphic however. In fact there are exactly 4 isomorphism classes contained in $\mathcal{G}(5,3)$, which we illustrate through exhaustive enumeration. Consider the graph shown in Figure 9.4(a): This graph consists of an isolated vertex (in the figure Vertex 5) and a copy of P_4. There are 5 ways that the isolated vertex can be chosen and from Lemma 9.24 we know there are $4!/2 = 12$ elements in the isomorphism class of P_4. Thus there are 60 graphs isomorphic to the on in Figure 9.4(a) in $\mathcal{G}(5,3)$.

Another type of graph we might consider is shown in Figure 9.4(b). This graph has an isolated vertex and a copy of the star graph S_3. Again, there are 5 distinct ways to choose a vertex as the isolated vertex and by Lemma 9.25 there are 4 distinct graphs in the S_3 isomorphism class. Thus there are 20 distinct graphs isomorphic to the on in Figure 9.4(b) in $\mathcal{G}(5,3)$.

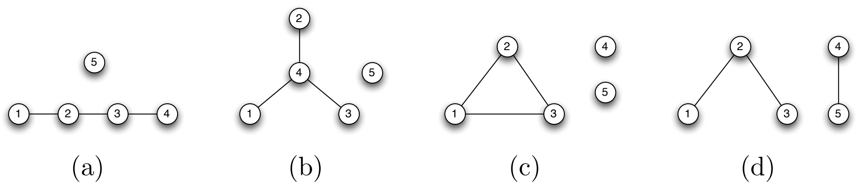


Figure 9.4. The 4 isomorphism types in the random graph family $\mathcal{G}(5,3)$. We show that there are 60 graphs isomorphic to this first graph (a) inside $\mathcal{G}(5,3)$, 20 graphs isomorphic to the second graph (b) inside $\mathcal{G}(5,3)$, 10 graphs isomorphic to the third graph (c) inside $\mathcal{G}(5,3)$ and 30 graphs isomorphic to the fourth graph (d) inside $\mathcal{G}(5,3)$.

The third graph type in $\mathcal{G}(5,3)$ is shown in Figure 9.4(c). Here there are two isolated vertices and one copy of K_3. By Lemma 9.27 we know there is only one distinct element in the isomorphism class of K_3, but there are $\binom{5}{2} = 10$ ways to choose the two isolated vertices. Thus there are 10 distinct graphs isomorphic to the on in Figure 9.4(c) in $\mathcal{G}(5,3)$.

In Figure 9.4(d) we have the final graph type that appears in $\mathcal{G}(5,3)$. We can tell this because we have a single copy of K_2 which has only one distinct element in its isomorphism class by Lemma 9.27 and a copy of S_2, which by Lemma 9.25 has 3 elements in its isomorphism class. There are $\binom{5}{2} = 10$ to pick the two vertices that form K_2 thus there are 30 distinct graphs isomorphic to the on in Figure 9.4(d) in $\mathcal{G}(5,3)$.

Since $60 + 20 + 10 + 30 = 120$ we know we have enumerated all the distinct graphs in $\mathcal{G}(5,3)$. This yields some interesting properties. For example, it we let X be the random variable that returns the number of isolated vertices assuming we draw a graph (sample) at random from the family $\mathcal{G}(5,3)$, then we see that:

$$\mathbb{E}(X) = (1)\frac{60}{120} + (1)\frac{20}{120} + (2)\frac{10}{120} + (0)\frac{30}{120} = \frac{100}{120} = \frac{5}{6} \tag{9.5}$$

We might instead define Y be to the random variable that is 1 if and only if there is a copy of K_3 (as a subgraph) of a graph chosen at random from $\mathcal{G}(5,3)$ and 0 else. Then Y is a Bernoulli Random Variable (see Remark 9.2.) In this case, it's easy to see that:

$$\Pr(Y = 1) = \frac{10}{120} = \frac{1}{12} \tag{9.6}$$

since only the graphs isomorphic to the graph in Figure 9.4(c) contain a copy of K_3.

Remark 9.29. Given a graph $G = (V, E)$ (possibly drawn from a random graph family) we say that G has a copy of K_n if there is a subgraph H of G that is isomorphic to K_n. Figure 9.4 obviously contains a copy of K_3 and every graph with at least one edge contains a copy of K_2.

Theorem 9.30. *The expected number of copies of K_s in a graph chosen at random from $\mathcal{G}(m,n)$ is:*

$$\binom{n}{s}\binom{m}{\binom{s}{2}}\binom{\binom{n}{2}}{\binom{s}{2}}^{-1} \tag{9.7}$$

PROOF. Define a random variable Y to be 1 just in case there the vertex set $\{v_1, \dots, v_s\}$ induces a complete sub-graph in a randomly chosen graph from $\mathcal{G}(m, n)$ and 0 otherwise. We now ask the question, what is the probability that $Y = 1$ (or equivalently what is $\mathbb{E}(Y)$). If we let this probability be p then, from Remark 9.2, the expected number of copies of K_s is simply:

$$\binom{n}{s}p$$

since there are $\binom{n}{s}$ ways to pick these s vertices and by Lemma 9.27 the isomorphism class of K_n contains only one element. To compute p, consider the following. There are out of all the graphs in $\mathcal{G}(m, n)$, we must choose one in which $k = \binom{s}{2}$ edges are proscribed (because they link the elements of $\{v_1, \dots, v_s\}$). We now may select $m - k$ edges from a possible collection of $\binom{n}{2} - k$ edges. Thus, there are:

$$\binom{\binom{n}{2} - k}{m - k}$$

ways this can be done. This means the probability of choosing any one of these graphs is:

$$p = \binom{\binom{n}{2} - k}{m - k}\binom{\binom{n}{2}}{m}^{-1} \tag{9.8}$$

since there are:

$$\binom{\binom{n}{2}}{m}$$

graphs in the family $\mathcal{G}(m, n)$ from Definition 9.22. Simplifying Equation 9.8 we obtain:

$$p = \binom{\binom{n}{2} - k}{m - k}\binom{n2}{m}^{-1} = \frac{\left(\binom{n}{2} - k\right)!}{\left(\binom{n}{2} - k - (m - k)\right)!(m - k)!}\frac{m!\left(\binom{n}{2} - m\right)!}{\binom{n}{2}!} = \tag{9.9}$$

$$\frac{\left(\binom{n}{2} - k\right)!}{(m - k)!}\frac{m!}{\binom{n}{2}!} = \frac{\left(\binom{n}{2} - k\right)!}{\binom{n}{2}!}\frac{m!}{(m - k)!} = \frac{\left(\binom{n}{2} - k\right)!k!}{\binom{n}{2}!}\frac{m!}{k!(m - k)!} = \binom{\binom{n}{2}}{k}^{-1}\binom{m}{k}$$

Thus we see that the expected number of copies of K_s in a graph chosen at random from $\mathcal{G}(m, n)$ is:

$$\binom{n}{s}\binom{m}{k}\binom{\binom{n}{2}}{k}^{-1} = \binom{n}{s}\binom{m}{\binom{s}{s}}\binom{\binom{n}{2}}{\binom{s}{s}}^{-1} \tag{9.10}$$

This completes the proof. $\square$

LEMMA 9.31. *There are exactly* $(n - 1)!/2$ *distinct graphs in the isomorphism type of* C_n *(the cycle graph on* n *vertices).*

PROOF. We will build a cycle iteratively using the n vertices allotted. Choose a vertex at random from the n vertices and denote it v_1. This vertex will be joined by an edge to exactly one vertex v_2. There are $n - 1$ possible choices for v_2. Now, v_2 will be joined by an edge to exactly one (new) vertex v_3 for which we have $n - 2$ possible choices. We can now see that the number of ways of making a such a cycle is $(n - 1)(n - 2)\cdots(1)$. However, The cycle that first creates an edge from v_1 to v_2 and then v_2 to v_3 and so on is precisely the same as the cycle that first creates and edge from v_1 to v_n and then from v_n to v_{n-1} and so on. Thus, the value $(n-1)(n-2)\cdots(1)$ double counts each cycle once. Therefore, the total number of distinct graphs in the isomorphism class of C_n is $(n-1)!/2$. □

THEOREM 9.32. *The expected number of copies of C_k in a random graph chosen from the family $\mathcal{G}(m,n)$ is:*

$$\frac{(k-1)!}{2}\binom{n}{k}\binom{m}{k}\binom{\binom{n}{2}}{k}^{-1} \tag{9.11}$$

PROOF. We follow the proof of Theorem 9.30. Define a random variable Y to be 1 just in case there the vertex set $\{v_1,\dots,v_s\}$ induces a cycle C_k in a randomly chosen graph from $\mathcal{G}(m,n)$ and 0 otherwise. We now ask the question, what is the probability that $Y = 1$ (or equivalently what is $\mathbb{E}(Y)$). If we let this probability be p then, from Remark 9.2 and Lemma 9.31, the expected number of *distinct* copies of C_k is simply:

$$\frac{(n-1)!}{2}\binom{n}{k}p$$

since there are $\binom{n}{k}$ ways to choose the k vertices in C_k and $(n-1)!/2$ distinct isomorphic copies of each cycle C_k on k vertices. Further, since C_k has k edges, we have already derived the value of p in the proof of of Theorem 9.30. We have:

$$p = \binom{\binom{n}{2}}{k}^{-1}\binom{m}{k}$$

Thus we conclude that the expected number of copies of C_k in a random graph chosen from the family $\mathcal{G}(m,n)$ is

$$\frac{(k-1)!}{2}\binom{n}{k}\binom{m}{k}\binom{\binom{n}{2}}{k}^{-1}$$

This completes the proof. □

EXERCISE 80. Let H be a graph on k vertices whose isomorphism class contains exactly s distinct elements. State and prove a theorem on the expected number of copies of H in a random graph chosen from the family $\mathcal{G}(m,n)$. [Hint: Theorem 9.32 is an example of such a general theorem. Extrapolate from this.]

REMARK 9.33. The spaces $\mathcal{G}(n,\frac{1}{2})$ and $\mathcal{G}(n,m)$ are closely related to each other. Consider $\mathcal{G}(n,\frac{1}{2})$ restricted to only those graphs with exactly m edges. The probability of any one of these graphs in $\mathcal{G}(n,\frac{1}{2})$ is:

$$\left(\frac{1}{2}\right)^m\left(\frac{1}{2}\right)^{\binom{n}{2}-m} \tag{9.12}$$

That is, they all have equal probability in $\mathcal{G}(n,\frac{1}{2})$. But, if we were to compute the conditional probability of any of these graphs in $\mathcal{G}(n,\frac{1}{2})$ given that we require a graph to have m edges, then their probabilities all reduce to precisely the probability one expects in the model $\mathcal{G}(n,m)$ by the properties of conditional probability.

CHAPTER 10

Some More Algebraic Graph Theory

1. Vector Spaces and Linear Transformation

REMARK 10.1. We generalize the notion of a vector in the definition of a vector space. It should be noted that all the definitions we have made thus far are valid in this framework assuming that vectors are defined as row or column vectors.

DEFINITION 10.2 (Vector Space). A *vector space* is a tuple $(\langle F, +, \cdot, 0, 1\rangle, V, +, \cdot)$ where

(1) $\langle F, +, \cdot, 0, 1\rangle$ is a field (with its own addition and multiplication operators defined) called the set of scalars,
(2) V is a set called the *set of vectors*,
(3) $+ : V \times V \rightarrow V$ is an addition operator defined on the set V, and
(4) $\cdot : F \times V \rightarrow V$.

Further the following properties hold for all vectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ in V and scalars in s, s_1 and s_2 in F:

(1) $(V, +)$ is a commutative group (of vectors) with identity element $\mathbf{0}$.
(2) Multiplication of vectors by a scalar distributes over vector addition; i.e., $s(\mathbf{v}_1 + \mathbf{v}_2) = s\mathbf{v}_1 + s\mathbf{v}_2$.
(3) Multiplication of vectors by a scalar distributes over field addition; i.e., $(s_1+s_2)\cdot\mathbf{v}_1 = s_1\mathbf{v}_1 + s_2\mathbf{v}_2$.
(4) Multiplication of a vector by a scalar respects the fields multiplication; i.e., $(s_1 \cdot s_2) \cdot v_1 = s_1 \cdot (s_2 \cdot v_1)$.
(5) Scalar identify is respected in the multiplication of vectors by a scalar; i.e., $1\mathbf{v_1} = \mathbf{v_1}$.

EXAMPLE 10.3. The simplest (and most familiar) example of a vector space has as its field $\mathbb{R}$ with addition and multiplication defined as expected and as its set of vectors tuples in $\mathbb{R}^n$ $(n \geq 1)$ with vector addition defined as one would expect. In this case, the vectors can be thought of as either column or row vectors (but usually as column vectors).

REMARK 10.4. Generally speaking, we will not explicitly call out all the different operations, vector sets and fields unless it is absolutely necessary. When referring to vector spaces over the field $\mathbb{R}$ with vectors $\mathbb{R}^n$ $(n \geq 1)$ we will generally just say the vector space $\mathbb{R}^n$ to mean the set of (column) vectors with n elements over the field of scalars $\mathbb{R}$. This is the vector space with which we will be most concerned at present.

REMARK 10.5. Vector spaces can become very abstract as we'll see when we construct the edge and vertex space of a graph. For now though it is easiest to remember that vector spaces behave like vectors of real numbers with some appropriate additions and multiplications defined. In general, all you need to define a vector space is a field (the scalars) a group (the vectors) and a multiplication operation (scalar-vector multiplication) that connects the two and that satisfies all the properties listed in Definition 10.2.

Remark 10.6. Note that dot products are not necessarily defined over general vector spaces. They are, however, defined over vector spaces in which vectors are n-tuples (or column / row vectors) with elements drawn from the underlying field as in $\mathbb{R}^n$.

Definition 10.7 (Subspace). If $\mathcal{V} = (\langle F, +, \cdot, 0, 1\rangle, V, +, \cdot)$ is a vector space and $U \subseteq V$ with $\mathcal{U} = (\langle F, +, \cdot, 0, 1\rangle, U, +, \cdot)$ also a vector space then, $\mathcal{U}$ is called a subspace of $\mathcal{V}$. Note that U must be closed under $+$ and $\cdot$.

Example 10.8. If we consider $\mathbb{R}^3$ as a vector space over the reals (as usual) then it has as a subspace several copies of $\mathbb{R}^2$. The easiest is to consider the subset of vectors:

$$U = \{(x, y, 0) : x, y \in \mathbb{R}\}$$

Clearly U is closed under the addition and scalar multiplication of the original vector space.

Definition 10.9 (Linear Transformation). Let $\mathcal{V}$ and $\mathcal{U}$ be two vector spaces (over some appropriately defined fields). A *linear transformation* is a function $f : \mathcal{V} \to \mathcal{U}$ such that:

(1) $f(\mathbf{v}_1 + \mathbf{v}_2) = f(\mathbf{v}_1) + f(\mathbf{v}_2)$ for all vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ in vector space $\mathcal{V}$ and
(2) $f(s\mathbf{v}) = sf(\mathbf{v})$ for all vectors $\mathbf{v}$ and scalars s in vector space $\mathcal{V}$.

Example 10.10. Consider the vector spaces $\mathbb{R}^2$ and $\mathbb{R}^3$ and the matrix:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Then the function $f : \mathbb{R}^2 \to \mathbb{R}^3$ with $f(\mathbf{x}) = \mathbf{M}\mathbf{x}$ is a linear transformation. To see this, consider

$$\mathbf{x}_1 = \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} x_{21} \\ x_{22} \end{bmatrix}$$

Then we have:

$$\mathbf{M}(\mathbf{x}_1 + \mathbf{x}_2) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \left(\begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} + \begin{bmatrix} x_{21} \\ x_{22} \end{bmatrix} \right) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_{11} + x_{21} \\ x_{12} + x_{22} \end{bmatrix} =$$

$$\begin{bmatrix} (x_{11} + x_{21}) + 2(x_{12} + x_{22}) \\ 3(x_{11} + x_{21}) + 4(x_{12} + x_{22}) \\ 5(x_{11} + x_{21}) + 6(x_{12} + x_{22}) \end{bmatrix} = \begin{bmatrix} (x_{11} + 2x_{12}) + (x_{21} + 2x_{22}) \\ (3x_{11} + 4x_{12}) + (3x_{12} + 4x_{22}) \\ (5x_{11} + 6x_{12}) + (5x_{12} + 6x_{22}) \end{bmatrix} =$$

$$\begin{bmatrix} x_{11} + 2x_{12} \\ 3x_{11} + 4x_{12} \\ 5x_{11} + 6x_{12} \end{bmatrix} + \begin{bmatrix} x_{21} + 2x_{22} \\ 3x_{12} + 4x_{22} \\ 5x_{12} + 6x_{22} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_{21} \\ x_{22} \end{bmatrix} = \mathbf{M}\mathbf{x}_1 + \mathbf{M}\mathbf{x}_2$$

A similar argument will show that $\mathbf{M}(c\mathbf{x}) = c\mathbf{M}\mathbf{x}$ for all vectors $\mathbf{x} \in \mathbb{R}^2$ and all scalars $c \in \mathbb{R}$.

Remark 10.11. It is (relatively) easy to generalize the previous example to see that (left) multiplication of a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ by a vector $\mathbf{x} \in \mathbb{R}^{n \times 1}$ constitutes a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^m$.

2. Linear Span and Basis

Remark 10.12. This section is a review of the material covered in Chapter 5.1. We now generalize it to an arbitrary field and vector space.

Definition 10.13. Let $\mathbf{x}_1, \dots, \mathbf{x}_m$ be vectors in a vector space $\mathcal{V}$ and let $\alpha_1, \dots, \alpha_m$ be scalars. Then

$$\alpha_1\mathbf{x}_1 + \cdots + \alpha_m\mathbf{x}_m \tag{10.1}$$

is a *linear combination* of the vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$.

Definition 10.14 (Span). Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ be a set of vectors in a vector space $\mathcal{V}$, then the span of $\mathcal{X}$ is the set:

$$\text{span}(\mathcal{X}) = \{\mathbf{y} \in \mathcal{V} : \mathbf{y} \text{ is a linear combination of vectors in } \mathcal{X}\} \tag{10.2}$$

Definition 10.15 (Linear Independence). Let $\mathbf{x}_1, \dots, \mathbf{x}_m$ be vectors in vector space $\mathcal{V}$. The vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$ are *linearly dependent* if there exists scalars $\alpha_1, \dots, \alpha_m$, not all zero, such that

$$\alpha_1\mathbf{x}_1 + \cdots + \alpha_m\mathbf{x}_m = \mathbf{0} \tag{10.3}$$

If the set of vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$ is not linearly dependent, then they are *linearly independent* and Equation 10.3 holds just in case $\alpha_i = 0$ for all $i = 1, \dots, n$.

Remark 10.16. It is worthwhile to note that the zero vector $\mathbf{0}$ makes any set of vectors a linearly dependent set.

Exercise 81. Prove the remark above.

Definition 10.17 (Basis). Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ be a set of vectors in vector space $\mathcal{V}$. The set $\mathcal{X}$ is called a *basis* of $\mathcal{V}$ if $\mathcal{X}$ is a linearly independent set of vectors and every vector in $\mathcal{V}$ is in the span of $\mathcal{X}$. That is, for any vector $\mathbf{w} \in \mathcal{V}$ we can find scalar values $\alpha_1, \dots, \alpha_m \in F$ such that

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i\mathbf{x}_i \tag{10.4}$$

Remark 10.18. The following statements on the size of a bases in vectors spaces are outside the scope of this course. Proof can be found in [Lan87].

Theorem 10.19. *Every basis of a vector space $\mathcal{V}$ has precisely the same cardinality.* □

Definition 10.20 (Dimension). The cardinality of any basis of a vector space $\mathcal{V}$ is called the *dimension* of the vector space.

Remark 10.21. Theorem 10.19 ensures that the dimension of a vector space is uniquely specified.

3. Vector Spaces of a Graph

Definition 10.22 (Galois Field with 2 Elements). The *Galois Field* with 2 elements (denoted GF2) is the field $(\{0,1\}, +, \cdot, 0, 1)$ where:

(1) $0 + 0 = 0$,
(2) $0 + 1 = 1$,
(3) $1 + 1 = 0$,
(4) $0 \cdot 0 = 0$,
(5) $0 \cdot 1 = 0$, and
(6) $1 \cdot 1 = 1$

Remark 10.23. The field GF2 is the first of an infinite number of finite fields that have several practical applications. The interested reader should consult [LP97] for an excellent introduction to applied abstract algebra.

Definition 10.24 (Symmetric Difference). Let S_1 and S_2 be any two sets. The *symmetric difference* of S_1 and S_2 is:

$$S_1 + S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1) \tag{10.5}$$

That is:

$$S_1 + S_2 = \{s \in S_1 \cap S_2 : s \text{ is in exactly one of } S_1 \text{ or } S_2\}$$

Definition 10.25 (Power Set). Let S be a set, then 2^S is the *power set* of S; that is, 2^S is the set of all subsets of S.

Definition 10.26 (Edge Space). Let $G = (V, E)$ be a graph. Then the *edge space* of G is the vector space with field GF2 and set of vectors 2^E, where vector addition is symmetric difference and scalar vector multiplication is defined as:

(1) If $S \subseteq E$ (i.e., $S \in 2^E$), then $0 \cdot S = \emptyset$ and
(2) If $S \subseteq E$, then $1 \cdot S = S$.

The edge space of G is generally denoted $\mathcal{E}$.

Theorem 10.27. *The tuple* $(\text{GF2}, 2^E, +, \cdot)$ *is a vector space.* □

Exercise 82. Prove Theorem 10.27.

Definition 10.28 (Vertex Space). Let $G = (V, E)$ be a graph. Then the *vertex space* of G is the vector space with field GF2 and set of vectors 2^V, where vector addition is symmetric difference and scalar vector multiplication is defined as:

(1) If $S \subseteq V$ (i.e., $S \in 2^V$), then $0 \cdot V = \emptyset$ and
(2) If $S \subseteq V$, then $1 \cdot V = V$.

The vertex space of G is generally denoted $\mathcal{V}$.

Theorem 10.29. *The tuple* $(\text{GF2}, 2^V, +, \cdot)$ *is a vector space.* □

Remark 10.30. Vector Space generated in this way are more abstract than the vector spaces we have discussed thus far. These vector spaces are connected to the theory of current and voltage in Kirchoff's Loop laws from Electricity and Magnatism [Pur84] and graph coloring.

Theorem 10.31. *Let $G = (V, E)$ be a graph. The set of singleton sets in 2^E forms a basis for $\mathcal{E}$. Therefore, $\mathcal{E}$ has dimension $|E|$.*

Proof. Let $E = \{e_1, \dots, e_m\}$ and let $S \subset E$. Then we have:

$$S = \alpha_1\{e_1\} + \alpha_2\{e_2\} + \cdots + \alpha_m\{e_m\} \tag{10.6}$$

with:

$$\alpha_i = \begin{cases} 1 & e_i \in S \\ 0 & \text{else} \end{cases}$$

Thus $B = \{\{e_1\}, \dots, \{e_m\}\}$ spans $\mathcal{E}$. To see that B is linearly independent, note that:

$$\emptyset = \alpha_1\{e_1\} + \alpha_2\{e_2\} + \cdots + \alpha_m\{e_m\}$$

is only possible if $\alpha_1 = \alpha_2 = \cdots = \alpha_m = 0$ since the elements of B are mutually disjoint. The fact that the dimension of $\mathcal{E}$ is $|E|$ follows at once from Definition 10.20. □

Exercise 83. State and prove a similar Theorem to Theorem 10.31 for $\mathcal{V}$.

Definition 10.32 (Characteristic Vector). Let $G = (V, E)$ Let S be a vector in $\mathcal{E}$. The characteristic vector of S is the m-dimensional vector of values drawn from GF2 so that:

$$S = \alpha_1\{e_1\} + \alpha_2\{e_2\} + \cdots + \alpha_m\{e_m\}$$

Remark 10.33. Obviously for any subset of V we may likewise define a characteristic vector based on Exercise 83. This means that $\mathcal{E}$ can really be considered to be the vector space GF2^n–the set of m-tuples of elements of GF2 for a graph $G = (V, E)$ with $|E| = n$. Likewise, if $|V| = m$, then $\mathcal{V}$ is equivalent to the vector space GF2^m. The following theorem is now straightforward to prove.

Theorem 10.34. *Let $G = (V, E)$ and let $\mathbf{M}$ be the incidence matrix of G. Then $\mathbf{M}$ is a linear transformation from $\mathcal{E}$ to $\mathcal{V}$ when elements of $\mathcal{E}$ are treated as their characteristic vectors and elements of $\mathcal{V}$ are treated as their characteristic vectors.*

Remark 10.35. Before proving Theorem 10.34 it is important to note that *all* operations are performed over GF2 and *not* the real numbers.

Proof of Theorem 10.34. The fact that multiplication by $\mathbf{M}$ is a linear transform from $\mathcal{E}$ to $\mathcal{V}$ is a result of the fact that matrix multiplication is always a linear transform from one vector space over a field to another vector space over the same field (see Remark 10.11). □

Exercise 84. Consider a graph $G = (V, E)$ and its incidence matrix $\mathbf{M}$. Let $\mathbf{x}$ be the characteristic vector for a standard basis vector in $\mathcal{E}$ (a vector corresponding to the one element edge sets of $\mathcal{E}$). What is the result (in $\mathcal{V}$) of the transformation $\mathbf{Mx}$?

4. Cycle Space

Definition 10.36 (Cycle Space). Let $G = (V, E)$ be a graph. The *cycle space* of G is an element of 2^E denoted $\mathcal{C}$ and is the smallest set (of sets) containing the $\emptyset$, all cycles in G (where each cycle is treated as a set of edges) and all unions of edge disjoint cycles in G.

Example 10.37. We show an example of the cycle space. The cycle space of a graph can be thought of as all the cycles contained in that graph along with the subgraphs consisting of cycles that only share vertices but *no edges*.

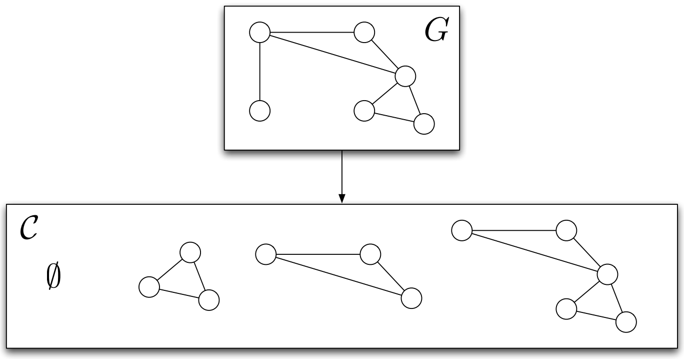


Figure 10.1. The cycle space of a graph can be thought of as all the cycles contained in that graph along with the subgraphs consisting of cycles that only share vertices but *no edges*. This is illustrated in this figure.

Theorem 10.38. *Let $G = (V, E)$ be a graph. The cycle space $\mathcal{C}$ is a subspace of $\mathcal{E}$.*

Proof. It suffices to prove that $\mathcal{C}$ is closed under $+$ since it is trivially closed under vector-scalar multiplication. Consider two elements C_1 and $C_2 \in \mathcal{C}$.

(1) If c_1 and c_2 are the edge disjoint cycles (or the empty set) then $C_1 + C_2$ is simply the union of edge disjoint cycles and clearly $C_1 + C_2 \in \mathcal{C}$.
(2) If c_1 is a union of edge disjoint cycles and C_2 is the union of edge disjoint cycles (or the empty set) and C_1 and C_2 share no edges in common, then it is again easy to see that $C_1 + C_2$ is simply the union of edge disjoint cycles and in $\mathcal{C}$.
(3) If C_1 and C_2 are unions of edge disjoint cycles but share edges in common, then C_1 and C_2 must share two or more cycles in common. In constructing $C_1 + C_2$ these common cycles will be removed (through symmetric differencing) and the result will be a union of edge disjoint cycles, which is clearly in $\mathcal{C}$.

Thus, $\mathcal{C}$ is closed under $+$ and it must be a vector space. □

Definition 10.39 (Fundamental Cycle). Let $G = (V, E)$ and let $F = (V, E')$ be a spanning forest (a spanning subgraph of G that is also a forest). A cycle is *fundamental* (with respect to F and G) if it is a cycle created by adding an edge e in E to F that is *not* in E'.

Example 10.40. We illustrate the creation of a fundamental cycle in the graph in Figure 10.2.

Definition 10.41 (Fundamental System of Edge Cycles). Let $G = (V, E)$ and let $F = (V, E')$ be a spanning forest. The *fundamental system of cycles* with respect to G and F is the set of all fundamental cycles of G with respect to F.

Exercise 85. Find the fundamental system of cycles for the graph shown in Figure 10.2.

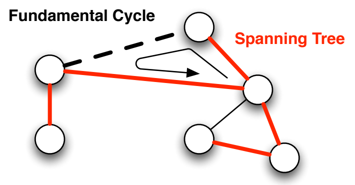


Figure 10.2. A fundamental cycle of a graph G (with respect to a spanning forest F) is a cycle created from adding an edge from the original edge set of G (not in F) to F.

THEOREM 10.42. *Let $G = (V, E)$ be a connected graph and let $T = (V, E')$ be a spanning tree of G. Then the fundamental system of cycles of G and T forms a basis for $\mathcal{C}$.*

PROOF. Recall first that by Theorem 3.59 such a spanning tree T exists when G is connected. Consider any fundamental cycle C. This cycle is constructed by adding exactly one edge to T and finding the cycle that results. Thus no fundamental cycle can be expressed as the sum of any other fundamental cycles because they will all be missing (at least) one edge. As a result, the fundamental system of cycles must be linearly independent.

Choose any element C of $\mathcal{C}$ and let $\{e_1, \dots, e_r\}$ be the edges of C that do not appear in E'. Further define the fundamental cycle C_i to be the one that arises as a result of adding e_i to T. The quantity $C + C_1 + \cdots + C_r = \emptyset$ if and only if $C = C_1 + \cdots + C_r$ because there are no edges in C that are not in $C_1 + \cdots + C_r$ and similarly no edges in $C_1 + \cdots + C_r$ that are not in C.

It is easy to see that no edge in the set $\{e_1, \dots, e_r\}$ appears in $C + C_1 + \cdots + C_r$ because each edge appears once in C and once in one of the C_i's and therefore, it will not appear in $C + C_1 + \cdots + C_r$. But this means that every edge in $C + C_1 + \cdots + C_r$ is an edge of T. More specifically, the sub-graph induced by the edges in $C + C_1 + \cdots + C_r$ is a sub-graph of T and thus necessarily acyclic. Every element of $\mathcal{C}$ induces a subgraph of G that has at least one cycle except for $\emptyset$. Thus, $C + C_1 + \cdots + C_r = \emptyset$.

Since our choice of C was arbitrary we can see that for any C we have:

$$C = \alpha_1 C_1 + \cdots + \alpha_k C_k$$

where $\{C_1, \dots, C_k\}$ is the fundamental set of cycles of G with respect to T and

$$\alpha_i = \begin{cases} 1 & \text{if the non-tree edge of } C_i \text{ is found in } C \\ 0 & \text{else} \end{cases}$$

Thus, the fundamental set of cycles is linearly independent and spans $\mathcal{C}$ and so it is a basis. This completes the proof. □

REMARK 10.43. The dimension of the space $\mathcal{C}$ for a graph $G = (V, E)$ is called the cyclomatic number of G and is, of course, equal to the size of (any) set of fundamental cycles generated by a spanning tree of G.

EXERCISE 86. The statement of Theorem 10.42 is stated and (more or less) proved following [**GY05**]. Diestel [**Die10**] has a different way of defining the basis that does not require G to be connected. Note that fundamental systems of cycles were defined for arbitrary graphs (rather than just connected ones). Is there any reason we couldn't just replace the spanning tree T with a spanning forrest (invoking Corollary 3.60) and state and prove a more general result?

5. Cut Space

Definition 10.44 (Cut Space). Let $G = (V, E)$ be a graph. The *cut space* of G is an element of 2^E denoted $\mathcal{C}^*$ and is the smallest set (of sets) containing the $\emptyset$, all minimal edge cuts in G and all unions of edge disjoint minimal edge cuts in G.

Example 10.45. We show an example of the cut space. The cut space of a graph can be thought of as all the minimal cuts contained in that graph along with the subgraphs consisting of minimal cuts that only share vertices but *no edges*.

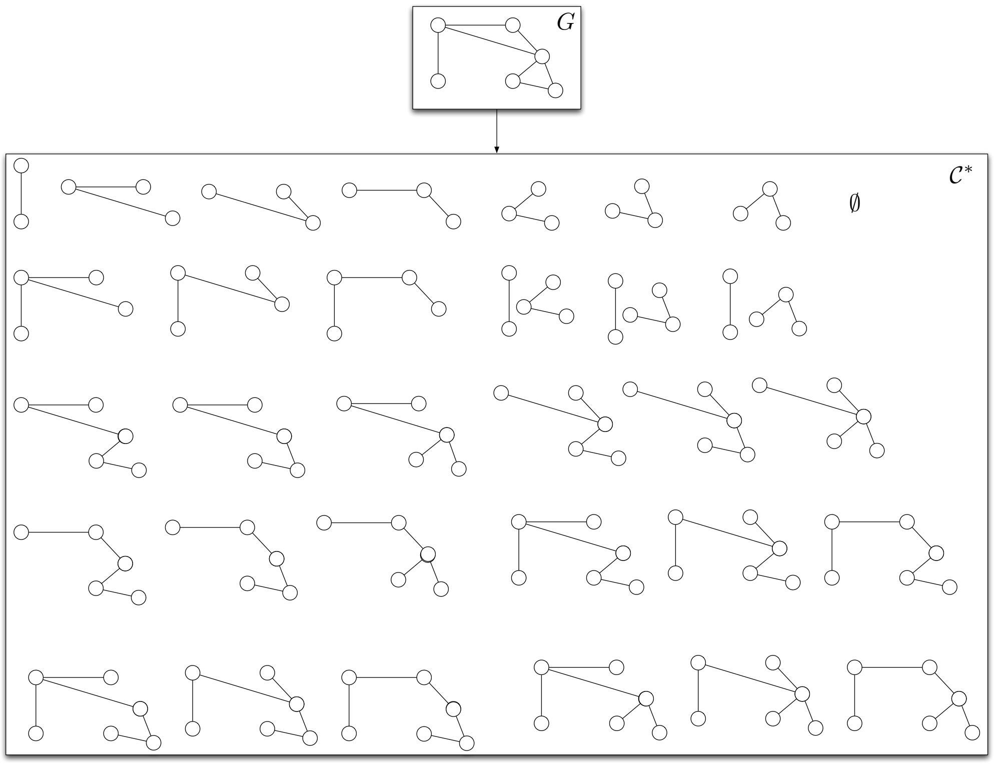


Figure 10.3. The cut space of a graph can be thought of as all the minimal cuts contained in that graph along with the subgraphs consisting of minimal cuts that only share vertices but *no edges*. This is illustrated in this figure.

Theorem 10.46. *Let $G = (V, E)$ be a graph. The cut space $\mathcal{C}^*$ is a subspace of $\mathcal{E}$.*

Proof. The proof is almost identical to the proof of Theorem 10.38, except we are dealing with elements made up of cuts, rather than cycles. □

Exercise 87. Using the proof of Theorem 10.38 construct a complete proof for Theorem 10.46.

Definition 10.47 (Partition Cut). Let $G = (V, E)$ be a graph and suppose that $V = V_1 \cup V_2$ with $V_1 \cap V_2 = \emptyset$. That is, V_1 and V_2 constitute a two element partition of V. Then the *partition cut* generated by V_1 and V_2 is the set of edges connecting elements in V_1 with elements in V_2. The partition cut is sometimes denotes $\langle V_1, V_2\rangle$.

Lemma 10.48. *Let $G = (V, E)$ be a connected graph and suppose that V_1 and V_2 partition V. Then the partition cut $\langle V_1, V_2\rangle$ is a minimal edge cut.*

Exercise 88. Prove Lemma 10.48. [Hint: The graph G is connected, so we may assume that the subgraphs induced by V_1 and V_2 are connected since $\langle V_1, V_2\rangle$ consists only of edges in which one end resides in V_1 and the other in V_2. Suppose you remove one edge from $\langle V_1, V_2\rangle$. The graph that results from removing these edges would still have a link connecting one vertex in V_1 with one vertex in V_2. The rest of the proof should be easy using the definition of connectivity.]

Definition 10.49 (Fundamental Edge Cut). Let $G = (V, E)$ and let $F = (V, E')$ be a spanning forest (a spanning subgraph of G that is also a forest). Let V_1 and V_2 be the vertices of the two new components formed when an edge e is removed from the forest F. Then the partition cut $\langle V_1, V_2\rangle$ is a *fundamental edge cut.*

Proposition 10.50. *Every fundamental edge cut is minimal.*

Proof. Apply Lemma 10.48 to the specific component of G in which the edge removed from the spanning forest resides. □

Example 10.51. We illustrate the creation of a fundamental edge-cut in the graph in Figure 10.4.

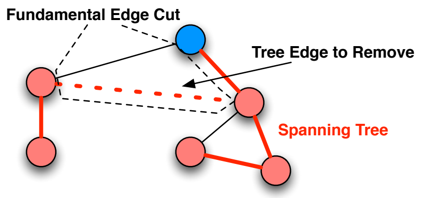


Figure 10.4. A fundamental edge cut of a graph G (with respect to a spanning forest F) is a partition cut created from partitioning the vertices based on a cut in a spanning tree and then constructing the resulting partition cut.

Definition 10.52 (Fundamental System of Edge Cuts). Let $G = (V, E)$ and let $F = (V, E')$ be a spanning forest. The *fundamental system of cycles* with respect to G and F is the set of all fundamental cycles of G with respect to F.

Exercise 89. Find the fundamental system of edge cuts cycles for the graph shown in Figure 10.4.

Lemma 10.53. *Let $G = (V, E)$ be a connected graph and let $T = (V, E')$ be a spanning tree of G. Every minimal edge cut of G contains at least one element of T.*

Proof. Let C be any minimal edge cut of G. We first show that the graph $G - C$ has 2 components. If $G - C$ has 1 component, then C is not an edge cut, contradicting our assumption. If $G-C$ has three components, then choose two of them. There must be edges connecting vertices in these two components in C because G was connected. Restoring any one of these edges will result in a graph that still has more than 1 component (since we assumed there were more than two components) and thus E' is not minimal.

Let V_1 and V_2 be the vertices of the two components that result from the removal of C from G. The fact that T is a spanning tree means that there is some (exactly one) edge $e \in E'$ connecting a vertex in V_1 to a vertex in V_2. This edge must be in C, otherwise, V_1 is connected to V_2 and C is not a cut. This completes the proof. □

Theorem 10.54. *Let $G = (V, E)$ be a connected graph. A set $E' \subseteq G$ is an edge cut if and only if every spanning tree of G contains at least one edge in E'.* □

Exercise 90. Prove Theorem 10.54. [Hint: Use Theorem 3.66 or Lemma 10.53 or both.]

Theorem 10.55. *Let $G = (V, E)$ be a connected graph and let $T = (V, E')$ be a spanning tree of G. Then the fundamental system of edge cuts of G and T forms a basis for $\mathcal{C}^*$.*

Proof. To see that the fundamental system of edge cuts is linearly independent, note that each system of edge cuts is uniquely defined by the removal of one edge from the spanning tree T and therefore this edge is an element of that cut. If V_1 and V_2 is the resulting partition, then there is exactly one edge in the spanning tree in the cut $\langle V_1, V_2\rangle$ (otherwise there would be more than one path connecting two vertices in the spanning tree) and thus this partition cut is uniquely defined by that edge. Thus no fundamental edge cut can be expressed as the sum of any other fundamental cycles because they will all be missing (at least) one edge. As a result, the fundamental system of edge cuts must be linearly independent.

Choose any element C of $\mathcal{C}^*$ and let $\{e_1, \dots, e_r\}$ be the edges of C that appear in E'. Further define the fundamental edge cut C_i to be the one that arises as a result of removing e_i from T.

It is easy to see that no edge in the set $\{e_1, \dots, e_r\}$ appears in $C + C_1 + \cdots + C_r$ because each edge appears once in C and once in one of the C_i's and therefore, it will not appear in $C + C_1 + \cdots + C_r$. But this means that every edge in $C + C_1 + \cdots + C_r$ is an edge that does not appear in T. Every element in $\mathcal{C}^*$ is the union of edge disjoint minimal edge cuts except for $\emptyset$. By Lemma 10.53, the only way $C + C_1 + \cdots + C_r$ will not contain an edge in T is if it is $\emptyset$. Thus $C = C_1 + \cdots + C_r$.

Since our choice of C was arbitrary we can see that for any C we have:

$$C = \alpha_1 C_1 + \cdots + \alpha_k C_k$$

where $\{C_1, \dots, C_k\}$ is the fundamental set of edge cuts of G with respect to T and

$$\alpha_i = \begin{cases} 1 & \text{if the edge of } C \text{ has edge } e_i \text{ found in } E' \\ 0 & \text{else} \end{cases}$$

Thus, the fundamental set of edge cuts is linearly independent and spans $\mathcal{C}^*$ and so it is a basis. This completes the proof. □

Definition 10.56 (Cycle Rank / Edge-Cut Rank). Let $G = (V, E)$ be a graph. The *edge-cut rank* of G is the number of edges in a spanning forest of G. The *cycle rank* (or Betti number) of G, denoted $\beta(G)$, is the number of edges in the relative complement of that spanning forest in G (see Definition 2.64).

Proposition 10.57. *Let $G = (V, E)$ be a graph. The Betti number of G is:*

$$\beta(G) = |E| - |V| - c(G)$$

While the edge cut rank of G is $|V| - c(G)$. □

Exercise 91. Prove Proposition 10.57. [Hint: Use Corollary 3.65.]

Corollary 10.58. *Let $G = (V, E)$ be a graph. The dimension of $\mathcal{C}$ is $\beta(G)$, while the dimension of the edge cut space is the edge cut rank, which is $|V| - c(G)$.*

Proof. This follows immediately from the proofs of Theorems 10.42 and 10.55. □

6. The Relation of Cycle Space to Cut Space

Lemma 10.59. *Let $G = (V, E)$ be a connected graph. Any cycle in G has an even number of edges in common with any minimal edge cut of G.*

Proof. Let C be any cycle and suppose that E' is a minimum edge cut in G. From the proof of Lemma 10.53 we know that the graph $G - E'$ has exactly two components. Let V_1 be the vertex set of one component and V_2 be the vertex set of the second component. If C is a subgraph of the first or second components, then C shares no edges in common with E' and the theorem is proved. Therefore, assume that C contains some vertices from V_1 and some from V_2. Consider the process of walking along C beginning at some vertex in V_1. At some point we must cross into V_2 along an edge in E' and then we will (eventually) cross back into V_1 along a different edge. This process may repeat a number of times, but each time we will incorporate 2 (new) edges from E'. Thus, C must have an even number of edges from E' since we begin and end our walk along C in either V_1 or V_2. □

Lemma 10.60. *Let $G = (V, E)$ be a connected graph. If a set $C \subseteq 2^E$ is in the cycle space $\mathcal{C}$ then it has an even number of edges in common with every element of $\mathcal{C}^*$. Similarly, if K is in the edge cut space $\mathcal{C}^*$, then it has an even number of edges in common with every element of $\mathcal{C}$.*

Proof. Let $C \in \mathcal{C}$. Then C is the union of edge disjoint cycles in G. Choose any element $K \in \mathcal{C}^*$. It is composed of the edge-disjoint unions of minimal edge cuts. Each cycle in C shares an even number of edges with each edge-cut of K and no two cycles (or edge cuts) share an edge. The result follows at once from Lemma 10.59.

The same argument suffices to show that if K is in the edge cut space, then it shares an even number of edges with every element of the cycle space. This completes the proof. □

Remark 10.61. You can actually show quite a bit more about these spaces. One can show that a set C is an element of the cycle space if and only if it shares an even number of edges with every element of the edge cut space and similarly that a set K is in the edge cut space if and only if it shares an even number of edges in common with every element of the cycle space. The interested reader should consult Chapter 4.6 of [GY05].

Definition 10.62 (Orthogonal Subspaces). Let $\mathcal{V}$ be a vector space defined over a field F with a dot product defined and let $\mathcal{W}_1$ and $\mathcal{W}_2$ be two subspaces. The subspaces spaces are *orthogonal* if for every vector $\mathbf{v}_1 \in \mathcal{W}_1$ and every vector $\mathbf{v}_2 \in \mathcal{W}_2$ we have $\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$, where $0 \in F$.

THEOREM 10.63. *Let $G = (V, E)$ be a graph with $|E| = n$. Then $\mathcal{C}$ and $\mathcal{C}^*$ are orthogonal subspaces.*

PROOF. Choose any elements $C \in \mathcal{C}$ and $\mathcal{K} \in \mathcal{C}^*$. Consider their characteristic vectors $\mathbf{c}$ and $\mathbf{k}$ in GF^n corresponding to these vectors. By Lemma 10.60, there are an even number of indices at which both $\mathbf{c}$ and $\mathbf{k}$ have value 1. Thus $\mathbf{c} \cdot \mathbf{k} = 0$ since in $\mathbf{GF}2$ and sum with an even number of 1's must be equal to 0. $\square$

REMARK 10.64. Before coming to our final result, we need one last definition and theorem from general Linear Algebra.

DEFINITION 10.65 (Direct Sum). Let $\mathcal{V}$ be a vector space with two subspaces $\mathcal{W}_1$ and $\mathcal{W}_2$. The *direct sum* of $\mathcal{W}_1$ and $\mathcal{W}_2$, written $\mathcal{W}_1 \oplus \mathcal{W}_2$ is the set of all vectors in $\mathcal{V}$ with form $\mathbf{v}_1 + \mathbf{v}_2$ for $\mathbf{v}_1 \in \mathcal{W}_1$ and $\mathbf{v}_2 \in \mathcal{W}_2$.

REMARK 10.66. The proof of the following theorem is well outside the scope of the course, but can be found in any text book on Linear Algebra. A version of it is found in Chapter 2 of [**Lan87**].

THEOREM 10.67. *Let $\mathcal{V}$ be a vector space with two subspaces $\mathcal{W}_1$ and $\mathcal{W}_2$. Then $\mathcal{W}_1$ and $\mathcal{W}_2$ is a subspace of $\mathcal{V}$ and:*

$$\dim(\mathcal{W}_1 \oplus \mathcal{W}_2) = \dim(\mathcal{W}_1) + \dim(\mathcal{W}_2) - \dim(\mathcal{W}_1 \cap \mathcal{W}_2) \tag{10.7}$$

where dim *indicates the dimension of the space and $\mathcal{W}_1 \cap \mathcal{W}_2$ is the vector space that results from intersecting the vector sets of $\mathcal{W}_1$ and $\mathcal{W}_2$.* $\square$

DEFINITION 10.68 (Orthogonal Complements). Let $\mathcal{V}$ be a vector space defined over a field F with a dot product defined and let $\mathcal{W}_1$ and $\mathcal{W}_2$ be two subspaces. The subspaces spaces are *orthogonal complements* if they are orthogonal and if $\mathcal{W}_1 \oplus \mathcal{W}_2 = \mathcal{V}$ and $\mathcal{V}_1 \cap \mathcal{W}_2 = \{\mathbf{0}\}$, where intersection is taken over the vectors of the subspaces.

THEOREM 10.69. *Let $G = (V, E)$ be a graph with cycle space $\mathcal{C}$ and edge cut space $\mathcal{C}^*$. These subspaces are orthogonal complements if and only if $\mathcal{C} \cap \mathcal{C}^* = \emptyset$.*

PROOF. The orthogonality of $\mathcal{C}$ and $\mathcal{C}^*$ is established in Theorem 10.63. Recall that the zero vector of $\mathcal{E}$ is $\emptyset$, so if $\mathcal{C} \cap \mathcal{C}^* = \emptyset$, then $\mathcal{C} \cap \mathcal{C}^* = \mathbf{0}$. From Theorem 10.67, we know that:

$$\dim(\mathcal{C} \oplus \mathcal{C}^*) = \dim(\mathcal{C}) + \dim(\mathcal{C}^*) - \dim(\mathcal{C} \cap \mathcal{C}^*) \tag{10.8}$$

From Corollary 10.58, we know that:

$$\dim(\mathcal{C}) = |E| - |V| - c(G)$$
$$\dim(\mathcal{C}^*) = |V| - c(G)$$

Thus:

$$\dim(\mathcal{C} \oplus \mathcal{C}^*) = |E|$$

if and only if

$$\dim(\mathcal{C} \cap \mathcal{C}^*) = 0$$

The latter can only happen when $\mathcal{C} \cap \mathcal{C}^* = \emptyset$. In this case, by necessity, $\mathcal{C} \oplus \mathcal{C}^* = \mathcal{E}$ and $\mathcal{C}$ and $\mathcal{C}^*$ are orthogonal complements. This completes the proof. $\square$

Bibliography

[AB00] R. Albert and A. Barabási, *Topology of evolving networks: Local events and universality*, Phys. Rev. Lett. **85** (2000), no. 24, 5234–5237.

[AB02] ———, *Statistical mechanics of complex networks*, Reviews of modern physics **74** (2002), no. 1, 47–97.

[ACL01] William Aiello, Fan Chung, and Linyuan Lu, *A random graph model for power law graphs*, Experiment. Math. **10** (2001), no. 1, 53–66.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[Ald08] D. L. Alderson, *Catching the "Network Science" Bug: Insight and Opportunity for the Operations Researcher*, Oper. Res. **56** (2008), no. 5, 1047–1065.

[BAJ99] A. Barabási, R. Albert, and H. Jeong, *Mean-field theory for scale-free random graphs*, Physica A **272** (1999), no. 1/2, 173–187.

[BAJB00] A. Barabási, R. Albert, H. Jeong, and G. Bianconi, *Power-law distribution of the world wide web*, Science **287** (2000), no. 5461, 2115.

[Bel57] R. Bellman, *Dynamic programming*, Princeton University Press, 1957.

[Ber73] C. Berge, *Graphs and hypergraphs*, North-Holland, 1973.

[BJS04] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali, *Linear programming and network flows*, Wiley-Interscience, 2004.

[BM08] A. Bondy and U. S. R. Murty, *Graph theory*, 3 ed., Springer, Graduate Texts in Mathematics, 2008.

[Bol00] Béla Bollobás, *Modern Graph Theory*, Springer, 2000.

[Bol01] B. Bollobás, *Random Graphs*, Cambridge University Press, 2001.

[Bol04] ———, *Extremal graph theory*, Dover Press, 2004.

[BP98] S. Brin and L. Page, *The anatomy of a large-scale hypertextual web search engine*, Seventh International World-Wide Web Conference (WWW 1998), 1998.

[BR03] Béla Bollobás and Oliver Riordan, *Robustness and vulnerability of scalefree random graphs*, Internet Mathematics **1** (2003), 1–35.

[Cha84] G. Chartrand, *Introductory graph theory*, Dover, 1984.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to algorithms*, 2 ed., The MIT Press, 2001.

[CSW05] P. J. Carrington, J. Scott, and S. Wasserman, *Models and Methods in Social Network Analysis*, Cambridge University Press, 2005.

[Dat95] B. N. Datta, *Numerical linear algebra*, Brooks/Cole, 1995.

[Die10] R. Diestel, *Graph theory*, 4 ed., Graduate Texts in Mathematics, Springer, 2010.

[Dij59] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik (59), 269–271.

[ER59] P. Erdös and A. Rényi, *On random graphs*, Publ. Math. Debrecen **6** (1959), 290–297.

[ER60] ———, *On the evolution of random graphs*, Publ. Math. Inst. Hungar. Acad. Sci. **5** (1960), 17–61.

[FCF11] A. Friggeri, G. Chelius, and E. Fleury, *Fellows: Crowd-sourcing the evaluation of an overlapping community model based on the cohesion measure*, Proc. Interdisciplinary Workshop on Information and Decision in Social Networks (Massachusetts Institute of Technology), Laboratory for lnformation and Decision Systems, May 31 - Jun 1 2011.

[Flo62] R. W. Floyd, *Algorithm 97: Shortest path*, Comm. ACM **5** (1962), no. 6, 345.

[Fra99] J. B. Fraleigh, *A First Course in Abstract Algebra*, 6 ed., Addison-Wesley, 1999.

[GB06] Christopher Griffin and Richard R. Brooks, *A note on the spread of worms in scale-free networks*, IEEE Transactions on Systems, Man and Cybernetics, Part B **36** (2006), no. 1, 198–202.

[Gil59] E. N. Gilbert, *Random Graphs*, Ann. Math. Statist. **4** (1959), 1141–1144.

[Gil61] ———, *Random plane networks*, J. Soc. Indus. Appl. Math. **9** (1961), no. 4, 533–543.

[GR01] C. Godsil and G. Royle, *Algebraic graph theory*, Springer, 2001.

[Gri11a] C. Griffin, *Game theory: Penn state math 486 lecture notes (v 1.02)*, http://www.personal.psu.edu/cxg286/Math486.pdf, 2010-2011.

[Gri11b] ———, *Linear programming: Penn state math 484 lecture notes (v 1.8)*, http://www.personal.psu.edu/cxg286/Math484_V1.pdf, 2010-2011.

[GY05] J. Gross and J. Yellen, *Graph theory and its applications*, 2 ed., CRC Press, Boca Raton, FL, USA, 2005.

[HU79] J. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, Reading, MA, 1979.

[Jos] S. Joseph, *The Max-Flow Min-Cut Theorem*, http://www.math.uri.edu/ eaton/maxflowmincut.pdf.

[Kru56] J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. AMS **7** (1956), no. 1.

[KV08] B. Korte and J. Vygen, *Combinatorial Optimization*, Springer-Verlag, 2008.

[KY08] D. Knoke and S. Yang, *Social Network Analysis*, Quantitative Applications in the Social Sciences, no. 154, SAGE Publications, 2008.

[Lan87] S. Lang, *Linear Algebra*, Springer-Verlag, 1987.

[LP97] R. Lidl and G. Pilz, *Applied Abstract Algebra*, Springer, 1997.

[Lu01] Linyuan Lu, *The diameter of random massive graphs*, Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, 2001, pp. 912–921.

[Mar00] D. Marker, *Model Theory: An Introduction*, 1 ed., Springer-Verlag, 2000.

[Mey01] C. D. Meyer, *Matrix analysis and applied linear algebra*, SIAM Publishing, 2001.

[MR95] Michael Molloy and Bruce Reed, *A critical point for random graphs with a given degree sequence*, Random Structures Algorithms **6** (1995), 161–179.

[NBW06] M. E. J. Newman, A. Barabási, and D. J. Watts, *The Structure and Dynamics of Networks*, Princeton University Press, 2006.

[New03] M. E. J. Newman, *The structure and function of complex networks*, SIAM Review **45** (2003), no. 2, 167–256.

[Oxl92] J. G. Oxley, *Matroid theory*, Oxford University Press, 1992.

[Pri57] R. C. Prim, *Shortst connection networks and some generalizations*, Bell System Technical Journal **36** (1957).

[PS98] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Dover Press, 1998.

[PSV01] Romualdo Pastor-Satorras and Alessandro Vespignani, *Epidemic dynamics and endemic states in complex networks*, Phys. Rev. E (3) **63** (2001), no. 66117.

[Pur84] E. M. Purcell, *Electricity and magnetism*, 2 ed., McGraw-Hill, 1984.

[Sim55] H. Simon, *On a class of skew distribution functions*, Biometrika **42** (1955), no. 3/4, 425–440.

[Sim05] S. Simpson, *Mathematical logic*, http://www.math.psu.edu/simpson/courses/math557/logic.pdf, December 2005.

[Spi11] L. Spizzirri, *Justication and application of eigenvector centrality*, `http://www.math.washington.edu/~morrow/336_11/papers/leo.pdf`, March 6 2011 (Last Checked: July 20, 2011).

[Tru94] R. J. Trudeau, *Introduction to graph theory*, 2 ed., Dover, 1994.

[WN99] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*, Wiley-Interscience, 1999.

[Zwi95] U. Zwick, *The smallest networks on which the ford-fulkerson maximum flow procedure may fail to terminate*, Theoretical Computer Science **148** (1995), no. 1, 165–170.