MATH20902: Discrete Mathematics

Mark Muldoon

March 20, 2020

Contents

I Notions and Notation

II Trees and the Matrix-Tree Theorem

III Eulerian and Hamiltonian Graphs

IV Distance in Graphs and Scheduling

V Planar Graphs

Part I

Notions and Notation

Lecture 1

First Steps in Graph Theory

This lecture introduces Graph Theory, the main subject of the course, and includes some basic definitions as well as a number of standard examples.

Reading: Some of the material in today's lecture comes from the beginning of Chapter 1 in

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, which is available online via SpringerLink.

If you are at the university, either physically or via the VPN, you can download the chapters of this book as PDFs.

1.1 The Königsberg Bridge Problem

Graph theory is usually said to have been invented in 1736 by the great Leonhard Euler, who used it to solve the Königsberg Bridge Problem. I used to find this hard to believe—the graph-theoretic graph is such a natural and useful abstraction that it's difficult to imagine that no one hit on it earlier—but Euler's paper about graphs[1] is generally acknowledged[2] as the first one and it certainly provides a satisfying solution to the bridge problem. The sketch in the left panel of Figure 1.1 comes from Euler's original paper and shows the main features of the problem. As one can see by comparing Figures 1.1 and 1.2, even this sketch is already a bit of an abstraction.

The question is, can one make a walking tour of the city that (a) starts and finishes in the same place and (b) crosses every bridge exactly once. The short answer to this question is "No" and the key idea behind proving this is illustrated in the right panel of Figure 1.1. It doesn't matter what route one takes while walking around on, say, the smaller island: all that really matters are the ways in which the bridges connect the four land masses. Thus we can shrink the small island to a

[1] L. Euler (1736), Solutio problematis ad geometriam situs pertinentis, *Commentarii Academiae Scientiarum Imperialis Petropolitanae* **8**, pp. 128–140.

[2] See, for example, Robin Wilson and John J. Watkins (2013), *Combinatorics: Ancient & Modern*, OUP. ISBN 978-0-19-965659-2.

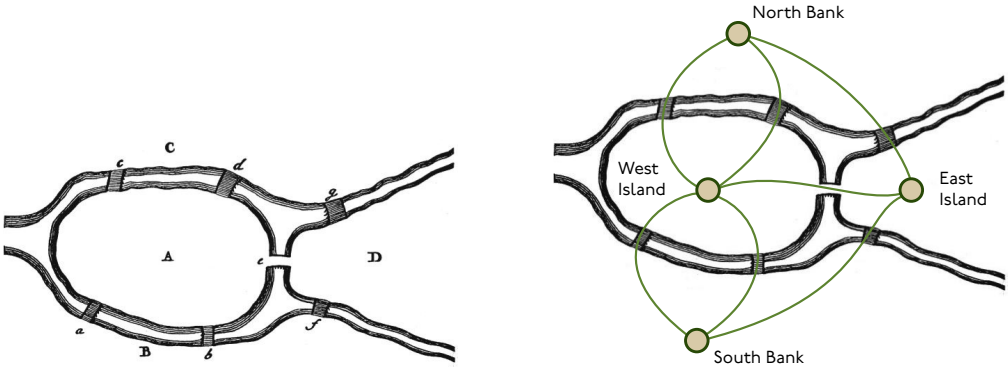


Figure 1.1: *The panel at left shows the seven bridges and four land masses that provide the setting for the Königsberg bridge problem, which asks whether it is possible to make a circular walking tour of the city that crosses every bridge exactly once. The panel at right includes a graph-theoretic abstraction that helps one prove that no such tour exists.*

Figure 1.2: *Königsberg is a real place—a port on the Baltic—and during Euler's lifetime it was part of the Kingdom of Prussia. The panel at left is a bird's-eye view of the city that shows the celebrated seven bridges. It was made by Matthäus Merian and published in 1652. The city is now called Kaliningrad and is part of the Russian Federation. It was bombed heavily during the Second World War: the panel at right shows a recent satellite photograph and one can still recognize the two islands and modern versions of some of the bridges, but very little else appears to remain.*

point—and do the same with the other island, as well as with the north and south banks of the river—and then connect them with arcs that represent the bridges. The problem then reduces to the question whether it is possible to draw a path that starts and finishes at the same dot, but traces each of over the seven arcs exactly once.

One can prove that such a tour is impossible by contradiction. Suppose that one exists: it must then visit the easternmost island (see Figure 1.3) and we are free to imagine that the tour actually starts there. To continue we must leave the island, crossing one of its three bridges. Then, later, because we are required to

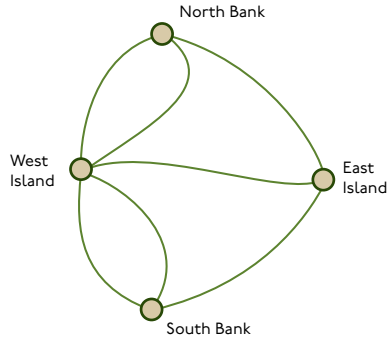


Figure 1.3: *The Königsberg Bridge graph on its own: it is not possible to trace a path that starts and ends on the eastern island without crossing some bridge at least twice.*

cross each bridge exactly once, we will have to return to the eastern island via a different bridge from the one we used when setting out. Finally, having returned to the eastern island once, we will need to leave again in order to cross the island's third bridge. But then we will be unable to return without recrossing one of the three bridges. And this provides a contradiction: the walk is supposed to start and finish in the same place and cross each bridge exactly once.

1.2 Definitions: graphs, vertices and edges

The abstraction behind Figure 1.3 turns out to be very powerful: one can draw similar diagrams to represent "connections" between "things" in a very general way. Examples include: representations of social networks in which the points are people and the arcs represent acquaintance; genetic regulatory networks in which the points are genes and the arcs represent activation or repression of one gene by another and scheduling problems in which the points are tasks that contribute to some large project and the arcs represent interdependence among the tasks. To help us make more rigorous statements, we'll use the following definition:

Definition 1.1. *A **graph** is a finite, nonempty set V, the **vertex set**, along with a set E, the **edge set**, whose elements $e \in E$ are pairs $e = (a, b)$ with $a, b \in V$.*

We will often write $G(V, E)$ to mean the graph G with vertex set V and edge set E. An element $v \in V$ is called a *vertex* (plural *vertices*) while an element $e \in E$ is called an *edge*.

The definition above is deliberately vague about whether the pairs that make up the edge set E are ordered pairs—in which case (a, b) and (b, a) with $a \neq b$ are distinct edges—or unordered pairs. In the unordered case (a, b) and (b, a) are just two equivalent ways of representing the same pair.

Definition 1.2. *An **undirected graph** is a graph in which the edge set consists of unordered pairs.*

Figure 1.4: *Diagrams representing graphs with vertex set $V = \{a, b\}$ and edge set $E = \{(a, b)\}$. The diagram at left is for an undirected graph, while the one at right shows a directed graph. Thus the arrow on the right represents the ordered pair (a, b).*

Definition 1.3. *A **directed graph** is a graph in which the edge set consists of ordered pairs. The term "directed graph" is often abbreviated as **digraph**.*

Although graphs are defined abstractly as above, it's very common to draw *diagrams* to represent them. These are drawings in which the vertices are shown as points or disks and the edges as line segments or arcs. Figure 1.4 illustrates the graphical convention used to mark the distinction between directed and undirected edges: the former are drawn as line segments or arcs, while the latter are shown as arrows. A directed edge $e = (a, b)$ appears as an arrow that points from a to b.

Sometimes one sees graphs with more than one edge[3] connecting the same two vertices; the Königsberg Bridge graph is an example. Such edges are called *multiple* or *parallel* edges. Additionally, one sometimes sees graphs with edges of the form $e = (v, v)$. These edges, which connect a vertex to itself, are called *loops* or *self loops*. All these terms are illustrated in Figure 1.5.

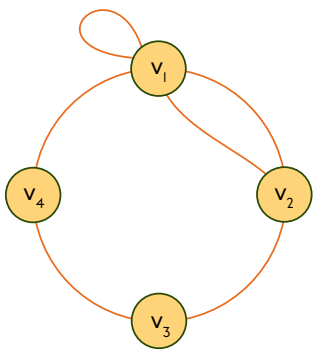


Figure 1.5: *A graph whose edge set includes the self loop (v_1, v_1) and two parallel copies of the edge (v_1, v_2).*

It is important to bear in mind that diagrams such as those in Figures 1.3–1.5 are only illustrations of the edges and vertices. In particular, the arcs representing edges may cross, but this does not necessarily imply anything: see Figure 1.6.

Remark. In this course when we say "graph" we will normally mean an undirected graph that contains no loops or parallel edges: if you look in other books you may

[3]In this case it is a slight abuse of terminology to talk about the edge "set" of the graph, as sets contain only a single copy of each of their elements. Very scrupulous books (and students) might prefer to use the term *edge list* in this context, but I will not insist on this nicety.

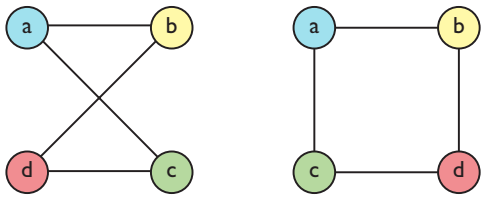


Figure 1.6: *Two diagrams for the same graph: the crossed edges in the leftmost version do not signify anything.*

see such objects referred to as *simple graphs*. By contrast, we will refer to a graph that contains parallel edges as a *multigraph*.

Definition 1.4. *Two vertices $a \neq b$ in an undirected graph $G(V, E)$ are said to be* ***adjacent*** *or to be* ***neighbours*** *if $(a,\, b) \in E$. In this case we also say that the edge $e = (a, b)$ is* ***incident on*** *the vertices a and b.*

Definition 1.5. *If the directed edge $e = (u, v)$ is present in a directed graph $H(V', E')$ we will say that u is a* ***predecessor*** *of v and that v is a* ***successor*** *of u. We will also say that u is the* ***tail*** *or* ***tail vertex*** *of the edge (u, v), while v is the* ***tip*** *or* ***tip vertex****.*

1.3 Standard examples

In this section I'll introduce a few families of graphs that we will refer to throughout the rest of the term.

The complete graphs K_n

The *complete graph* K_n is the undirected graph on n vertices whose edge set includes every possible edge. If one numbers the vertices consecutively the edge and vertex set are

$$\begin{aligned} V &= \{v_1,\, v_2,\, \ldots,\, v_n\} \\ E &= \{(v_j, v_k) \,|\, 1 \leq j \leq (n-1),\, (j+1) \leq k \leq n\}\,. \end{aligned}$$

There are thus

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

edges in total: see Figure 1.7 for the first few examples.

The path graphs P_n

These graphs are formed by stringing n vertices together in a path. The word "path" actually has a technical meaning in graph theory, but you needn't worry about that

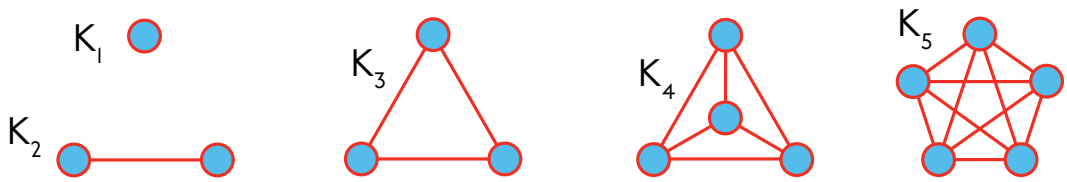


Figure 1.7: *The first five members of the family K_n of complete graphs.*

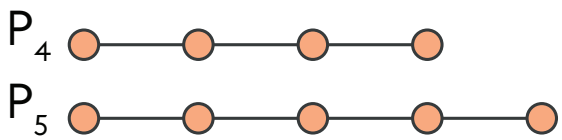


Figure 1.8: *Diagrams for the path graphs P_4 and P_5.*

today. P_n has vertex and edge sets as listed below,

$$\begin{aligned} V &= \{v_1,\, v_2,\, \ldots,\, v_n\} \\ E &= \{(v_j, v_{j+1}) \,|\, 1 \le j < n\}\,, \end{aligned}$$

and Figure 1.8 shows two examples.

The cycle graphs C_n

The *cycle graph* C_n, sometimes also called the *circuit graph*, is a graph in which $n \ge 3$ vertices are arranged in a ring. If one numbers the vertices consecutively the edge and vertex set are

$$\begin{aligned} V &= \{v_1,\, v_2,\, \ldots,\, v_n\} \\ E &= \{(v_1, v_2),\, (v_2, v_3),\, \ldots,\, (v_j, v_{j+1}),\, \ldots,\, (v_{n-1}, v_n),\, (v_n, v_1)\}\,. \end{aligned}$$

C_n has n edges that are often written (v_j, v_{j+1}), where the subscripts are taken to be defined periodically so that, for example, $v_{n+1} \equiv v_1$. See Figure 1.9 for examples.

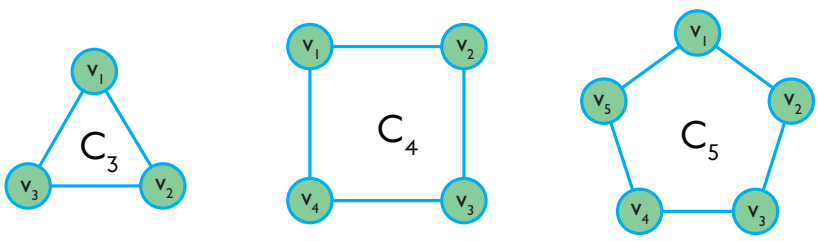


Figure 1.9: *The first three members of the family C_n of cycle graphs.*

The complete bipartite graphs $K_{m,n}$

The *complete bipartite graph* $K_{m,n}$ is a graph whose vertex set is the union of a set V_1 of m vertices with second set V_2 of n different vertices and whose edge set includes every possible edge running between these two subsets:

$$\begin{aligned} V &= V_1 \cup V_2 \\ &= \{u_1, \ldots, u_m\} \cup \{v_1, \ldots, v_n\} \\ E &= \{(u,v) \,|\, u \in V_1,\, v \in V_2\}\,. \end{aligned}$$

$K_{m,n}$ thus has $|E| = mn$ edges: see Figure 1.10 for examples.

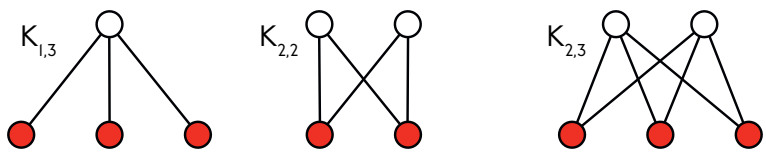


Figure 1.10: *A few members of the family $K_{m,n}$ of complete bipartite graphs. Here the two subsets of the vertex set are illustrated with colour: the white vertices constitute V_1, while the red ones form V_2.*

There are other sorts of bipartite graphs too:

Definition 1.6. *A graph $G(V,E)$ is said to be a* ***bipartite graph*** *if*

- *it has a nonempty edge set: $E \neq \emptyset$ and*
- *its vertex set V can be decomposed into two nonempty, disjoint subsets*

  $$V = V_1 \cup V_2 \;\text{ with }\; V_1 \cap V_2 = \emptyset \;\text{ and }\; V_1 \neq \emptyset \;\text{ and }\; V_2 \neq \emptyset$$

  in such a way that all the edges $(u,v) \in E$ contain a member of V_1 and a member of V_2.

The cube graphs I_d

These graphs are specified in a way that's closer to the purely combinatorial, set-theoretic definition of a graph given above. I_d, the *d-dimensional cube graph*, has vertices that are strings of d zeroes or ones, and all possible labels occur. Edges connect those vertices whose labels differ in exactly one position. Thus, for example, I_2 has vertex and edge sets

$$V = \{00,\, 01,\, 10,\, 11\} \qquad \text{and} \qquad E = \{(00,01),\, (00,10),\, (01,11),\, (10,11)\}\,.$$

Figure 1.11 shows diagrams for the first few cube graphs and these go a long way toward explaining the name. More generally, I_d has vertex and edge sets given by

$$\begin{aligned} V &= \left\{w \,|\, w \in \{0,1\}^d\right\} \\ E &= \{(w,w') \,|\, w \text{ and } w' \text{ differ in a single position}\}\,. \end{aligned}$$

This means that I_d has $|V| = 2^d$ vertices, but it's a bit harder to count the edges. In the last part of today's lecture we'll prove a theorem that enables one to show that I_d has $|E| = d\,2^{d-1}$ edges.

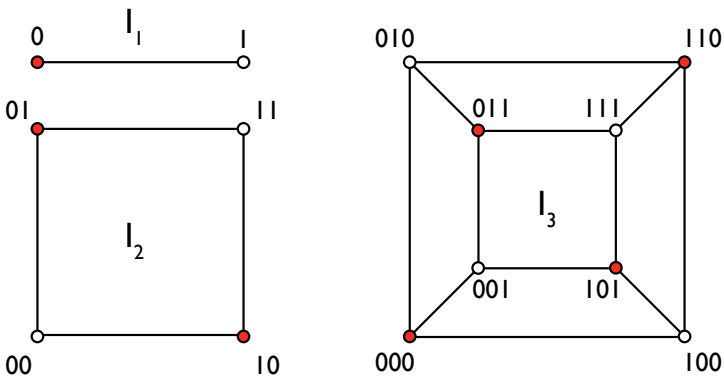


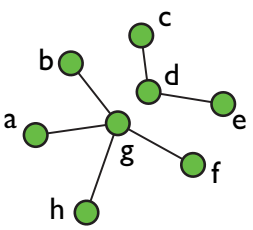
Figure 1.11: *The first three members of the family I_d of cube graphs. Notice that all the cube graphs are bipartite (the red and white vertices are the two disjoint subsets from Definition 1.6), but that, for example, I_3 is* not *a complete bipartite graph.*

1.4 A first theorem about graphs

I find it wearisome to give, or learn, one damn definition after another and so I'd like to conclude the lecture with a small, but useful theorem. To do this we need one more definition:

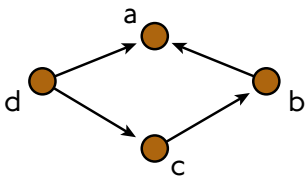
Definition 1.7. *In an undirected graph $G(V, E)$ the* ***degree*** *of a vertex $v \in V$ is the number of edges that include the vertex. One writes* $\deg(v)$ *for "the degree of v".*

So, for example, every vertex in the complete graph K_n has degree $n-1$, while every vertex in a cycle graph C_n has degree 2; Figure 1.12 provides more examples. The generalization of degree to directed graphs is slightly more involved. A vertex v in a digraph has two degrees: an *in-degree* that counts the number of edges having v at their tip and an *out-degree* that counts number of edges having v at their tail. See Figure 1.13 for an example.



v	a	b	c	d	e	f	g	h
$\deg(v)$	1	1	1	2	1	1	4	1

Figure 1.12: *The degrees of the vertices in a small graph. Note that the graph consists of two "pieces".*



v	$\deg_{in}(v)$	$\deg_{out}(v)$
a	2	0
b	1	1
c	1	1
d	0	2

Figure 1.13: *The degrees of the vertices in a small digraph.*

Once we have the notion of degree, we can formulate our first theorem:

Theorem 1.8 (Handshaking Lemma, Euler 1736)**.** *If $G(V, E)$ is an undirected graph then*

$$\sum_{v \in V} \deg(v) = 2|E|. \tag{1.1}$$

Proof. Each edge contributes twice to the sum of degrees, once for each of the two vertices on which it is incident. □

The following two results are immediate consequences:

Corollary 1.9. *In an undirected graph there must be an even number of vertices that have odd degree.*

Corollary 1.10. *The cube graph I_d has $|E| = d\,2^{d-1}$.*

The first is fairly obvious: the right hand side of (1.1) is clearly an even number, so the sum of degrees appearing on the left must be even as well. To get the formula for the number of edges in I_d, note that it has 2^d vertices, each of degree d, so the Handshaking Lemma tells us that

$$2|E| \;=\; \sum_{v \in V} \deg(v) \;=\; 2^d \times d$$

and thus $|E| = (d \times 2^d)/2 = d\,2^{d-1}$.

Lecture 2

Representation, Sameness and Parts

Reading: Some of the material in today's lecture comes from the beginning of Chapter 1 in

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, which is available online via SpringerLink.

If you are at the university, either physically or via the VPN, you can download the chapters of this book as PDFs.

2.1 Ways to represent a graph

The first part of this lecture is concerned with various ways of specifying a graph. It may seem unnecessary to have so many different descriptions for a mathematical object that is, fundamentally, just a pair of finite sets, but each of the representations below will prove convenient when we are developing algorithms (step-by-step computational recipes) to solve problems involving graphs.

2.1.1 Edge lists

From the first lecture, we already how to represent a graph $G(V, E)$ by specifying its vertex set V and its *edge list* E as, for example,

Example 2.1 (Edge list representation)**.** *The undirected graph $G(V, E)$ with*

$$V = \{1, 2, 3\} \qquad \textit{and} \qquad E = \{(1, 2),\ (2, 3),\ (1, 3)\}$$

is K_3, the complete graph on three vertices. But if we regard the edges as directed then G is the graph pictured at the right of Figure 2.1

Of course, if every vertex in $G(V, E)$ appears in some edge (equivalently, if every vertex has nonzero degree), then we can dispense with the vertex set and specify the graph by its edge list alone.

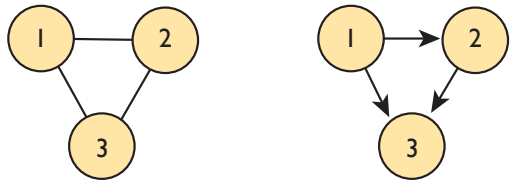


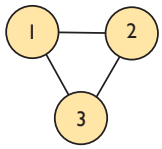
Figure 2.1: *If graph from Example 2.1 is regarded as undirected (our default assumption) then it is K_3, the complete graph on three vertices, but if it's directed then it's the digraph at right above.*

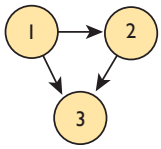
2.1.2 Adjacency matrices

A second approach is to give an *adjacency matrix*, often written A. One builds an adjacency matrix by first numbering the vertices, so that the vertex set becomes $V = \{v_1, v_2, \ldots, v_n\}$ for a graph on n vertices. The adjacency matrix A is then an $n \times n$ matrix whose entries are given by the following rule:

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Once again, the directed and undirected cases are different. For the graphs from Example 2.1 we have:

if G is  then $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$,

but if G is  then $A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$.

Remark 2.2. *The following properties of the adjacency matrix follow readily from the definition in Eqn. (2.1).*

- *The adjacency matrix is not unique because it depends on a numbering scheme for the vertices. If one renumbers the vertices, the rows and columns of A will be permuted accordingly.*
- *If $G(V, E)$ is undirected then its adjacency matrix A is symmetric. That's because we think of the edges as unordered pairs, so, for example, $(1, 2) \in E$ is the same thing as $(2, 1) \in E$.*
- *If the graph has no loops then $A_{jj} = 0$ for $1 \leq j \leq n$. That is, there are zeroes down the main diagonal of A.*

- *One can compute the degree of a vertex by adding up entries in the adjacency matrix. I leave it as an exercise for the reader to establish that in an undirected graph,*

$$\deg(v_j) = \sum_{k=1}^{n} A_{jk} = \sum_{k=1}^{n} A_{kj}, \tag{2.2}$$

where the first sum runs across the j-th row, while the second runs down the j-th column. Similarly, in a directed graph we have

$$\deg_{out}(v_j) = \sum_{k=1}^{n} A_{jk} \qquad \textit{and} \qquad \deg_{in}(v_j) = \sum_{k=1}^{n} A_{kj}. \tag{2.3}$$

- *Sometimes one sees a modified form of the adjacency matrix used to describe multigraphs (graphs that permit two or more edges between a given pair of vertices). In this case one takes*

$$A_{ij} \;=\; \textit{number of times the edge } (i,j) \textit{ appears in } E \tag{2.4}$$

2.1.3 Adjacency lists

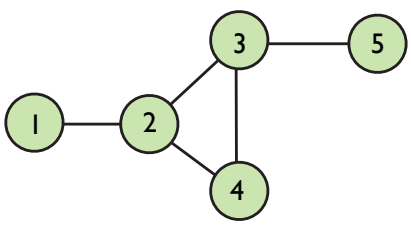
One can also specify an undirected graph by giving the *adjacency lists* of all its vertices.

Definition 2.3. *In an undirected graph $G(V,E)$ the* ***adjacency list*** *associated with a vertex v is the set $A_v \subseteq V$ defined by*

$$A_v = \{u \in V \mid (u,v) \in E\}.$$

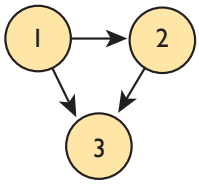
An example appears in Figure 2.2. It follows readily from the definition of degree that

$$\deg(v) = |A_v|. \tag{2.5}$$



$A_1 = \{2\}$
$A_2 = \{1,\ 3,\ 4\}$
$A_3 = \{2,\ 4,\ 5\}$
$A_4 = \{2,\ 3\}$
$A_5 = \{3\}$

Figure 2.2: *The graph at left has adjacency lists as shown at right.*



v	Predecessors	Successors
1	$\emptyset$	$\{2, 3\}$
2	$\{1\}$	$\{3\}$
3	$\{1, 2\}$	$\emptyset$

Figure 2.3: *The directed graph at left has the predecessor and successor lists shown at right.*

Similarly, one can specify a directed graph by providing separate lists of *successors* or *predecessors* (these terms were defined in Lecture 1) for each vertex.

Definition 2.4. *In an directed graph $G(V, E)$ the* ***predecessor list*** *of a vertex v is the set $P_v \subseteq V$ defined by*

$$P_v = \{u \in V \mid (u, v) \in E\}$$

while the ***successor list*** *of v is the set $S_v \subseteq V$ defined by*

$$S_v = \{u \in V \mid (v, u) \in E\}.$$

Figure 2.3 gives some examples. The analogues of Eqn. (2.5) for a directed graph are

$$\deg_{in}(v) = |P_v| \qquad \text{and} \qquad \deg_{out}(v) = |S_v|. \tag{2.6}$$

2.2 When are two graphs the same?

For the small graphs that appear in these notes, it's usually fairly obvious when two of them are the same. But in general it's nontrivial to be rigorous about what we mean when we say two graphs are "the same". The point is that if we stick to the abstract definition of a graph-as-two-sets, we need to formulate our definition of sameness in a similar style. Informally we'd like to say that two graphs are the same (we'll use the term *isomorphic* for this) if it is possible to relabel the vertex sets in such a way that their edge sets match up. More precisely:

Definition 2.5. *Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are said to be* ***isomorphic*** *if there exists a bijection*[1] *$\alpha : V_1 \to V_2$ such that the edge $(\alpha(a), \alpha(b)) \in E_2$ if and only if $(a, b) \in E_1$.*

Generally it's difficult to decide whether two graphs are isomorphic. In particular, there are no known fast algorithms[2] (we'll learn to speak more precisely about what it means for an algorithm to be "fast" later in the term) to decide. One can,

[1] Recall that a bijection is a mapping that's one-to-one and onto.

[2] Algorithms for graph isomorphism are the subject of intense current research: see Erica Klarreich's Jan. 2017 article in *Quanta Magazine*, `Complexity Theory Problem Strikes Back`, for a popular account of some recent results.

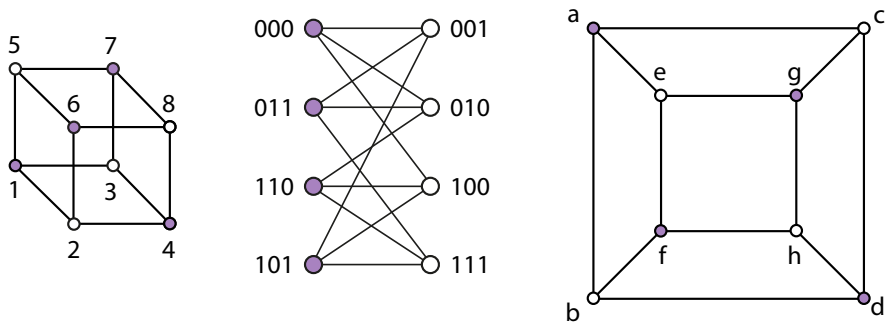


Figure 2.4: *Here are three different graphs that are all isomorphic to the cube graph I_3, which is the middle one. The bijections that establish the isomorphisms are listed in Table 2.1.*

v	000	001	010	011	100	101	110	111
$\alpha_L(v)$	1	2	3	4	5	6	7	8
$\alpha_R(v)$	a	b	c	d	e	f	g	h

Table 2.1: *If we number the graphs in Figure 2.4 so that the leftmost is $G_1(V_1, E_1)$ and the rightmost is $G_3(V_3, E_3)$, then the bijections $\alpha_L : V_2 \to V_1$ and $\alpha_R : V_2 \to V_3$ listed above establish that G_2 is isomorphic, respectively, to G_1 and G_3.*

of course, simply try all possible bijections between the two vertex sets, but there are $n!$ of these for graphs on n vertices and so this brute force approach rapidly becomes impractical. On the other hand, it's often possible to detect quickly that two graphs *aren't* isomorphic. The simplest such tests are based on the following propositions, whose proofs are left to the reader.

Proposition 2.6. *If $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are isomorphic then $|V_1| = |V_2|$ and $|E_1| = |E_2|$.*

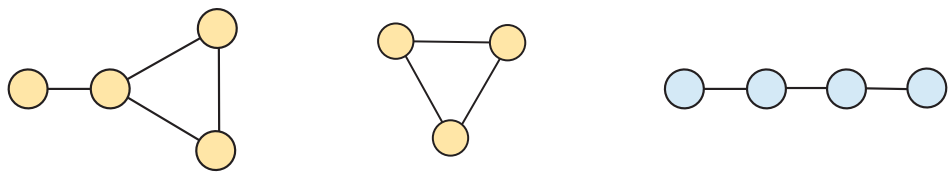
Proposition 2.7. *If $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are isomorphic and $\alpha : V_1 \to V_2$ is the bijection that establishes the isomorphism, then $\deg(v) = \deg(\alpha(v))$ for every $v \in V_1$ and $\deg(u) = \deg(\alpha^{-1}(u))$ for every $u \in V_2$.*

Another simple test depends on the following quantity, examples of which appear in Figure 2.5.

Definition 2.8. *The **degree sequence** of an undirected graph $G(V, E)$ is a list of the degrees of the vertices, arranged in ascending order.*

The corresponding test for non-isomorphism depends on the following proposition, whose proof is left as an exercise.

Proposition 2.9. *If $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are isomorphic then they have the same degree sequence.*



$(1, 2, 2, 3)$ $(2, 2, 2)$ $(1, 1, 2, 2)$

Figure 2.5: *Three small graphs and their degree sequences.*

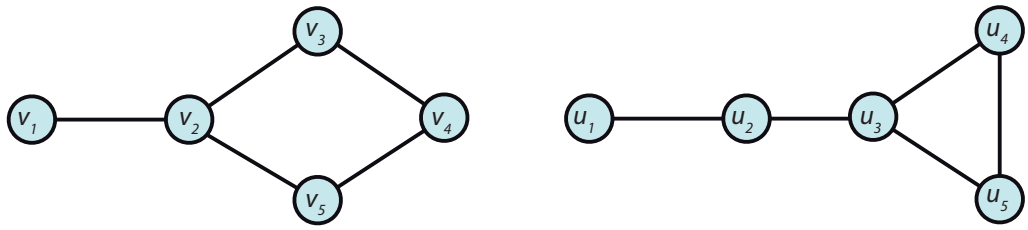


Figure 2.6: *These two graphs both have degree sequence* $(1, 2, 2, 2, 3)$*, but they're not isomorphic: see Example 2.10 for a proof.*

Unfortunately although it's a necessary condition for two isomorphic graphs to have the same degree sequence, a shared degree sequence isn't sufficient to establish isomorphism. That is, it's possible for two graphs to have the same degree sequence, but not be isomorphic: Figure 2.6 shows one such pair, but it's easy to make up more.

Example 2.10 (Proof that the graphs in Figure 2.6 aren't isomorphic)**.** *Both graphs in Figure 2.6 have the same degree sequence,* $(1, 2, 2, 2, 3)$*, so both contain a single vertex of degree 1 and a single vertex of degree 3. These vertices are adjacent in the graph at left, but not in the one at right and this observation forms the basis for a proof by contradiction that the graphs aren't isomorphic.*

Assume, for contradiction, that they are isomorphic and that

$$\alpha : \{v_1,\, v_2,\, v_3,\, v_4,\, v_5\} \to \{u_1,\, u_2,\, u_3,\, u_4,\, u_5\}$$

is the bijection that establishes the isomorphism. Then Prop. 2.7 implies that it must be true that $\alpha(v_1) = u_1$ *(as these are the sole vertices of degree one) and* $\alpha(v_2) = u_3$*. But then the presence of the edge* (v_1, v_2) *on the left would imply the existence of an edge* $(\alpha(v_1), \alpha(v_2)) = (u_1, u_3)$ *on the right, and no such edge exists. This contradicts our assumption that* α *establishes an isomorphism, so no such* α *can exist and the graphs aren't isomorphic.*

2.3 Terms for parts of graphs

Finally, we'll often want to speak of parts of graphs and the two most useful definitions here are:

Definition 2.11. *A* ***subgraph*** *of a graph* $G(V, E)$ *is a graph* $G'(V', E')$ *where* $V' \subseteq V$ *and* $E' \subseteq E$.

and

Definition 2.12. *Given a graph* $G(V, E)$ *and a subset of its vertices* $V' \subseteq V$*, the* ***subgraph induced by*** V' *is the subgraph* $G'(V', E')$ *where*

$$E' = \{(u, v) \mid u, v \in V' \text{ and } (u, v) \in E\}.$$

That is, the subgraph induced by the vertices V' consists of V' itself and *all* those edges in the original graph that involve only vertices from V'. Both these definitions are illustrated in Figure 2.7.

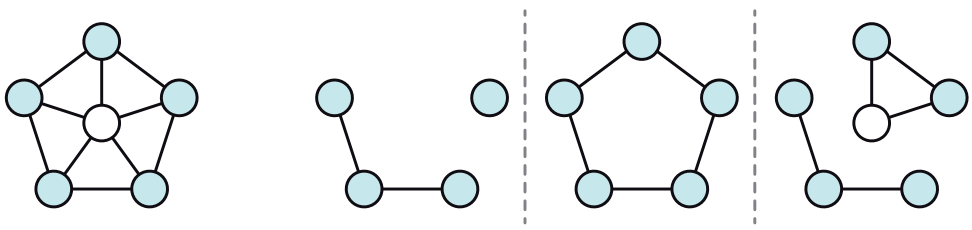


Figure 2.7: *The three graphs at right are subgraphs of the one at left. The middle one is the subgraph induced by the blue shaded vertices.*

Lecture 3

Graph Colouring

The material from the first two lectures provides enough background that we can begin to discuss a problem—graph colouring—that is both mathematically rich and practically applicable.

Reading:
The material for this lecture appears, in very condensed form, in Chapter 9 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition (Available from SpringerLink)

A somewhat longer discussion, with many interesting exercises, appears in

> John M. Harris, Jeffry L. Hirst and Michael J. Mossinghoff (2008), *Combinatorics and Graph Theory*, 2nd edition. (Available from SpringerLink)

3.1 Notions and notation

Definition 3.1. *A k-**colouring** of an undirected graph $G(V, E)$ is a function*

$$\phi : V \to \{1, \ldots, k\}$$

*that assigns distinct values to adjacent vertices: that is, $(u, v) \in E \Rightarrow \phi(u) \neq \phi(v)$. If G has a k-colouring then it is said to be k-**colourable**.*

I'll refer to the values assigned by $\phi(v)$ as "colours" and say that a graph is k-colourable if one can draw it in such a way that no two adjacent vertices have the same colour. Examples of graphs and colourings include

- K_n, the complete graph on n vertices is clearly n-colourable, but not $(n-1)$ colourable;
- $K_{m,n}$, the complete bipartite graph on groups of m and n vertices, is 2-colourable.

Both classes of example are illustrated in Figure 3.1.

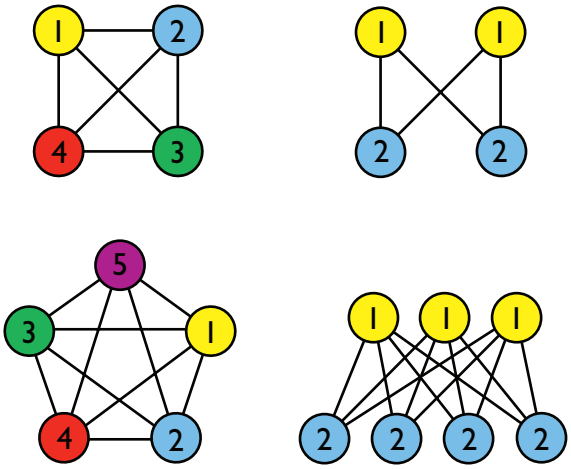


Figure 3.1: *The complete graphs K_4 and K_5 as well as the complete bipartite graphs $K_{2,2}$ and $K_{3,4}$, each coloured using the smallest possible number of colours. Here the colouring is represented in two ways: as numbers giving $\phi(v)$ for each vertex v and with, well, colours (see the electronic version).*

Definition 3.2. *The* ***chromatic number*** $\chi(G)$ *is the smallest number k such that G is k-colourable.*

Thus, as the examples above suggest, $\chi(K_n) = n$ and $\chi(K_{m,n}) = 2$. The latter is a special case of the following easy lemma, whose proof is left as an exercise.

Lemma 3.3. *A graph G has chromatic number $\chi(G) = 2$ if and only if it is bipartite.*

Another useful result is

Lemma 3.4. *If H is a subgraph of G and G is k-colourable, then so is H.*

and an immediate corollary is

Lemma 3.5. *If H is a subgraph of G then $\chi(H) \leq \chi(G)$.*

which comes in handy when trying to prove that a graph has a certain chromatic number.

The proof of Lemma 3.4 is straightforward: the idea is that constraints on the colours of vertices arise from edges and so, as every edge in H is also present in G, it can't be any harder to colour H than it is to colour G. Equivalently: if we have a colouring of G and want a colouring of H we can simply use the same colour assignments. To be more formal, say that the vertex and edge sets of G are V and E, respectively, while those of H are $V' \subseteq V$ and $E' \subseteq E$. If a map $\phi_G : V \to \{1, \ldots, k\}$ is a k-colouring of G, then $\phi_H : V' \to \{1, \ldots, k\}$ defined as the restriction of ϕ_G to V' produces a k-colouring of H.

Lemma 3.5 then follows from the observation that, although Lemma 3.4 assures us that H has a colouring that uses $\chi(G)$ colours, it may also be possible to find some other colouring that uses fewer.

3.2 An algorithm to do colouring

The chromatic number $\chi(G)$ is defined as a kind of ideal: it's the minimal k for which we can find a k-colouring. This might make you suspect that it's hard to find $\chi(G)$ for an arbitrary graph—how could you ever know that you'd used the smallest possible number of colours? And, aside from a few exceptions such as those in the previous section, you'd be right to think this: there is no known fast (we'll make the notion of "fast" more precise soon) algorithm to find an optimal (in the sense of using the smallest number of colours) colouring.

3.2.1 The greedy colouring algorithm

There is, however, a fairly easy way to to compute a (possibly non-optimal) colouring $c : V \to \mathbb{N}$. The idea is to number the vertices and then, starting with $c(v_1) = 1$, visit the remaining vertices in order, assigning them the lowest-numbered colour not yet used for a neighbour. The algorithm is called *greedy* because it has no sense of long-range strategy: it just proceeds through the list of vertices, blindly choosing the colour that seems best at the moment.

Algorithm 3.6 (Greedy colouring)**.**
Given a graph G with edge set E, vertex set $V = \{v_1, \ldots, v_n\}$ and adjacency lists A_v, construct a function $c : V \to \mathbb{N}$ such that if the edge $e = (v_i, v_j) \in E$, then $c(v_i) \neq c(v_j)$.

(1) Initialize
Set $c(v_j) \leftarrow 0$ *for all* $1 \leq j \leq n$
$c(v_1) \leftarrow 1$
$j \leftarrow 2$

(2) $c(v_j) \leftarrow \min\big(k \in \mathbb{N} \,|\, k > 0$ *and* $c(u) \neq k \ \forall u \in A_{v_j}\big)$

(3) Are we finished? Is $j = n$?

- *If so, stop: we've constructed a function c with the desired properties.*
- *If not, set $j \leftarrow (j + 1)$ and go to step (2).*

Remarks

- The algorithm above is meant to be explicit enough that one could implement it in R or MATLAB. It thus includes expressions such as $j \leftarrow 2$ which means "set j to 2" or "j gets the (new) value 2". The operator $\leftarrow$ is sometimes called the *assignment* operator and it appears in some form in all the programming languages I know. Sometimes it's expressed with notation like `j = j + 1`, but this is a jarring, nonsensical-looking thing for a mathematician and so I'll avoid it.

- We will discuss several more algorithms in this course, but will not be much more formal about how they are specified. This is mainly because a truly rigorous account of computation would take us into the realms of computability theory, a part of mathematical logic, and would require much of the rest of the term, leaving little time for our main subjects.

Finally, to emphasize further the mechanical nature of greedy colouring, we could rewrite it in a style that looks even closer to MATLAB code:

Algorithm 3.7 (Greedy colouring: as pseudo-code)**.**
Given a graph G with edge set E, vertex set $V = \{v_1, \ldots, v_n\}$ and adjacency lists A_v, construct a function $c : V \to \mathbb{N}$ such that if the edge $e = (v_i, v_j) \in E$, then $c(v_i) \neq c(v_j)$.

(1) Set $c(v_j) \leftarrow 0$ for $1 \leq j \leq n$.

(2) $c(v_1) \leftarrow 1$.

(3) for $2 \leq j \leq n$ {

(4) Choose a colour $k > 0$ for vertex v_j that differs from those of its neighbours
$c(v_j) \leftarrow \min\big(k \in \mathbb{N} \,|\, k > 0 \text{ and } c(u) \neq k \ \forall u \in A_{v_j}\big)$

(5) } End of loop over vertices v_j.

Both versions of the algorithm perform exactly the same steps, in the same order, so comparison of these two examples may clarify the different approaches to presenting algorithms.

3.2.2 Greedy colouring may use too many colours

If we use Algorithm 3.7 to construct a function $c : V \to \mathbb{N}$, then we can regard it as a k-colouring by setting $\phi(v_j) = c(v_j)$, where k is given by

$$k = \max_{v_j \in V} c(v_j). \tag{3.1}$$

For the reasons discussed above, this k provides only an upper bound on the chromatic number of G. To drive this point home, consider Figure 3.2, which illustrates the process of applying the greedy colouring algorithm to two graphs, one in each column.

For the graph in the left column—call it G_1—the algorithm produces a 3-colouring, which is actually optimal. To see why, notice that the subgraph consisting of vertices v_1, v_2 and v_3 (along with the associated edges) is isomorphic to K_3. Thus we need at least 3 colours for these three vertices and so, using Lemma 3.5, we can conclude that $\chi(G_1) \geq 3$. On the other hand, the greedy algorithm provides an explicit example of a 3-colouring, which implies that $\chi(G_1) \leq 3$, so we have proven that $\chi(G_1) = 3$.

The graph in the right column–call it G_2—is isomorphic to G_1 (a very keen reader could write out the isomorphism explicitly), but its vertices are numbered differently and this means that Algorithm 3.7 colours them in a different order and arrives at a sub-optimal k-colouring with $k = 4$.

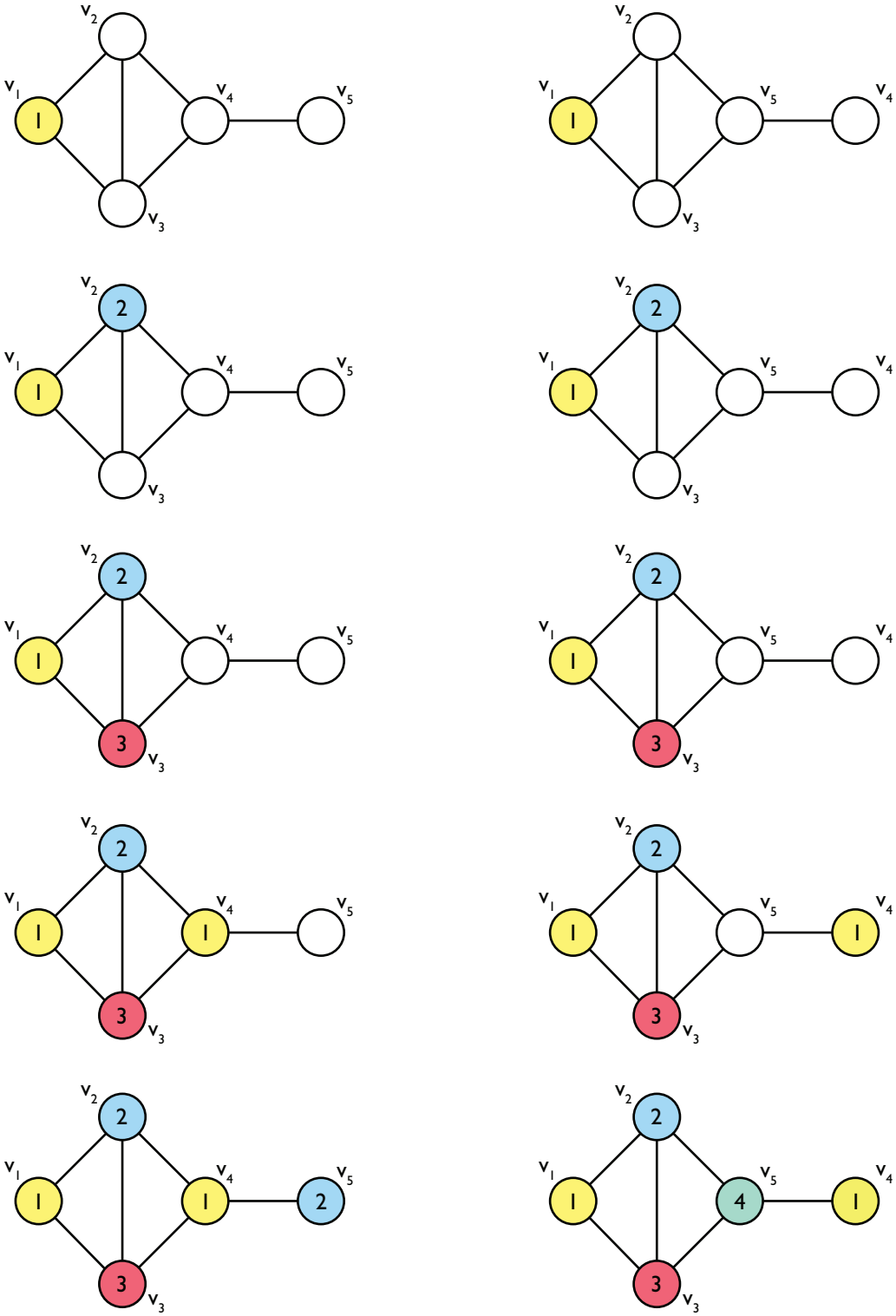


Figure 3.2: *Two examples of applying Algorithm 3.7: the colouring process runs from the top of a column to the bottom. The graphs in the right column are the same as those in the left, save that the labels on vertices 4 and 5 have been switched. As in Figure 3.1, the colourings are represented both numerically and graphically.*

3.3 An application: avoiding clashes

I'd like to conclude by introducing a family of applications that involve avoiding some sort of clash—where some things shouldn't be allowed to happen at the same time or in the same place. A prototypical example is:

Example 3.8. *Suppose that a group of ministers serve on committees as described below:*

Committee	***Members***
Culture, Media & Sport	*Alexander, Burt, Clegg*
Defence	*Clegg, Djanogly, Evers*
Education	*Alexander, Gove*
Food & Rural Affairs	*Djanogly, Featherstone*
Foreign Affairs	*Evers, Hague*
Justice	*Burt, Evers, Gove*
Technology	*Clegg, Featherstone, Hague*

What is the minimum number of time slots needed so that one can schedule meetings of these committees in such a way that the ministers involved have no clashes?

One can turn this into a graph-colouring problem by constructing a graph whose vertices are committees and whose edges connect those that have members in common: such committees can't meet simultaneously, or their shared members will have clashes. A suitable graph appears at left in Figure 3.3, where, for example, the vertex for the Justice committee (labelled Just) is connected to the one for the Education committee (Ed) because Gove serves on both.

The version of the graph at right in Figure 3.3 shows a three-colouring and, as the vertices CMS, Ed and Just form a subgraph isomorphic to K_3, this is the smallest number of colours one can possibly use and so the chromatic number of the committee-and-clash graph is 3. This means that we need at least three time slots to schedule the meetings. To see why, think of a vertex's colour as a time slot: none of the vertices that receive the same colour are adjacent, so none of the corresponding committees share any members and thus that whole group of committees can be scheduled to meet at the same time. There are variants of this problem that involve, for example, scheduling exams so that no student will be obliged to be in two places at the same time or constructing sufficiently many storage cabinets in a lab so that chemicals that would react explosively if stored together can be housed separately: see this week's Problem Set for another example.

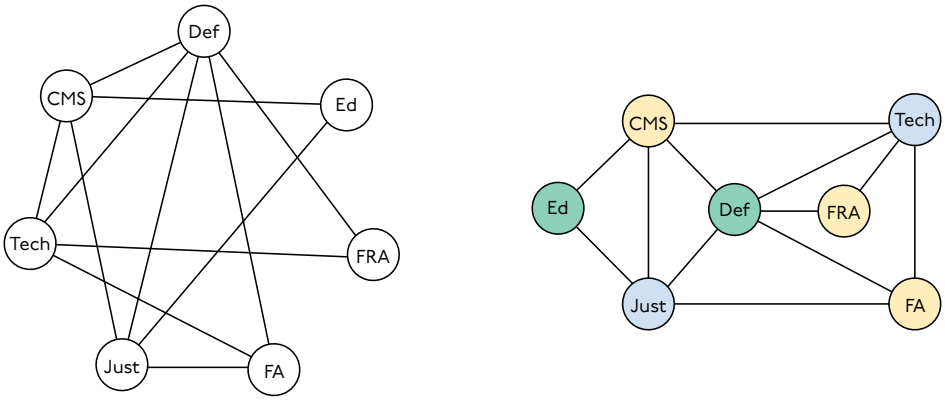


Figure 3.3: *The graph at left has vertices labelled with abbreviated committee names and edges given by shared members. The graph at right is isomorphic, but has been redrawn for clarity and given a three-colouring, which turns out to be optimal.*

Lecture 4

Efficiency of algorithms

The development of algorithms (systematic recipes for solving problems) will be a theme that runs through the whole course. We'll be especially interested in saying rigorous-but-useful things about how much work an algorithm requires. Today's lecture introduces the standard vocabulary for such discussions and illustrates it with several examples including the Greedy Colouring Algorithm, the standard algorithm for multiplication of two square $n \times n$ matrices and the problem of testing whether a number is prime.

Reading:
The material in the first part of today's lecture comes from Section 2.5 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition (available online via SpringerLink),

though the discussion there mentions some graph-theoretic matters that we have not yet covered.

4.1 Introduction

The aim of today's lecture is to develop some convenient terms for the way in which the amount of work required to solve a problem with a particular algorithm depends on the "size" of the input. Note that this is a property of the *algorithm*: it may be possible to find a better approach that solves the problem with less work. If we were being very careful about these ideas we would make a distinction between the quantities I'll introduce below, which, strictly speaking, describe the *time complexity* of an algorithm, and a separate set of bounds that say how much computer memory (or how many sheets of paper, if we're working by hand) an algorithm requires. This latter quantity is called the *space complexity* of the algorithm, but we won't worry about that much in this course.

To get an idea of the kinds of results we're aiming for, recall the standard algorithm for pencil-and-paper addition of two numbers (write one number above the other, draw a line underneath ...).

$$\begin{array}{r} 2011 \\ 21 \\ \hline 2032 \end{array}$$

The basic step in this process is the addition of two decimal digits, for example, in the first column, $1 + 1 = 2$. The calculation here thus requires two basic steps.

More generally, the number of basic steps required to perform an addition $a + b$ using the pencil-and-paper algorithm depends on the numbers of decimal digits in a and b. The following proposition (whose proof is left to the reader) explains why it's thus natural to think of $\log_{10}(a)$ and $\log_{10}(b)$ as the sizes of the inputs to the addition algorithm.

Definition 4.1 (Floor and ceiling)**.** *For a real number $x \in \mathbb{R}$, define $\lfloor x \rfloor$, which is read as "floor of x", to be the greatest integer less-than-or-equal-to x. Similarly, define $\lceil x \rceil$, "ceiling of x", to be the least integer greater-than-or-equal-to x. In more conventional notation these functions are given by*

$$\lfloor x \rfloor = \max \{k \in \mathbb{Z} \,|\, k \leq x\} \quad \textit{and} \quad \lceil x \rceil = \min \{k \in \mathbb{Z} \,|\, k \geq x\}\,.$$

Proposition 4.2 (Logs and length in decimal digits)**.** *The decimal representation of a number $n > 0 \in \mathbb{N}$ has exactly $d = 1 + \lfloor \log_{10}(n) \rfloor$ decimal digits.*

In light of these results, one might hope to say something along the lines of

> *The pencil-and-paper addition algorithm computes the sum $a + b$ in $1 + \min\left(\lfloor \log_{10}(a) \rfloor, \lfloor \log_{10}(b) \rfloor\right)$ steps.*

This is a bit of a mouthful and, worse, isn't even right. Quick-witted readers will have noticed that we haven't taken proper account of carried digits. The example above didn't involve any, but if instead we had computed

$$\begin{array}{r} 1959 \\ 21 \\ \hline 1980 \end{array}$$

we would have carried a 1 from the first column to the second and so would have needed to do three basic steps: $9 + 1$, $1 + 5$, and $6 + 2$.

In general, if the larger of a and b has d decimal digits then computing $a + b$ could require as many as $d - 1$ carrying additions. That means our statement above should be replaced by something like

> *The number of steps N required for the pencil-and-paper addition algorithm to compute the sum $a + b$ satisfies the following bounds:*
>
> $$1 + \min\left(\lfloor \log_{10}(a) \rfloor, \lfloor \log_{10}(b)) \rfloor\right) \leq N \leq 1 + \lfloor \log_{10}(a) \rfloor + \lfloor \log_{10}(b) \rfloor$$

This introduces an important theme: knowing the size of the input isn't always enough to determine exactly how long a computation will take, but it may be possible to place bounds on the running-time.

The statements above are rather fiddly and not especially useful. In practice, people want such estimates so they can decide whether a problem is do-able at all. They want to know whether, say, given a calculator that can add two 5-digit numbers in one second[1], it would be possible to work out the University of Manchester's payroll in less than a month. For these sorts of questions[2] one doesn't want the cumbersome, though precise sorts of statements formulated above, but rather something semi-quantitative along the lines of:

> *If we measure the size of the inputs to the pencil-and-paper addition algorithm in terms of the number of digits, then if we double the size of the input, the amount of work required to get the answer also doubles.*

The remainder of today's lecture will develop a rigorous framework that we can use to make such statements.

4.2 Examples and issues

The three main examples in the rest of the lecture will be

- Greedy colouring of a graph $G(V, E)$ with $|V|$ vertices and $|E|$ edges.
- Multiplication of two $n \times n$ matrices A and B.
- Testing whether a number $n \in \mathbb{N}$ is prime by trial division.

In the rest of this section I'll discuss the associated algorithms briefly, with an eye to answering three key questions:

(1) What are the details of the algorithm and what should we regard as the basic step?

(2) How should we measure the size of the input?

(3) What kinds of estimates can we hope to make?

4.2.1 Greedy colouring

Algorithm 3.7 constructs a colouring for a graph $G(V, E)$ by examining, in turn, the adjacency lists of each of the vertices in $V = \{v_1, \ldots, v_n\}$. If we take as our

[1] Up until the mid-1940's, a calculator was a *person* and so the speed mentioned here was not unrealistic. Although modern computing machinery doesn't work with decimal representations and can perform basic operations in tiny fractions of a second, similar reasoning still allows one to determine the limits of practical computability.

[2] Estimates similar to the ones we've done here bear on much less contrived questions such as: if there are processors that can do arithmetic on two 500-digit numbers in a nanosecond, how many of them would GCHQ need to buy if they want to be able to crack a 4096-bit RSA cryptosystem within an hour?

basic step the process of looking at neighbour's colour, then the algorithm requires a number of steps given by

$$\sum_{v \in V} |A_v| \;=\; \sum_{v \in V} \deg(v) \;=\; 2|E| \tag{4.1}$$

where the final equality follows from Theorem 1.8, the Handshaking Lemma. This suggests that we should measure the size of the problem in terms of the number of edges.

4.2.2 Matrix multiplication

If A and B are square, $n \times n$ matrices then the i, j-th entry in the product AB is given by:

$$(AB)_{i,j} = \sum_{k=1}^{n} A_{ik} B_{kj}.$$

Here we'll measure the size of the problem with n, the number of rows in the matrices, and take as our basic steps the arithmetic operations, addition and multiplication of two real numbers. The formula above then makes it easy to see that it takes n multiplications and $(n-1)$ additions to compute a single entry in the product matrix and so, as there are n^2 such entries, we need

$$n^2 \left(n + (n-1)\right) = 2n^3 - n^2 \tag{4.2}$$

total arithmetic operations to compute the whole product.

Notice that I have not included any measure of the magnitude of the matrix entries in our characterisation of the size of the problem. I did this because (a) it agrees with the usual practice among numerical analysts, who typically analyse algorithms designed to work with numbers whose machine representations have a fixed size and (b) I wanted to emphasise that an efficiency estimate depends in detail on how one chooses to measure size of the inputs. The final example involves a problem and algorithm where it does make sense to think about the magnitude of the input.

4.2.3 Primality testing and worst-case estimates

It's easy to prove by contradiction the following proposition.

Proposition 4.3 (Smallest divisor of a composite number)**.** *If $n \in \mathbb{N}$ is composite (that is, not a prime), then it has a divisor b that satisfies $b \leq \sqrt{n}$.*

We can thus test whether a number is prime with the following simple algorithm:

Algorithm 4.4 (Primality testing via trial division)**.**
Given a natural number $n \in \mathbb{N}$, determine whether it is prime.

(1) For each $b \in \mathbb{N}$ in the range $2 \leq b \leq \sqrt{n}$ {

(2) Ask: does b divide n?
If Yes, report that n is composite.
If No, continue to the next value of b.

(3) }

(4) If no divisors were found, report that n is prime.

This problem is more subtle than the previous examples in a couple of ways. A natural candidate for the basic step here is the computation of (n mod b), which answers the question "Does b divide n?". But the kinds of numbers whose primality one wants to test in, for example, cryptographic applications are large and so we might want take account of the magnitude of n in our measure of the input size. If we compute (n mod b) with the standard long-division algorithm, the amount of work required for a basic step will itself depend on the number of digits in n and so, as in our analysis of the pencil-and-paper addition algorithm, it'll prove convenient to measure the size of the input with $d = \log_{10}(n)$, which is approximately the number of decimal digits in n.

A further subtlety is that because the algorithm reports an answer as soon as it finds a factor, the amount of work required varies wildly, even among n with the same number of digits. For example, half of all 100-digit numbers are even and so will be revealed as composite by the very first value of b we'll try. Primes, on the other hand, will require $\lfloor\sqrt{n}\rfloor - 1$ tests. A standard way to deal with this second issue is to make estimates about the *worst case* efficiency: in this case, that's the running-time required for primes. A much harder approach is to make an estimate of the *average case* running-time obtained by averaging over all inputs with a given size.

4.3 Bounds on asymptotic growth

As the introduction hinted, were really after statements about how quickly the amount of work increases as the size of the problem does. The following definitions provide a very convenient language in which to formulate such statements.

Definition 4.5 (Bounds on asymptotic growth)**.** *The rate of growth of a function $f : \mathbb{N} \to \mathbb{R}^+$ is often characterised in terms of some simpler function $g : \mathbb{N} \to \mathbb{R}^+$ in the following ways*

- $f(n) = O(g(n))$ *if* $\exists\, c_1 > 0$ *such that, for all sufficiently large* n, $f(n) \leq c_1 g(n)$;
- $f(n) = \Omega(g(n))$ *if* $\exists\, c_2 > 0$ *such that, for all sufficiently large* n, $f(n) \geq c_2 g(n)$;
- $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $f(n) = \Omega(g(n))$.

Notice that the definitions of $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ include the phrase "for all sufficiently large n". This is equivalent to saying, for example,

> $f(n) = \Omega(g(n))$ if there exist some $c_1 > 0$ and $N_1 \geq 0$ such that for all $n \geq N_1$, $f(n) \leq c_1 g(n)$.

The point is that the definitions are only concerned with *asymptotic* bounds—they're all about the limit of large n.

4.4 Analysing the examples

Here I'll apply the definitions from the previous section to our three examples. In practice people are mainly concerned with the O-behaviour of an algorithm. That is, they are mainly interested in getting an asymptotic *upper* bound on the amount of work required. This makes sense when one doesn't know anything special about the inputs and is thinking about buying computer hardware, but as an exercise I'll obtain Θ-type bounds where possible.

4.4.1 Greedy colouring

Proposition 4.6 (Greedy colouring)**.** *If we take the process of checking a neighbour's colour as the basic step, greedy colouring requires $\Theta(|E|)$ basic steps.*

We've already argued that the number of basic steps is $2|E|$, so if we take $c_1 = 3$ and $c_2 = 1$ we have, for all graphs, that the number of operations $f(|E|) = 2|E|$ required for the greedy colouring algorithm satisfies

$$c_2|E| \leq f(|E|) \leq c_1|E| \qquad \text{or} \qquad |E| \leq 2|E| \leq 3|E|$$

and so the algorithm is both $O(|E|)$ and $\Omega(|E|)$ and hence is $\Theta(|E|)$.

4.4.2 Matrix multiplication

Proposition 4.7 (Matrix multiplication)**.** *If we characterise the size of the inputs with n, the number of rows in the matrices, and take as our basic step the arithmetic operations of addition and multiplication of two matrix entries, then matrix multiplication requires $\Theta(n^3)$ basic steps.*

We argued in Section 4.2.2 that the number of arithmetic operations required to multiply a pair of $n \times n$ matrices is $f(n) = 2n^3 - n^2$. It's not hard to see that for $n \geq 1$,

$$n^3 \leq 2n^3 - n^2 \qquad \text{or equivalently} \qquad 0 \leq n^3 - n^2$$

and so $f(n) = \Omega(n^3)$. Further, it's also easy to see that for $n \geq 1$ we have

$$2n^3 - n^2 \leq 3n^3 \qquad \text{or equivalently} \qquad -n^3 - n^2 \leq 0,$$

and so we also have that $f(n) = O(n^3)$. Combining these bounds, we've established that

$$f(n) = \Theta(n^3),$$

which is a special case of a more general result. One can prove—see the Problem Sets—that if $f : \mathbb{N} \to \mathbb{R}^+$ is a polynomial in n of degree k, then $f = \Theta(n^k)$. Algorithms that are $O(n^k)$ for some $k \in \mathbb{R}$ are often called *polynomial time algorithms.*

4.4.3 Primality testing via trial division

Proposition 4.8 (Primality testing)**.** *If we characterise the size of the input with $d = \log_{10}(n)$ (essentially the number of decimal digits in n) and take as our basic step the computation of $n \bmod b$ for a number $2 \leq b \leq \sqrt{n}$, then primality testing via trial division requires $O(10^{d/2})$ steps.*

In the most demanding cases, when n is actually prime, we need to compute $n \bmod b$ for each integer b in the range $2 \leq b \leq \sqrt{n}$. This means we need to do $\lfloor \sqrt{n} \rfloor - 1$ basic steps and so the number of steps $f(d)$ required by the algorithm satisfies

$$\begin{aligned} f(d) &\leq \lfloor \sqrt{n} \rfloor - 1 \\ &\leq \lfloor \sqrt{n} \rfloor \\ &\leq \sqrt{n}. \end{aligned}$$

To get the righthand side in terms of the problem size $d = \lfloor \log_{10}(n) \rfloor + 1$, note that, for all $x \in \mathbb{R}$, $x < \lfloor x \rfloor + 1$ and so

$$\log_{10}(n) < d.$$

Then

$$\sqrt{n} \;=\; n^{1/2} \;=\; \left(10^{\log_{10}(n)}\right)^{1/2}$$

which implies that

$$\begin{aligned} f(d) &\leq \sqrt{n} \\ &\leq \left(10^{\log_{10}(n)}\right)^{1/2} \\ &\leq \left(10^{d}\right)^{1/2} \\ &\leq 10^{d/2}, \end{aligned}$$

and thus that primality testing via trial division is $O(10^{d/2})$.

Such algorithms are often called *exponential-time* or just *exponential* and they are generally regarded as impractical for, as one increases d, the computational requirements can jump abruptly from something modest and doable to something impossible. Further, we haven't yet taken any account of the way the sizes of the numbers n and b affect the amount of work required for the basic step. If we were to do so—if, say, we choose single-digit arithmetic operations as our basic steps—the bound on the operation count would only grow larger: trial division is *not* a feasible way to test large numbers for primality.

4.5 Afterword

I chose the algorithms discussed above for simplicity, but they are not necessarily the best known ways to solve the problems. I also simplified the analysis by judicious choice of problem-size measurement and basic step. For example, in practical matrix multiplication problems the multiplication of two matrix elements is more computationally expensive than addition of two products, to the extent that people normally just ignore the additions and try to estimate the number of multiplications. The standard algorithm is still $\Theta(n^3)$, but more efficient algorithms are known. The basic idea is related to a clever observation about the number of multiplications required to compute the product of two complex numbers

$$(a + ib) \times (c + id) = (ac - bd) + i(ad + bc). \tag{4.3}$$

The most straightforward approach requires us to compute the four products ac, bd, ad and bc. But Gauss noticed that one can instead compute just three products, ac, bd and

$$q = (a + b)(c + d) = ac + ad + bc + bd,$$

and then use the relation

$$(ad + bc) = q - ac - bd$$

to compute the imaginary part of the product in Eqn. (4.3). In 1969 Volker Strassen discovered a similar trick whose simplest application allows one to compute the product of two 2×2 matrices with only 7 multiplications, as opposed to the 8 that the standard algorithm requires. Building on this observation, he found an algorithm that can compute all the entries in the product of two $n \times n$ matrices using only $O(n^{\log_2(7)}) \approx O(n^{2.807})$ multiplications[3].

More spectacularly, it turns out that there is a polynomial-time algorithm for primality testing. It was discovered in the early years of this century by Agrawal, Kayal and Saxena (often shortened to AKS)[4]. This is particularly cheering in that two of the authors, Kayal and Saxena, were undergraduate project students when they did this work.

[3] I learned about Strassen's work in a previous edition of William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery (2007), *Numerical Recipes in C++*, 3rd edition, CUP, Cambridge. ISBN: 978-0-521-88068-8, which is very readable, but for a quick overview of the area you might want to look at Sara Robinson (2005), `Toward an optimal algorithm for matrix multiplication`, *SIAM News*, **38**(9).

[4] See: Manindra Agrawal, Neeraj Kayal and Nitin Saxena (2004), PRIMES is in P, *Annals of Mathematics*, **160**(2):781–793. DOI: `10.4007/annals.2004.160.781`. The original AKS paper is quite approachable, but an even more reader-friendly treatment of their proof appears in Andrew Granville (2005), It is easy to determine whether a given integer is prime, *Bulletin of the AMS*, **42**:3–38. DOI: `10.1090/S0273-0979-04-01037-7`.

Lecture 5

Walks, Trails, Paths and Connectedness

Reading: Some of the material in this lecture comes from Section 1.2 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, which is available online via SpringerLink.

If you are at the university, either physically or via the VPN, you can download the chapters of this book as PDFs.

Several of the examples in the previous lectures—for example two of the subgraphs in Figure 2.7 and the graph in Figure 1.12—consist of two or more "pieces". If one thinks about the definition of a graph as a pair of sets, these multiple pieces don't present any mathematical problem, but it proves useful to have precise vocabulary to discuss them.

5.1 Walks, trails and paths

The first definition we need involves a sequence of edges

$$(e_1, e_2, \ldots, e_L) \tag{5.1}$$

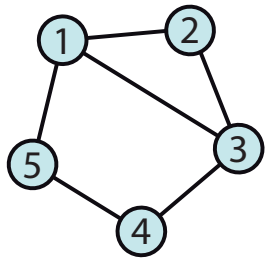
Note that some edges may appear more than once.

Definition 5.1. *A sequence of edges such as the one in Eqn (5.1) is a* ***walk*** *in a graph $G(V, E)$ if there exists a corresponding sequence of vertices*

$$(v_0, v_1, \ldots, v_L). \tag{5.2}$$

such that $e_j = (v_{j-1}, v_j) \in E$. Note that the vertices don't have to be distinct. A walk for which $v_0 = v_L$ is a ***closed walk****.*

This definition makes sense in both directed and undirected graphs and in the latter case corresponds to a path that goes along the edges in the sense of the arrows that represent them.



The walk specified by the edge sequence

$$(e_1, e_2, e_3, e_4) = ((1,2),\ (2,3),\ (3,1),\ (1,5))$$

has corresponding vertex sequence

$$(v_0, v_1, v_2, v_3, v_4) = (1, 2, 3, 1, 5),$$

while the vertex sequence
$(v_0, v_1, v_2, v_3) = (1, 2, 3, 1)$ corresponds to a closed walk.

Figure 5.1: *Two examples of walks.*

Definition 5.2. *The **length** of a walk is the number of edges in the sequence. For the walk in Eqn. 5.1 the length is thus L.*

It'll prove useful to define two more constrained sorts of walk:

Definition 5.3. *A **trail** is a walk in which all the edges e_j are distinct and a **closed trail** is a closed walk that is also a trail.*

Definition 5.4. *A **path** is a trail in which all the vertices in the sequence in Eqn (5.2) are distinct.*

Definition 5.5. *A **cycle** is a closed trail in which all the vertices are distinct, except for the first and last, which are identical.*

Remark 5.6. *In an undirected graph a cycle is a subgraph isomorphic to one of the cycle graphs C_n and must include at least three edges, but in directed graphs and multigraphs it is possible to have a cycle with just two edges.*

Remark 5.7. *As the three terms walk, trail and path mean very similar things in ordinary speech, it can be hard to keep their graph-theoretic definitions straight, even though they make useful distinctions. The following observations may help:*

- *All trails are walks and all paths are trails. In set-theoretic notation:*

$$\textit{Walks} \supseteq \textit{Trails} \supseteq \textit{Paths}$$

- *There are trails that aren't paths: see Figure 5.2.*

5.2 Connectedness

We want to be able to say that two vertices are connected if we can get from one to the other by moving along the edges of the graph. Here's a definition that builds on the terms defined in the previous section:

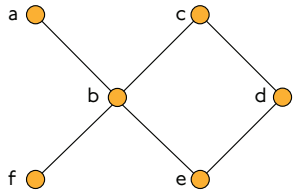


Figure 5.2: *The walk specified by the vertex sequence (a, b, c, d, e, b, f) is a trail as all the edges are distinct, but it's not a path as the vertex b is visited twice.*

Definition 5.8. *In a graph $G(V, E)$, two vertices a and b are said to be* ***connected*** *if there is a walk given by a vertex sequence $(v_0, \ldots, v_L)$ where $v_0 = a$ and $v_L = b$. Additionally, we will say that a vertex is connected to itself.*

Definition 5.9. *A graph in which each pair of vertices is connected is a* ***connected graph****.*

See Figure 5.3 for an example of a connected graph and another that is not connected.

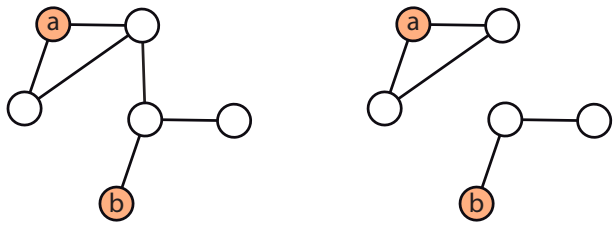


Figure 5.3: *The graph at left is connected, but the one at right is not, because there is no walk connecting the shaded vertices labelled a and b.*

Once we have the definitions above, it's possible to make a precise definition of the "pieces" of a graph. It depends on the notion of an *equivalence relation*, which you should have met earlier your studies.

Definition 5.10. *A relation $\sim$ on a set S is an* ***equivalence relation*** *if it is:*

reflexive: $a \sim a$ *for all* $a \in S$*;*

symmetric: $a \sim b \;\Rightarrow\; b \sim a$ *for all* $a, b \in S$*;*

transitive: $a \sim b$ *and* $b \sim c \;\Rightarrow\; a \sim c$ *for all* $a, b, c \in S$*.*

The main use of an equivalence relation on S is that it decomposes S into a collection of disjoint *equivalence classes*. That is, we can write

$$S = \bigcup_j S_j$$

where $S_j \cap S_k = \emptyset$ if $j \neq k$ and $a \sim b$ if and only if $a, b \in S_j$ for some j.

5.2.1 Connectedness in undirected graphs

The key idea is that "is-connected-to" is an equivalence relation on the vertex set of a graph. To see this, we need only check the three properties:

reflexive: This is true by definition, and is the main reason why we say that a vertex is always connected to itself.

symmetric: If there is a walk from a to b then we can simply reverse the corresponding sequence of edges to get a walk from b to a.

transitive: Suppose a is connected to b, so that the graph contains a walk corresponding to some vertex sequence

$$(a = u_0, u_1, \ldots, u_{L_1-1}, u_{L_1} = b)$$

that connects a to b. If there is also a walk from b to c given by some vertex sequence

$$(b = v_0, v_1, \ldots, v_{L_2-1}, v_{L_2} = c)$$

then we can get a walk from a to c by tracing over the two walks listed above, one after the other. That is, there is a walk from a to c given by the vertex sequence

$$(a, u_1, \ldots, u_{L_1-1}, b, v_1, \ldots, v_{L_2-1}, c)\,.$$

We have shown that if a is connected to b and b is connected to c, then a is connected to c and this is precisely what it means for "is-connected-to" to be a transitive relation.

The process of traversing one walk after another, as we did in the proof of the transitive property, is sometimes called *concatenation* of walks.

Definition 5.11. *In an undirected graph $G(V, E)$ a* ***connected component*** *is a subgraph induced by an equivalence class under the relation "is-connected-to" on V.*

The disjointness of equivalence classes means that each vertex belongs to exactly one connected component and so we will sometimes talk about the *connected component of a vertex.*

5.2.2 Connectedness in directed graphs

In directed graphs "is-connected-to" isn't an equivalence relation because it's not symmetric. That is, even if we know that there's a walk from some vertex a to another vertex b, we have no guarantee that there's a walk from b to a: Figure 5.4 provides an example. None the less, there is an analogue of a connected component in a directed graph that's captured by the following definitions:

Definition 5.12. *In a directed graph $G(V, E)$ a vertex b is said to be* ***accessible*** *or* ***reachable*** *from another vertex a if G contains a walk from a to b. Additionally, we'll say that all vertices are accessible (or reachable) from themselves.*

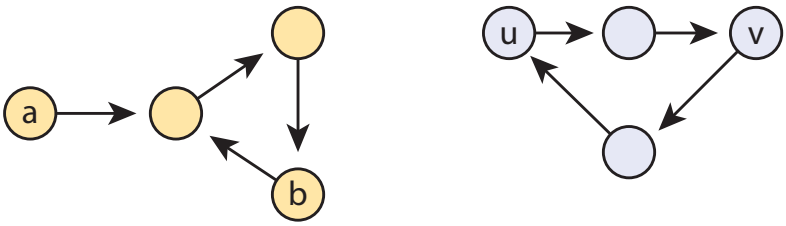


Figure 5.4: *In a directed graph it's possible to have a walk from vertex a to vertex b without having a walk from b to a, as in the digraph at left. In the digraph at right there are walks from u to v and from v to u so this pair is strongly connected.*

Definition 5.13. *Two vertices a and b in a directed graph are* ***strongly connected*** *if b is accessible from a* and *a is accessible from b. Additionally, we regard a vertex as strongly connected to itself.*

With these definitions it's easy to show (see the Problem Sets) that "is-strongly-connected-to" is an equivalence relation on the vertex set of a directed graph and so the vertex set decomposes into a disjoint union of *strongly connected components*. This prompts the following definition:

Definition 5.14. *A directed graph $G(V, E)$ is* ***strongly connected*** *if every pair of its vertices is strongly connected. Equivalently, a digraph is strongly connected if it contains exactly one strongly connected component.*

Finally, there's one other notion of connectedness applicable to directed graphs, *weak connectedness*:

Definition 5.15. *A directed graph $G(V, E)$ is* ***weakly connected*** *if, when one converts all its edges to undirected ones, it becomes a connected, undirected graph.*

Figure 5.5 illustrates the difference between strongly and weakly connected graphs. Finally, I'd like to introduce a piece of notation for the graph that one gets by ignoring the directedness of the edges in a digraph:

Definition 5.16. *If $G(V, E)$ is a directed multigraph then $|G|$ is the undirected multigraph produced by ignoring the directedness of the edges. Note that if both the directed edges (a, b) and (b, a) are present in a digraph $G(V, E)$, then* two *parallel copies of the undirected edge (a, b) appear in $|G|$.*

5.3 Afterword: a useful proposition

As with the Handshaking Lemma in Lecture 1, I'd like to finish off a long run of definitions by using them to formulate and prove a small, useful result.

Proposition 5.17 (Connected vertices are joined by a path)**.** *If two vertices a and b are connected, so that there is a walk from a to b, then there is also a path from a to b.*

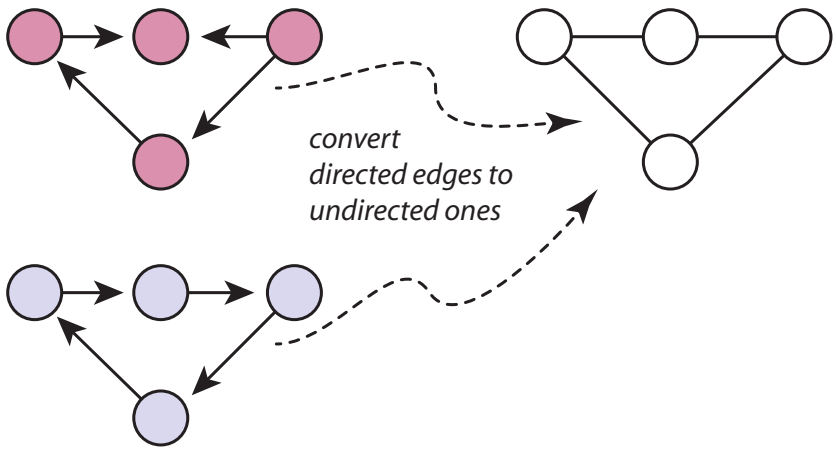


Figure 5.5: *The graph at the top is weakly connected, but not strongly connected, while the one at the bottom is both weakly and strongly connected.*

Proof of Proposition 5.17

To say that two vertices a and b in a graph $G(V, E)$ are connected means that there is a walk given by a vertex sequence

$$(v_0, \ldots, v_L) \tag{5.3}$$

where $v_0 = a$ and $v_L = b$. There are two possibilities:

(i) all the vertices in the sequence are distinct;

(ii) some vertex or vertices appear more than once.

In the first case the walk is also a path and we are finished. In the second case it is always possible to find a path from a to b by removing some edges from the walk in Eqn. (5.3). This sort of "path surgery" is outlined below and illustrated in Example 5.18.

We are free to assume that the set of repeated vertices doesn't include a or b as we can easily make this true by trimming some vertices off the two ends of the sequence. To be concrete, we can define a new walk by first trimming off everything before the last appearance of a—say that's v_j—to yield a walk specified by the vertex sequence

$$(v_j, \ldots, v_L)$$

and then, in that walk, remove everything that comes after the first appearance of b—say that's v_k—so that we end up with a new walk

$$(a = v'_0, \ldots, v'_{L'} = b) = (v_j, \ldots, v_k). \tag{5.4}$$

To finish the proof we then need to deal with the case where the walk in (5.4), which doesn't contain any repeats of a or b, still contains repeats of one or more

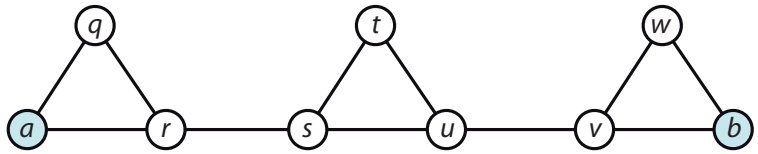


Figure 5.6: *In the graph above the shaded vertices a and b are connected by the path (a, r, s, u, v, b).*

other vertices. Suppose that $c \in V$ with $c \neq a, b$ is such a repeated vertex: we can eliminate repeated visits to c by defining a new walk specified by the vertex sequence

$$(a = u_0, \ldots, u_{L''} = b) = (v'_0, \ldots, v'_j, v'_{k+1}, \ldots, v'_{L'}) \tag{5.5}$$

where v'_j is the first appearance of c in the sequence at left in Eqn. (5.4) and v'_k is the last. There can only be finitely many repeated vertices in the original walk (5.3) and so, by using the approach sketched above repeatedly, we can eliminate them all, leaving a path from a to b. Very scrupulous students may wish to rewrite this proof using induction on the number of repeated vertices.

Example 5.18 (Connected vertices are connected by a path)**.** *Consider the graph is Figure 5.6. The vertices a and b are connected by the walk*

$$(v_0, \ldots, v_{15}) = (\underline{a,\ q,\ r,}\ a,\ r,\ s,\ t,\ u,\ s,\ t,\ u,\ v,\ b\underline{,\ w,\ v,\ b})$$

which contains many repeated vertices. To trim it down to a path we start by eliminating repeats of a and b using the approach from Eqn. (5.4), which amounts to trimming off those vertices that are underlined in the vertex sequence above.

To see how this works, notice that the vertex a appears as v_0 and v_3 in the original walk and we want $v'_0 = v_j$ in (5.4) to be its last appearance, so we set $v_j = v_3$. Similarly, b appears as v_{12} and v_{15} in the original walk and we want $v'_{L'} = v_k$ to be b's first appearance, so we set $v_k = v_{12}$. This leaves us with

$$\begin{aligned}(v'_0, \ldots, v'_9) &= (v_3, \ldots, v_{12}) \\ &= (a,\ r,\ s,\ \underline{t,\ u,\ s,}\ t,\ u,\ v,\ b)\end{aligned} \tag{5.6}$$

Finally, we eliminate the remaining repeated vertices by applying the approach from Eqn. (5.5) to the sequence $(v'_0, \ldots, v'_9)$. This amounts to chopping out the sequence of vertices underlined in Eqn. (5.6). To follow the details, note that each of the vertices s, t and u appears twice in (5.6). To eliminate, say, the repeated appearances of s we should use (5.4) with $u_j = v'_2$ as the first appearance of s in Eqn. (5.6) and $u_k = v'_5$ as the last. This leaves us with the new walk

$$(u_0, \ldots, u_6) \ = \ (v'_0, v'_1, v'_2, v'_6, v'_7, v'_8, v'_9) \ = \ (a, r, s, t, u, v, b)$$

which is a path connecting a to b.

Part II

Trees and the Matrix-Tree Theorem

Lecture 6

Trees and forests

This section of the notes introduces an important family of graphs—trees and forests—and also serves as an introduction to inductive proofs on graphs.

Reading:
The material in today's lecture comes from Section 1.2 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition (available online via SpringerLink),

though the discussion there includes a lot of material about counting trees that we'll handle in a different way.

6.1 Basic definitions

We begin with a flurry of definitions.

Definition 6.1. *A graph $G(V, E)$ is* ***acyclic*** *if it doesn't include any cycles.*

Another way to say a graph is acyclic is to say that it contains no subgraphs isomorphic to one of the cycle graphs.

Definition 6.2. *A* ***tree*** *is a connected, acyclic graph.*

Definition 6.3. *A* ***forest*** *is a graph whose connected components are trees.*

Trees play an important role in many applications: see Figure 6.1 for examples.

6.1.1 Leaves and internal nodes

Trees have two sorts of vertices: *leaves* (sometimes also called *leaf nodes*) and *internal nodes*: these terms are defined more carefully below and are illustrated in Figure 6.2.

Definition 6.4. *A vertex $v \in V$ in a tree $T(V, E)$ is called a* ***leaf*** *or* ***leaf node*** *if* $\deg(v) = 1$ *and it is called an* ***internal node*** *if* $\deg(v) > 1$.

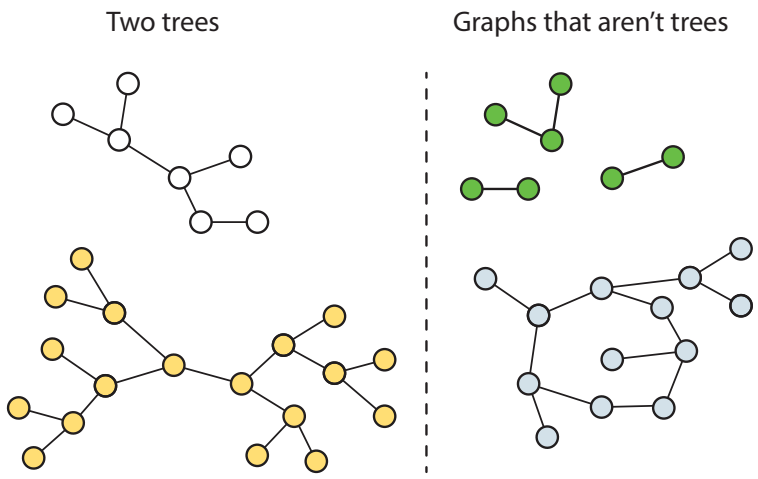


Figure 6.1: *The two graphs at left (white and yellow vertices) are trees, but the two at right aren't: the one at upper right (with green vertices) has multiple connected components (and so it isn't connected) while the one at lower right (blue vertices) contains a cycle. The graph at upper right is, however, a forest as each of its connected components is a tree.*

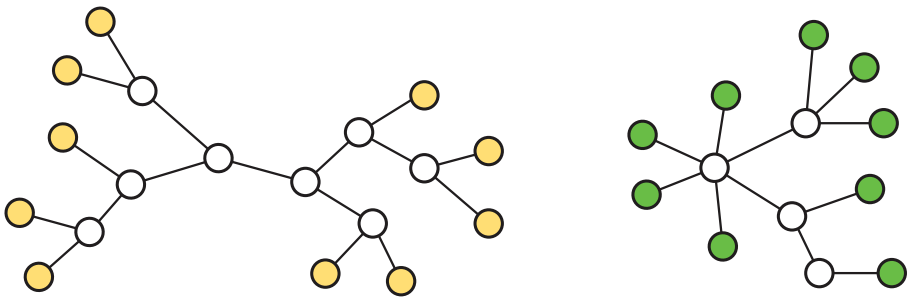


Figure 6.2: *In the two trees above the internal nodes are white, while the leaf nodes are coloured green or yellow.*

6.1.2 Kinds of trees

Definition 6.5. *A* ***binary tree*** *is a tree in which every internal node has degree three.*

Definition 6.6. *A* ***rooted tree*** *is a tree with a distinguished leaf node called the* root node*.*

Warning to the reader: The definition of rooted tree above is common among biologists, who use trees to represent evolutionary lineages (see Darwin's sketch at right in Figure 6.3). Other researchers, especially computer scientists, use the same term to mean something slightly different.

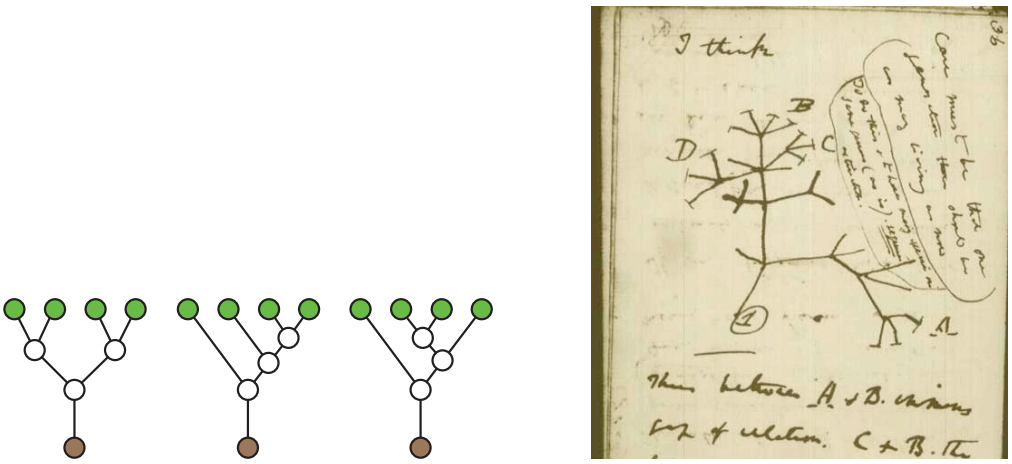


Figure 6.3: *At left are three examples of rooted binary trees. In all cases the root node is brown, the leaves are green and the internal nodes are white. At right is a page from one of Darwin's notebooks, showing the first known sketch of an evolutionary tree: here the nodes represent species and the edges indicate evolutionary descent.*

6.2 Three useful lemmas and a proposition

Lemma 6.7 (Minimal $|E|$ in a connected graph)**.** *A connected graph on n vertices has at least $(n-1)$ edges.*

Lemma 6.8 (Maximal $|E|$ in an acyclic graph)**.** *An acyclic graph on n vertices has at most $(n-1)$ edges.*

Definition 6.9. *A vertex v is said to be* ***isolated*** *if it has no neighbours. Equivalently, v is isolated if* $\deg(v) = 0$*.*

Lemma 6.10 (Vertices of degree 1)**.** *If a graph $G(V, E)$ has $n \geq 2$ vertices, none of which are isolated, and $(n-1)$ edges then G has at least two vertices of degree 1.*

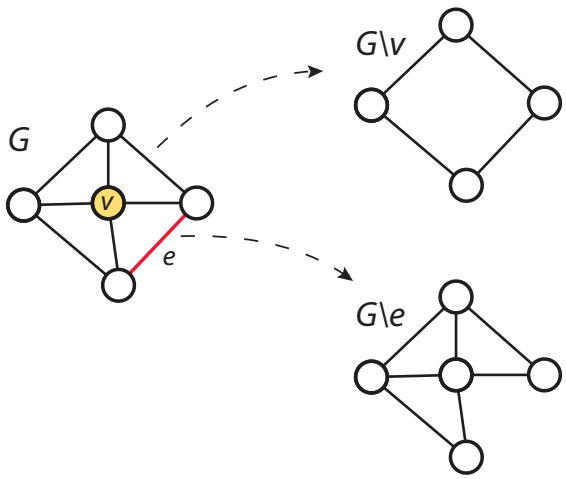


Figure 6.4: *A graph $G(V, E)$ and the subgraphs $G\backslash v$ formed by deleting the yellow vertex v and $G\backslash e$ formed by deleting the red edge e.*

6.2.1 A festival of proofs by induction

Proofs by induction about graphs generally have three parts

- a **base case** that typically involves a graph with very few vertices or edges (often just one or two) and for which the result is obvious;
- an **inductive hypothesis** in which one assumes the result is true for all graphs with, say, n_0 or fewer vertices (or perhaps m_0 or fewer edges);
- an **inductive step** where one starts with a graph that satisfies the hypotheses of the theorem and has, say, $n_0 + 1$ vertices (or $m_0 + 1$ edges or whatever is appropriate) and then *reduces* the theorem as it applies to this larger graph to something involving smaller graphs (to which the inductive hypothesis applies), typically by *deleting* an edge or vertex.

6.2.2 Graph surgery

The proofs below accomplish their inductive steps by deleting either an edge or a vertex, so here I introduce some notation for these processes.

Definition 6.11. *If $G(V, E)$ is a graph and $v \in V$ is one of its vertices then $G\backslash v$ is defined to be the subgraph formed by deleting v and all the edges that are incident on v.*

Definition 6.12. *If $G(V, E)$ is a graph and $e \in E$ is one of its edges then $G\backslash e$ is defined to be the subgraph formed by deleting e.*

Both these definitions are illustrated in Figure 6.4.

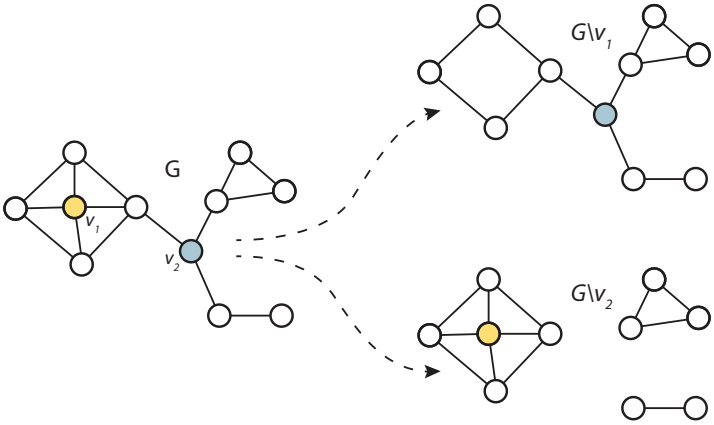


Figure 6.5: *In the inductive step of the proof of Lemma 6.7 we delete some arbitrary vertex $v \in V$ in a connected graph $G(V, E)$ to form the graph $G\backslash v$. The result may still be a connected graph, as in $G\backslash v_1$ at upper right, or may fall into several connected components, as in $G\backslash v_2$ at lower right.*

Proof of Lemma 6.7

We'll prove Lemma 6.7 by induction on the number of vertices. First let us rephrase the lemma in an equivalent way:

> If $G(V, E)$ is a connected graph on $|V| = n$ vertices, then $|E| \geq n - 1$.

Base case: There is only one graph with $|V| = 1$ and it is, by definition, connected and has $|E| = 0$, which satisfies the lemma. One could alternatively start from K_2, which is the only connected graph on two vertices and has $|E| = 1$.

Inductive hypothesis: Suppose that the lemma is true for all graphs $G(V, E)$ with $1 \leq |V| \leq n_0$, for some fixed n_0.

Inductive step: Now consider a connected graph $G(V, E)$ with $|V| = n_0 + 1$: the lemma we're trying to prove then says $|E| \geq n_0$. Choose some vertex $v \in V$ and delete it, forming the graph $G\backslash v$. We'll say that the new graph has vertex set $V' = V\backslash v$ and edge set E'. There are two possibilities (see Figure 6.5):

(i) $G\backslash v$ is still a connected graph;

(ii) $G\backslash v$ has $k \geq 2$ connected components: call these $G_1(V_1, E_1), \ldots, G_k(V_k, E_k)$.

In the first case—where $G\backslash v$ is connected—we also know $|V'| = |V| - 1 = n_0$ and so the inductive hypothesis applies and tells us that $|E'| \geq (n_0 - 1)$. But as G was connected, the vertex v that we deleted must have had at least one neighbour, and hence at least one edge, so we have

$$|E| \;\geq\; |E'| + 1 \;\geq\; (n_0 - 1) + 1 \;\geq\; n_0$$

which is exactly the result we sought.

In the second case—where deleting v causes G to fall into $k \geq 2$ connected components—we can call the components $G_1(V_1, E_1), G_2(V_2, E_2), \cdots, G_k(V_k, E_k)$ with $n_j = |V_j|$. Then

$$\sum_{j=1}^{k} n_j \;=\; \sum_{j=1}^{k} |V_j| \;=\; |V'| \;=\; |V| - 1 \;=\; n_0.$$

Further, the j-th connected component is a connected graph on $n_j < n_0$ vertices and so the inductive hypothesis applies to each component separately, telling us that $|E_j| \geq n_j - 1$. But then we have

$$|E'| \;=\; \sum_{j=1}^{k} |E_j| \;\geq\; \sum_{j=1}^{k} (n_j - 1) \;\geq\; \left(\sum_{j=1}^{k} n_j \right) - k \;\geq\; n_0 - k. \tag{6.1}$$

And, as we know that the original graph G was connected, we also know that the deleted vertex v was connected by at least one edge to each of the k components of $G \backslash v$. Combining this observation with Eqn. (6.1) gives us

$$|E| \;\geq\; |E'| + k \;\geq\; (n_0 - k) + k \;\geq\; n_0,$$

which proves the lemma for the second case too.

Proof of Lemma 6.8

Once again, we'll do induction on the number of vertices. As above, we begin by rephrasing the lemma:

If $G(V, E)$ is an acyclic graph on $|V| = n$ vertices, then $|E| \leq n - 1$.

Base case: Either K_1 or K_2 could serve as the base case: both are acyclic graphs that have a maximum of $|V| - 1$ edges.

Inductive hypothesis: Suppose that Lemma 6.8 is true for all acyclic graphs with $|V| \leq n_0$, for some fixed n_0.

Inductive step: Consider an acyclic graph $G(V, E)$ with $|V| = n_0 + 1$: we want to prove that $|E| \leq n_0$. Choose an arbitrary edge $e = (a, b) \in E$ and delete it to form $G'(V, E') = G \backslash e$, which has the same vertex set as G, but a smaller edge set $E' = E \backslash e$.

First note that G' must have one more connected component than G does because a and b, the two vertices that appear in the deleted edge e, are connected in G, but cannot be connected in G'. If they were still connected, there would (by Prop. 5.17) be a path connecting them in G' that, when combined with e, would form a cycle in G, contradicting the assumption that G is acyclic. Thus we know that G' has $k \geq 2$ connected components that we can call $G_1(V_1, E_1), \ldots, G_k(V_k, E_k)$.

If we again define $n_j = |V_j|$, we know that $n_j \leq n_0$ for all j and so the inductive hypothesis applies to each component separately: $|E_j| \leq n_j - 1$. Adding these up yields

$$|E'| \;=\; \sum_{j=1}^{k} |E_j| \;\leq\; \sum_{j=1}^{k} (n_j - 1) \;\leq\; \left(\sum_{j=1}^{k} n_j \right) - k \;\leq\; (n_0 + 1) - k.$$

And then, as $|E| = |E'| + 1$ we have

$$|E| \;=\; |E'| + 1 \;\leq\; (n_0 + 1 - k) + 1 \;\leq\; n_0 + (2 - k) \;\leq\; n_0,$$

where the final inequality follows from the observation that G' has $k \geq 2$ connected components.

Proof of Lemma 6.10

The final Lemma in this section is somewhat technical: we'll use it in the proof of a theorem in Section 6.3. The lemma says that graphs $G(V, E)$ that have $|V| = n$ and $|E| = (n - 1)$ and have no isolated vertices must contain at least two vertices with degree one. The proof is by contradiction and uses the Handshaking Lemma.

Imagine the vertices are numbered and arranged in order of increasing degree so $V = \{v_1, \ldots, v_n\}$ and $\deg(v_1) \leq \deg(v_2) \leq \cdots \leq \deg(v_n)$. The Handshaking Lemma then tells us that

$$\sum_{j=1}^{n} \deg(v_j) \;=\; 2|E| \;=\; 2(n - 1) \;=\; 2n - 2. \tag{6.2}$$

As there are no isolated vertices, we also know that $\deg(v_j) \geq 1$ for all j. Now assume—aiming for a contradiction—that there is at most a single vertex with degree one. That is, assume $\deg(v_1) \geq 1$, but $\deg(v_j) \geq 2 \;\; \forall j \geq 2$. Then

$$\begin{aligned}
\sum_{j=1}^{n} \deg(v_j) &= \deg(v_1) + \sum_{j=2}^{n} \deg(v_j) \\
&\geq 1 + \sum_{j=2}^{n} 2 \\
&\geq 1 + (n - 1) \times 2 \\
&\geq 2n - 1.
\end{aligned}$$

This contradicts Eqn. (6.2), which says that the sum of degrees is $2n - 2$. Thus it must be true that two or more vertices have degree one, which is the result we sought.

6.3 A theorem about trees

The lemmas of the previous section make it possible to give several nice characterisations of a tree and the theorem below, which has a form that one often finds in Discrete Maths or Algebra books, shows that they're all equivalent.

Theorem 6.13 (Jungnickel's Theorem 1.2.8)**.** *For a graph $G(V, E)$ on $|V| = n$ vertices, any two of the following imply the third:*

(a) G is connected.

(b) G is acyclic.

(c) G has $(n-1)$ edges.

6.3.1 Proof of the theorem

The theorem above is really three separate propositions bundled into one statement: we'll prove them in turn.

(a) and (b) $\implies$ (c)

On the one hand, our lemma about the minimal number of edges in a connected graph (Lemma 6.7) says that property (a) implies that $|E| \geq (n-1)$. On the other hand our lemma about the maximal number of edges in an acyclic graph (Lemma 6.8) says $|E| \leq (n-1)$. The only possibility compatible with both these inequalities is $|E| = (n-1)$.

(a) and (c) $\implies$ (b)

To prove this by contradiction, assume that it's possible to have a connected graph $G(V, E)$ that has $(n-1)$ edges and contains a cycle. Choose some edge e that's part of the cycle and delete it to form $H = G \backslash e$. H is then a connected graph (removing an edge from a cycle does not change the number of connected components) with only $n-2$ edges, which contradicts our earlier result (Lemma 6.7) about the minimal number of edges in a connected graph.

(b) and (c) $\implies$ (a)

We'll prove—by induction on $n = |V|$—that an acyclic graph with $|V| - 1$ edges must be connected.

Base case: There is only one graph with $|V| = 1$. It's acyclic, has $|V| - 1 = 0$ edges and is connected.

Inductive hypothesis: Suppose that all acyclic graphs with $1 \leq |V| \leq n_0$ vertices and $|E| = |V| - 1$ edges are connected.

Inductive step: Now consider an acyclic graph $G(V, E)$ with $|V| = n_0 + 1$ and $|E| = n_0$: we need to prove that it's connected. First, notice that such a graph cannot have any isolated vertices, for suppose there was some vertex v with $\deg(v) = 0$. We could then delete v to produce $H = G \backslash v$, which would be an acyclic graph with n_0 vertices and n_0 edges, contradicting our lemma (Lemma 6.8) about the maximal number of edges in an acyclic graph.

Thus G contains no isolated vertices and so, by the technical lemma from the previous section (Lemma 6.10), we know that it has at least two vertices of degree one. Say that one of these two is $u \in V$ and delete it to make $G'(V', E') = G \backslash u$. Then G' is still acyclic, because G is, and deleting vertices can't create cycles. Furthermore G' has $|V'| = |V| - 1 = n_0$ vertices and $|E'| = |E| - 1 = n_0 - 1$ edges. This means that the inductive hypothesis applies and we can conclude that G' is connected. But if G' is connected, so is G and we are finished.

Lecture 7

The Matrix-Tree Theorems

This section of the notes introduces a pair of very beautiful theorems that use linear algebra to count trees in graphs.

Reading:
The next few lectures are not covered in Jungnickel's book, though a few definitions in our Section 7.2.1 come from his Section 1.6. But the main argument draws on ideas that you should have met in Foundations of Pure Mathematics, Linear Algebra and Algebraic Structures.

7.1 Kirchoff's Matrix-Tree Theorem

Our goal over the next few lectures is to establish a lovely connection between Graph Theory and Linear Algebra. It is part of a circle of beautiful results discovered by the great German physicist Gustav Kirchoff in the mid-19th century, when he was studying electrical circuits. To formulate his result we need a few new definitions.

Definition 7.1. *A subgraph $T(V, E')$ of a graph $G(V, E)$ is a* ***spanning tree*** *if it is a tree that contains every vertex in V.*

Figure 7.1 gives some examples.

Definition 7.2. *If $G(V, E)$ is a graph on n vertices with $V = \{v_1, \ldots, v_n\}$ then its* ***graph Laplacian*** *L is an $n \times n$ matrix whose entries are*

$$L_{ij} = \begin{cases} \deg(v_j) & \text{If } i = j \\ -1 & \text{If } i \neq j \text{ and } (v_i, v_j) \in E \\ 0 & \text{Otherwise} \end{cases}$$

Equivalently, $L = D - A$, where D is a diagonal matrix with $D_{jj} = \deg(v_j)$ and A is the graph's adjacency matrix.

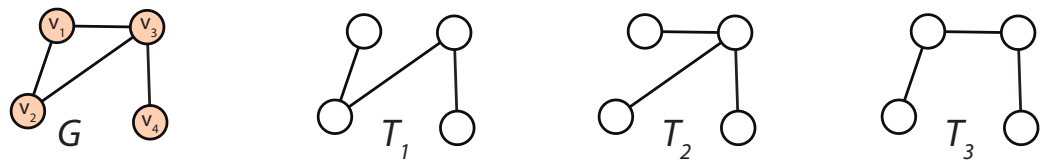


Figure 7.1: *A graph $G(V, E)$ with $V = \{v_1, \ldots, v_4\}$ and three of its spanning trees: T_1, T_2 and T_3. Note that although T_1 and T_3 are isomorphic, we regard them as* ***different*** *spanning trees for the purposes of the Matrix-Tree Theorem.*

Example 7.3 (Graph Laplacian)**.** *The graph G whose spanning trees are illustrated in Figure 7.1 has graph Laplacian*

$$\begin{aligned} L &= D - A \\ &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \end{aligned} \tag{7.1}$$

Once we have these two definitions it's easy to state the Matrix-Tree theorem

Theorem 7.4 (Kirchoff's Matrix-Tree Theorem, 1847)**.** *If $G(V, E)$ is an undirected graph and L is its graph Laplacian, then the number N_T of spanning trees contained in G is given by the following computation.*

(1) Choose a vertex v_j and eliminate the j-th row and column from L to get a new matrix $\hat{L}_j$;

(2) Compute

$$N_T = \det(\hat{L}_j). \tag{7.2}$$

The number N_T in Eqn. (7.2) counts spanning trees that are distinct **as subgraphs of** G: equivalently, we regard the vertices as distinguishable. Thus some of the trees that contribute to N_T may be isomorphic: see Figure 7.1 for an example.

This result is remarkable in many ways—it seems amazing that the answer doesn't depend on which vertex we choose when constructing $\hat{L}_j$—but to begin with let's simply use the theorem to compute the number of spanning trees for the graph in Example 7.3

Example 7.5 (Counting spanning trees)**.** *If we take G to be the graph whose Laplacian is given in Eqn. (7.1) and choose $v_j = v_1$ we get*

$$\hat{L}_1 = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

and so the number of spanning trees is

$$\begin{aligned} N_T &= \det(\hat{L}_1) \\ &= 2 \times \det \begin{bmatrix} 3 & -1 \\ -1 & 1 \end{bmatrix} - (-1) \times \det \begin{bmatrix} -1 & -1 \\ 0 & 1 \end{bmatrix} \\ &= 2 \times (3-1) + (-1-0) \\ &= 4 - 1 \ = \ 3 \end{aligned}$$

I'll leave it as an exercise for the reader to check that one gets the same result from $\det(\hat{L}_2)$, $\det(\hat{L}_3)$ *and* $\det(\hat{L}_4)$.

7.2 Tutte's Matrix-Tree Theorem

We'll prove Kirchoff's theorem as a consequence of a much more recent result[1] about directed graphs. To formulate this we need a few more definitions that generalise the notion of a tree to digraphs.

7.2.1 Arborescences: directed trees

Recall the definition of *accessible* from Lecture 5:

> In a directed graph $G(V, E)$ a vertex b is said to be **accessible** from another vertex a if G contains a walk from a to b. Additionally, we'll say that all vertices are accessible from themselves.

This allows us to define the following suggestive term:

Definition 7.6. *A vertex $v \in V$ in a directed graph $G(V, E)$ is a* ***root*** *if every other vertex is accessible from v.*

We'll then be interested in the following directed analogue of a tree:

Definition 7.7. *A directed graph $T(V, E)$ is a* ***directed tree*** *or* ***arborescence*** *if*

(i) T contains a root

(ii) The graph $|T|$ that one obtains by ignoring the directedness of the edges is a tree.

See Figure 7.2 for an example. Of course, it's then natural to define an analogue of a spanning tree:

Definition 7.8. *A subgraph $T(V, E')$ of a digraph $G(V, E)$ is a* ***spanning arborescence*** *if T is an arborescence that contains all the vertices of G.*

[1] Proved by Bill Tutte about a century after Kirchoff's result in W.T. Tutte (1948), The dissection of equilateral triangles into equilateral triangles, *Math. Proc. Cambridge Phil. Soc.*, **44**(4):463–482.

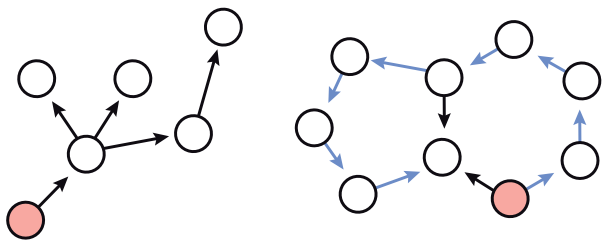


Figure 7.2: *The graph at left is an arborescence whose root vertex is shaded red, while the graph at right contains a spanning arborescence whose root is shaded red and whose edges are blue.*

7.2.2 Tutte's theorem

Theorem 7.9 (Tutte's Directed Matrix-Tree Theorem, 1948)**.** *If $G(V, E)$ is a digraph with vertex set $V = \{v_1, \ldots, v_n\}$ and L is an $n \times n$ matrix whose entries are given by*

$$L_{ij} = \begin{cases} \deg_{in}(v_j) & \text{If } i = j \\ -1 & \text{If } i \neq j \text{ and } (v_i, v_j) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{7.3}$$

then the number N_j of spanning arborescences with root at v_j is

$$N_j = \det(\hat{L}_j) \tag{7.4}$$

where $\hat{L}_j$ is the matrix produced by deleting the j-th row and column from L.

Here again, the number N_j in Eqn. (7.4) counts spanning arborescences that are distinct **as subgraphs of** G: equivalently, we regard the vertices as distinguishable. Thus some of the arborescences that contribute to N_j may be isomorphic, but if they involve different edges we'll count them separately.

Example 7.10 (Counting spanning arborescences)**.** *First we need to build the matrix L defined by Eqn. (7.3) in the statement of Tutte's theorem. If we choose G to be the graph pictured at upper left in Figure 7.3 then this is $L = D_{in} - A$ where D_{in} is a diagonal matrix with $D_{jj} = \deg_{in}(v_j)$ and A is the graph's adjacency matrix.*

$$\begin{aligned} L &= D_{in} - A \\ &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 1 & -1 \\ -1 & -1 & 0 & 2 \end{bmatrix} \end{aligned}$$

Then Table 7.1 summarises the results for the number of rooted trees.

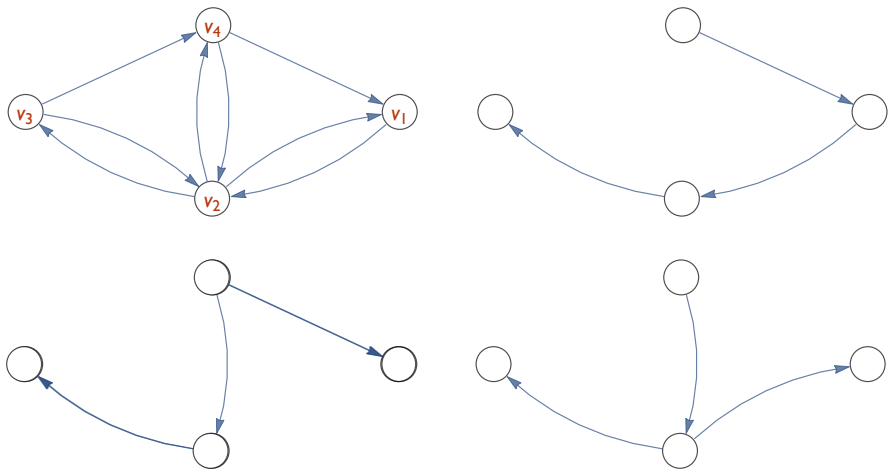


Figure 7.3: *The digraph at upper left, on which the vertices are labelled, has three spanning arborescences rooted at v_4.*

j	$\hat{L}_j$	$\det(\hat{L}_j)$
1	$\begin{bmatrix} 3 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & 0 & 2 \end{bmatrix}$	2
2	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 2 \end{bmatrix}$	4
3	$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{bmatrix}$	7
4	$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 1 \end{bmatrix}$	3

Table 7.1: *The number of spanning arborescences for the four possible roots in the graph at upper left in Figure 7.3.*

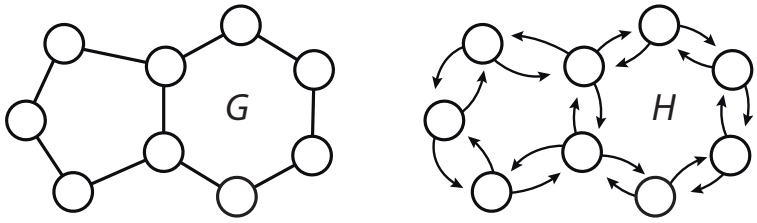


Figure 7.4: *If we convert an undirected graph such as G at left to a directed graph such as H at right, it is easy to count the spanning trees in G by counting spanning arborescences in H.*

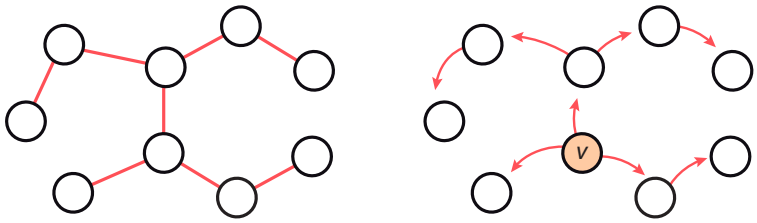


Figure 7.5: *The undirected graph at left is a spanning tree for G in Figure 7.4, while the directed graph at right is a spanning arborescence for H (right side of Fig. 7.4) rooted at the shaded vertex v.*

7.3 From Tutte to Kirchoff

The proofs of these theorems are long and so I will merely sketch some parts. One of these is the connection between Tutte's directed Matrix-Tree theorem and Kirchoff's undirected version. The key idea is illustrated in Figures 7.4 and 7.5. If we want to count spanning trees in an undirected graph $G(V, E)$ we should first make a directed graph $H(V, E')$ that has the same vertex set as G, but has two directed edges—one running in each direction—for each of the edges in G. That is, if G has an undirected edge $e = (a, b)$ then H has both the directed edges (a, b) and (b, a).

Now we choose some arbitrary vertex v in H and count the spanning arborescences that have v as a root. It's not hard to see that each spanning *tree* in G corresponds to a unique v-rooted *arborescence* in H, and vice-versa. More formally, there is a bijection between the set of spanning trees in G and v-rooted spanning arborescences in H: see Figure 7.5. The keen reader might wish to write out a careful statement of how this bijection acts (that is, which tree gets matched with which arborescence).

Finally, note that for our directed graph H, which includes the edges (a, b) and (b, a) whenever the original, undirected graph contains (a, b), we have

$$\deg_{in}(v) = \deg_{out}(v) \;=\; \deg_G(v) \text{ for all } v \in V$$

where the in- and out-degrees are in H and $\deg_G(v)$ is in G. This means that the matrix L appearing in Tutte's theorem is equal, element-by-element, to the graph Laplacian appearing in Kirchoff's theorem. So if we use Tutte's approach to compute

the number of spanning arborescences in H, the result will be the same numerically as if we'd used Kirchoff's theorem to count spanning trees in G.

Lecture 8

Matrix-Tree Ingredients

This lecture introduces some ideas that we will need for the proof of the Matrix-Tree Theorem. Many of them should be familiar from Foundations of Pure Mathematics or Algebraic Structures.

Reading:
The material about permutations and the determinant of a matrix presented here is pretty standard and can be found in any number of places: the *Wikipedia* articles on *Determinant* (especially the section on $n \times n$ matrices) and *Permutation* are not bad places to start.

The remaining ingredient for the proof of the Matrix-Tree theorems is the *Principle of Inclusion/Exclusion*. It is covered in the first year module Foundations of Pure Mathematics, but it is also a standard technique in Combinatorics and so is discussed in many introductory books[1]. The *Wikipedia* article is, again, a good place to start. Finally, as a convenience for students from outside the School of Mathematics, I have included an example in Section 8.4.4 and an Appendix, Section 8.5, that provides full details of all the proofs.

8.1 Lightning review of permutations

First we need a few facts about permutations that you should have learned earlier in your degree.

Definition 8.1. *A* ***permutation*** *on n-objects is a bijection σ from the set $\{1, \ldots, n\}$ to itself.*

We'll express permutations as follows

$$\sigma = \begin{pmatrix} 1 & 2 & \ldots & n \\ \sigma(1) & \sigma(2) & \ldots & \sigma(n) \end{pmatrix}$$

where $\sigma(j)$ is the image of j under the bijection σ.

Definition 8.2. *The set* $\mathrm{fix}(\sigma)$ *is defined as*

$$\mathrm{fix}(\sigma) = \{j \mid \sigma(j) = j\}.$$

[1]See, for example, Dossey, Otto, Spence, and Vanden Eynden (2006), *Discrete Mathematics* or, for a short, clear account, Anderson (1974), *A First Course in Combinatorial Mathematics*.

8.1.1 The Symmetric Group S_n

One can turn the set of all permutations on n objects into a group by using *composition* (applying one function to the output of another) of permutations as the group multiplication. The resulting group is called the **symmetric group on n objects** or S_n and it has the following properties.

- The identity element is the permutation in which $\sigma(j) = j$ for all $1 \leq j \leq n$.
- S_n has $n!$ elements.

8.1.2 Cycles and sign

Definition 8.3 (Cycle permutations)**.** *A **cycle** is a permutation σ specified by a sequence of distinct integers $i_1, i_2, \ldots, i_\ell \in \{1, \ldots, n\}$ with the properties that*

- $\sigma(j) = j$ *if* $j \notin \{i_1, i_2, \ldots, i_\ell\}$
- $\sigma(i_j) = i_{j+1}$ *for* $1 \leq j < \ell$
- $\sigma(i_\ell) = i_1$.

*Here ℓ is the **length** of the cycle and a cycle with $\ell = 2$ is called a **transposition**.*

We'll express the cycle σ specified by the sequence $i_1, i_2, \ldots, i_\ell$ with the notation

$$\sigma = (i_1, i_2, \ldots, i_\ell).$$

and we'll say that two cycles $\sigma_1 = (i_1, i_2, \ldots, i_{\ell_1})$ and $\sigma_2 = (j_1, j_2, \ldots, j_{\ell_2})$ are **disjoint** if

$$\{i_1, \ldots i_{\ell_1}\} \cap \{j_1, \ldots j_{\ell_2}\} \;=\; \emptyset.$$

The main point about cycles is that they're like the "prime factors" of permutations in the following sense:

Proposition 8.4. *A permutation has a unique (up to reordering of the cycles) representation as a product of disjoint cycles.*

This representation is often referred to as the *cycle decomposition* of the permutation.

Finally, given the cycle decomposition of a permutation one can define a function that we will need in the next section.

Definition 8.5 (Sign of a permutation)**.** *The function* $\mathrm{sgn} : S_n \to \{\pm 1\}$ *can be computed as follows:*

- *If σ is the identity permutation, then* $\mathrm{sgn}(\sigma) = 1$.
- *If σ is a cycle of length ℓ then* $\mathrm{sgn}(\sigma) = (-1)^{\ell-1}$.
- *If σ has a decomposition into $k \geq 2$ disjoint cycles whose lengths are $\ell_1, \ldots, \ell_k$ then*

$$\mathrm{sgn}(\sigma) = (-1)^{L-k} \qquad \textit{where} \qquad L = \sum_{j=1}^{k} \ell_j.$$

This definition of $\mathrm{sgn}(\sigma)$ is equivalent to one that you may know from other courses:

$$\mathrm{sgn}(\sigma) \;=\; \begin{cases} 1 & \text{If } \sigma \text{ is the product of an even number of transpositions} \\ -1 & \text{otherwise} \end{cases}$$

8.2 Using graphs to find the cycle decomposition

Lest we forget graph theory completely, I'd like to conclude our review of permutations by constructing a certain graph that makes it easy to read-off the cycle decomposition of a permutation. The same construction establishes a bijection that maps permutations $\sigma \in S_n$ to subgraphs of K_n whose strongly-connected components are either isolated vertices or disjoint, directed cycles. This bijection is another key ingredient in the proof of Tutte's Matrix Tree Theorem.

Definition 8.6. *Given a permutation $\sigma \in S_n$, define the directed graph G_σ to have vertex set $V = \{1, \ldots, n\}$ and edge set*

$$E = \{(j, \sigma(j)) \mid j \in V \text{ and } \sigma(j) \neq j\}.$$

The following proposition then makes it easy to find the cycle decomposition of a permutation σ:

Proposition 8.7. *The cycle $(i_1, i_2, \ldots, i_l)$ appears in the cycle decomposition of σ if and only if the directed cycle defined by the vertex sequence*

$$(i_1, i_2, \ldots, i_l, i_1)$$

is a subgraph of G_σ.

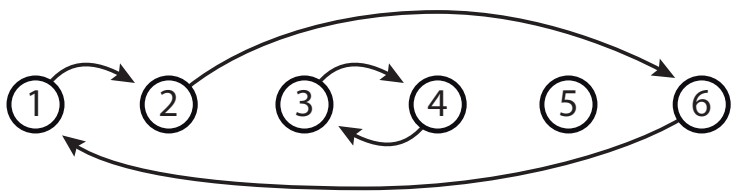
Example 8.8 (Graphical approach to cycle decomposition)**.** *Consider a permutation from S_6 given by*

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 6 & 4 & 3 & 5 & 1 \end{pmatrix}.$$

Then the digraph G_σ has vertex set $V = \{1, \ldots, 6\}$ and edge set

$$E = \{(1,2),\ (2,6),\ (3,4),\ (4,3),\ (6,1)\}.$$

A diagram for this graph appears below and clearly includes two disjoint directed cycles



Thus our permutation has $\mathrm{fix}(\sigma) = \{5\}$ *and its cycle decomposition is*

$$\sigma = (1,2,6)(3,4) = (3,4)(1,2,6) = (4,3)(2,6,1),$$

where I have included some versions where the order of the two disjoint cycles is switched and one where the terms within the cycle are written in a different (but equivalent) order.

With a little more work (that's left to the reader) one can prove that this graphical approach establishes a bijection between S_n and that family of subgraphs of K_n which consists of unions of disjoint cycles. The bijection sends the identity permutation to the subgraph consisting of n isolated vertices and sends a permutation σ that is the product of $k \geq 1$ disjoint cycles

$$\sigma = (i_{1,1},\, \ldots,\, i_{1,\ell_1}) \cdots (i_{k,1},\, \ldots,\, i_{k,\ell_k}) \tag{8.1}$$

to the subgraph G_σ that has vertex set $V = \{v_1, \ldots, v_n\}$ and whose edges are those that appear in the k disjoint, directed cycles $C_1, \ldots, C_k$, where C_j is the cycle specified by the vertex sequence

$$\left(v_{i_{j,1}},\, \ldots,\, v_{i_{j,\ell_j}},\, v_{i_{j,1}}\right). \tag{8.2}$$

In Eqns. (8.1) and (8.2) the notation $i_{j,r}$ is the vertex number of the r-th vertex in the j-th cycle, while ℓ_j is the length of the j-th cycle.

8.3 The determinant is a sum over permutations

The Matrix-Tree theorems relate the number of spanning trees or arborescences to the determinant of a matrix and so it should not be surprising that another of our key ingredients is a fact about determinants. The standard recursive approach to computing determinants—in which one computes the determinant of an $n \times n$ matrix as a sum over the determinants of $(n-1) \times (n-1)$ submatrices—is equivalent to a sum over permutations:

Proposition 8.9. *If A is an $n \times n$ matrix then*

$$\det(A) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{j=1}^{n} A_{j\sigma(j)}. \tag{8.3}$$

Some of you may have encountered this elsewhere, though most will be meeting it for the first time. I won't prove it, as that would be too much of a diversion from graphs, but Eqn. (8.3) has the blessing of *Wikipedia*, where it is attributed to Leibniz, and proofs appear in many undergraduate algebra texts[2]. The very keen reader could also construct an inductive proof herself, starting from the familiar recursive formula.

Finally, I'll demonstrate that it works for the two smallest nontrivial examples.

[2] I found one in I.N. Herstein (1975), *Topics in Algebra*, 2nd ed., Wiley.

Example 8.10 (2×2 matrices)**.** *First we need a list of the elements of S_2 and their signs:*

Name	σ	$\mathrm{sgn}(\sigma)$
σ_1	$\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}$	1
σ_2	$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$	-1

Then we can compute the determinant of a 2×2 matrix in the usual way

$$\det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

and then again using Eqn (8.3)

$$\begin{aligned} \det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} &= \sum_{k=1}^{2} \mathrm{sgn}(\sigma_k) \prod_{j=1}^{n} a_{j\sigma_k(j)} \\ &= \mathrm{sgn}(\sigma_1) \times a_{1\sigma_1(1)}a_{2\sigma_1(2)} + \mathrm{sgn}(\sigma_2) \times a_{1\sigma_2(1)}a_{2\sigma_2(2)} \\ &= (1) \times a_{11}a_{22} + (-1) \times a_{12}a_{21} \\ &= a_{11}a_{22} - a_{12}a_{21} \end{aligned}$$

Example 8.11 (3×3 matrices)**.** *Table 8.1 lists the elements of S_3 in the form we'll need. First, we compute the determinant in the usual way, a tedious but straightforward business.*

$$\det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} =$$

$$a_{11} \times \det \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} - a_{12} \times \det \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} + a_{13} \times \det \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

$$= a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31}$$

Then we calculate again using Eqn (8.3) and the numbering scheme for the elements of S_3 that's shown in Table 8.1.

$$\begin{aligned} \det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} &= \sum_{k=1}^{6} \mathrm{sgn}(\sigma_k) \prod_{j=1}^{n} a_{j\sigma_k(j)} \\ &= \mathrm{sgn}(\sigma_1) \times a_{1\sigma_1(1)}a_{2\sigma_1(2)}a_{3\sigma_1(3)} + \cdots + \mathrm{sgn}(\sigma_6) \times a_{1\sigma_6(1)}a_{2\sigma_6(2)}a_{3\sigma_6(3)} \\ &= (1) \times a_{11}a_{22}a_{33} + (-1) \times a_{11}a_{23}a_{32} + (-1) \times a_{12}a_{21}a_{33} \\ &+ (1) \times a_{12}a_{23}a_{31} + (1) \times a_{13}a_{21}a_{32} + (-1) \times a_{13}a_{22}a_{31} \\ &= a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} \end{aligned}$$

σ_k	$\text{fix}(\sigma_k)$	Cycle Decomposition	$\text{sgn}(\sigma_k)$
$\sigma_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\{1,2,3\}$	–	1
$\sigma_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$	$\{1\}$	$(2,3)$	-1
$\sigma_3 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$	$\{3\}$	$(1,2)$	-1
$\sigma_4 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$	$\emptyset$	$(1,2,3)$	1
$\sigma_5 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$	$\emptyset$	$(3,2,1)$	1
$\sigma_6 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$	$\{2\}$	$(1,3)$	-1

Table 8.1: *The cycle decompositions of all the elements in S_3, along with the associated functions* $\text{sgn}(\sigma)$ *and* $\text{fix}(\sigma)$.

8.4 The Principle of Inclusion/Exclusion

The remaining ingredient for the proof of the Matrix-Tree theorems is the *Principle of Inclusion/Exclusion*. As it is covered in a core first year module, none of the proofs in the rest of this lecture are examinable.

8.4.1 A familiar example

Suppose we have some finite "universal" set U and two subsets, $X_1 \subseteq U$ and $X_2 \subseteq U$. If the subsets are disjoint then it's easy to work out the number of elements in their union:

$$X_1 \cap X_2 = \emptyset \;\Rightarrow\; |X_1 \cup X_2| = |X_1| + |X_2|.$$

The case where the subsets have a non-empty intersection provides the simplest instance of the Principle of Inclusion/Exclusion. You may already know a similar result from Probability.

Lemma 8.12 (Inclusion/Exclusion for two sets)**.** *If X_1 and X_2 are finite sets then*

$$|X_1 \cup X_2| = |X_1| + |X_2| - |X_1 \cap X_2|. \tag{8.4}$$

Note that this formula, which is illustrated in Figure 8.1, works even $X_1 \cap X_2 = \emptyset$, as then $|X_1 \cap X_2| = 0$. The proof of this lemma appears in the Appendix, in Section 8.5.1.

8.4.2 Three subsets

Before moving to the general case, let's consider one more small example, this time with three subsets X_1, X_2 and X_3: we can handle this case by clever use

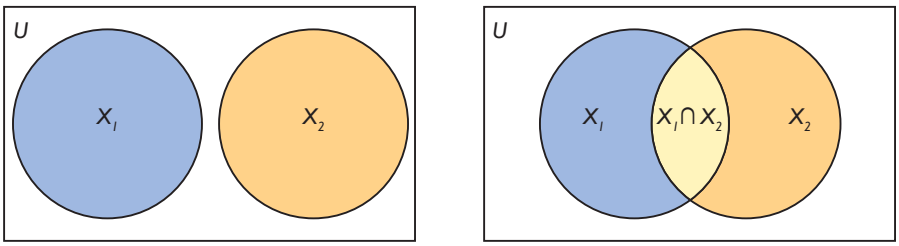


Figure 8.1: *In the example at left* $X_1 \cap X_2 = \emptyset$*, so* $|X_1 \cup X_2| = |X_1| + |X_2|$*, but in the example at right* $X_1 \cap X_2 \neq \emptyset$ *and so* $|X_1 \cup X_2| = |X_1| + |X_2| - |X_1 \cap X_2| < |X_1| + |X_2|$.

of Lemma 8.12 from the previous section. If we regard $(X_1 \cup X_2)$ as a single set and X_3 as a second set, then Eqn. (8.4) says

$$\begin{aligned}
|(X_1 \cup X_2) \cup X_3| &= |(X_1 \cup X_2)| + |X_3| - |(X_1 \cup X_2) \cap X_3| \\
&= (|X_1| + |X_2| - |X_1 \cap X_2|) + |X_3| - |(X_1 \cup X_2) \cap X_3| \\
&= |X_1| + |X_2| + |X_3| - |X_1 \cap X_2| - |(X_1 \cup X_2) \cap X_3|
\end{aligned}$$

Focusing on the final term, we can use standard relations about unions and intersections to say

$$(X_1 \cup X_2) \cap X_3 = (X_1 \cap X_3) \cup (X_2 \cap X_3).$$

Then, applying Eqn. (8.4) to the pair of sets $(X_1 \cap X_3)$ and $(X_2 \cap X_3)$, we obtain

$$\begin{aligned}
|(X_1 \cup X_2) \cap X_3| &= |(X_1 \cap X_3) \cup (X_2 \cap X_3)| \\
&= |X_1 \cap X_3| + |X_2 \cap X_3| - |(X_1 \cap X_3) \cap (X_2 \cap X_3)| \\
&= |X_1 \cap X_3| + |X_2 \cap X_3| - |X_1 \cap X_2 \cap X_3|
\end{aligned}$$

where, in going from the second line to the third, we have used

$$(X_1 \cap X_3) \cap (X_2 \cap X_3) = X_1 \cap X_2 \cap X_3.$$

Finally, putting all these results together, we obtain the analogue of Eqn. (8.4) for three subsets:

$$\begin{aligned}
|(X_1 \cup X_2) \cup X_3| &= |X_1| + |X_2| + |X_3| - |X_1 \cap X_2| - |(X_1 \cup X_2) \cap X_3| \\
&= |X_1| + |X_2| + |X_3| - |X_2 \cap X_3| \\
&\quad - (|X_1 \cap X_3| + |X_2 \cap X_3| - |(X_1 \cap X_2 \cap X_3|) \\
&= (|X_1| + |X_2| + |X_3|) \\
&\quad - (|X_1 \cap X_2| + |X_1 \cap X_3| + |X_2 \cap X_3|) \\
&\quad + |X_1 \cap X_2 \cap X_3|. \qquad (8.5)
\end{aligned}$$

Figure 8.2 helps make sense of this formula and prompts the following observations:

- Elements of $X_1 \cup X_2 \cup X_3$ that belong to exactly one of the X_j are counted exactly once by the sum $(|X_1| + |X_2| + |X_3|)$ and do not contribute to any of the terms involving intersections.

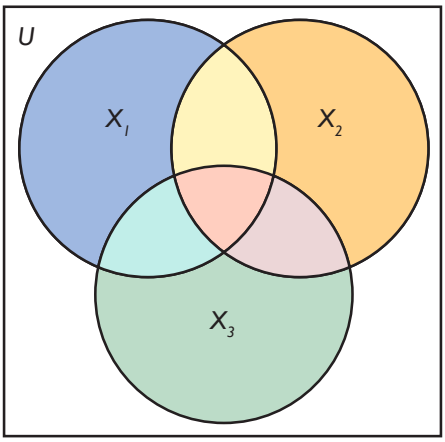


Figure 8.2: *In the diagram above all of the intersections appearing in Eqn. (8.5) are nonempty.*

- Elements of $X_1 \cup X_2 \cup X_3$ that belong to exactly two of the X_j are double-counted by the sum, $(|X_1| + |X_2| + |X_3|)$, but this double-counting is corrected by the term involving two-fold intersections.
- Finally, elements of $X_1 \cup X_2 \cup X_3$ that belong to all three of the sets are triple-counted by the initial sum $(|X_1| + |X_2| + |X_3|)$. This triple-counting is then completely cancelled by the term involving two-fold intersections. Then, finally, this cancellation is repaired by the final term, which counts each such element once.

8.4.3 The general case

The *Principle of Inclusion/Exclusion* generalises the results in Eqns. (8.1) and (8.5) to unions of arbitrarily many subsets.

Theorem 8.13 (The Principle of Inclusion/Exclusion)**.** *If U is a finite set and $\{X_j\}_{j=1}^n$ is a collection of n subsets, then*

$$\begin{aligned}
\left|\bigcup_{j=1}^n X_j\right| &= |X_1 \cup \cdots \cup X_n| \\
&= |X_1| + \cdots + |X_n| \\
&\quad - |X_1 \cap X_2| - \cdots - |X_{n-1} \cap X_n| \\
&\quad + |X_1 \cap X_2 \cap X_3| + \cdots + |X_{n-2} \cap X_{n-1} \cap X_n| \\
&\quad \vdots \\
&\quad + (-1)^{m-1} \sum_{1 \le i_1 \le \cdots \le i_m \le n} |X_{i_1} \cap \cdots \cap X_{i_m}| \\
&\quad \vdots \\
&\quad + (-1)^{n-1} |X_1 \cap \cdots \cap X_n|
\end{aligned} \tag{8.6}$$

or, more concisely,

$$|X_1 \cup \cdots \cup X_n| \;=\; \sum_{I \subseteq \{1,\ldots,n\},\, I \neq \emptyset} (-1)^{|I|-1} \left| \bigcap_{i \in I} X_i \right| \tag{8.7}$$

The proof of this result appears in the Appendix, in Section 8.5.2 below.

8.4.4 An example

How many of the integers n with $1 \leq n \leq 150$ are coprime to 70? This is a job for the Principle of Inclusion/Exclusion. First note that the prime factorization of 70 is $70 = 2 \times 5 \times 7$. Now consider a universal set $U = \{1, \ldots, 150\}$ and the three subsets X_1, X_2 and X_3 consisting of multiples of 2, 5 and 7, respectively. A member of U that shares a prime factor with 70 belongs to at least one of the X_j and so the number we're after is

$$\begin{aligned}
|U| - |X_1 \cup X_2 \cup X_3| &= |U| - (|X_1| + |X_2| + |X_3|) \\
&\quad + (X_1 \cap X_2| + |X_1 \cap X_3| + |X_2 \cap X_3|) \\
&\quad - \; |X_1 \cap X_2 \cap X_3|. \\
&= 150 - (75 + 30 + 21) + (15 + 10 + 4) - 2 \\
&= 150 - 126 + 29 - 2 \\
&= 51
\end{aligned} \tag{8.8}$$

where I have used the numbers in Table 8.2 which lists the various cardinalities that we need.

Set	Description	Cardinality
X_1	multiples of 2	75
X_2	multiples of 5	30
X_3	multiples of 7	21
$X_1 \cap X_2$	multiples of 10	15
$X_1 \cap X_3$	multiples of 14	10
$X_2 \cap X_3$	multiples of 35	4
$X_1 \cap X_2 \cap X_3$	multiples of 70	2

Table 8.2: *The sizes of the various intersections needed for the calculation in Eqn. (8.8).*

8.5 Appendix: Proofs for Inclusion/Exclusion

The proofs in this section will not appear on the exam, but are provided for those who are interested or for whom the subject is new.

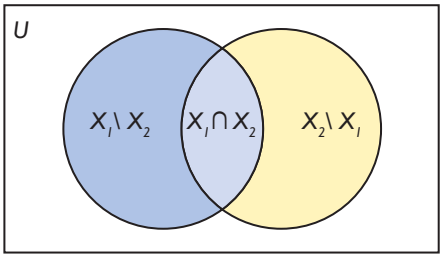


Figure 8.3: *Here $X_1 \backslash X_2$ and $X_1 \cap X_2$ are shown in shades of blue, while $X_2 \backslash X_1$ is in yellow.*

8.5.1 Proof of Lemma 8.12, the case of two sets

Recall that X_1 and X_2 are subsets of some universal set U and that we seek to prove that $|X_1 \cup X_2| = |X_1| + |X_2| - |X_1 \cap X_2|$.

Proof. Note that the sum $|X_1|+|X_2|$ counts each member of the intersection $X_1 \cap X_2$ twice, once as a member of X_1 and then again as a member of X_2. Subtracting $|X_1 \cap X_2|$ corrects for this double-counting. Alternatively, for those who prefer proofs that look more like calculations, begin by defining

$$X_1 \backslash X_2 = \{x \in U \mid x \in X_1, \text{ but } x \notin X_2\}.$$

Then, as is illustrated in Figure 8.3, $X_1 = (X_1 \backslash X_2) \cup (X_1 \cap X_2)$. Further, the sets $X_1 \backslash X_2$ and $X_1 \cap X_2$ are disjoint by construction, so

$$|X_1| \;=\; |X_1 \backslash X_2| + |X_1 \cap X_2| \qquad \text{or} \qquad |X_1 \backslash X_2| \;=\; |X_1| - |X_1 \cap X_2|. \tag{8.9}$$

Similarly, $X_1 \backslash X_2$ and X_2 are disjoint and $X_1 \cup X_2 = (X_1 \backslash X_2) \cup X_2$ so

$$\begin{aligned} |X_1 \cup X_2| &= |X_1 \backslash X_2| + |X_2| \\ &= |X_1| - |X_1 \cap X_2| + |X_2| \\ &= |X_1| + |X_2| - |X_1 \cap X_2| \end{aligned}$$

where, in passing from the first line to the second, we have used (8.9). The last line is the result we were trying to prove, so we are finished. □

8.5.2 Proof of Theorem 8.13

One can prove this result in at least two ways:

- by induction, with a calculation that is essentially the same as the one used to obtain the $n = 3$ case—Eqn. (8.5)—from the $n = 2$ one—Eqn. (8.4);
- by showing that each $x \in X_1 \cup \cdots \cup X_n$ contributes exactly one to the sum on the right hand side of Eqn. (8.7).

The first approach is straightforward, if a bit tedious, but the second is more interesting and is the one discussed here.

The key idea is to think of the the elements of $X_1 \cup \cdots \cup X_n$ individually and ask what each one contributes to the sum in Eqn. (8.7). Suppose that an element $x \in X_1 \cup \cdots \cup X_n$ belongs to exactly ℓ of the subsets, with $1 \leq \ell \leq n$: we will prove that x makes a net contribution of 1. For the sake of concreteness, we'll say $x \in X_{i_1}, \ldots, X_{i_\ell}$ where $i_1, \ldots, i_\ell$ are distinct elements of $\{1, \ldots, n\}$.

- As we've assumed that x belongs to exactly ℓ of the subsets X_j, it contributes a total of ℓ to the first row, $|X_1| + \cdots + |X_n|$, of the long sum in Eqn. (8.6).
- Further, x contributes a total of $-\binom{\ell}{2}$ to the sum in the row involving two-way intersections
$$-|X_1 \cap X_2| - \cdots - |X_{n-1} \cap X_n|.$$
To see this, note that if $x \in X_j \cap X_k$ then both j and k must be members of the set $\{i_1, \ldots, i_\ell\}$.
- Similar arguments show that if $k \leq \ell$, then x contributes a total of
$$(-1)^{k-1}\binom{\ell}{k} = (-1)^{k-1}\left(\frac{\ell!}{k!\,(\ell-k)!}\right)$$
to the sum in the row of Eqn. (8.6) that involves k-fold intersections.
- Finally, for $k > \ell$ there are no k-fold intersections that contain x and so x makes a contribution of zero to the corresponding rows in Eqn. (8.6).

Putting these observations together we see that x make a net contribution of

$$\ell - \binom{\ell}{2} + \binom{\ell}{3} - \ \ldots \ + (-1)^{\ell-1}\binom{\ell}{\ell} \tag{8.10}$$

This sum can be made to look more familiar by considering the following application of the Binomial Theorem:

$$\begin{aligned} 0 &= (1-1)^\ell \\ &= \sum_{j=0}^{\ell} (-1)^j (1)^{\ell-j} \binom{\ell}{j} \\ &= 1 - \ell + \binom{\ell}{2} - \binom{\ell}{3} + \ \ldots \ + (-1)^\ell \binom{\ell}{\ell}. \end{aligned}$$

Thus

$$0 \;=\; 1 - \left[\ell - \binom{\ell}{2} + \binom{\ell}{3} - \;\ldots\; + (-1)^{\ell-1}\binom{\ell}{\ell}\right]$$

or

$$\ell - \binom{\ell}{2} + \binom{\ell}{3} - \;\ldots\; + (-1)^{\ell-1}\binom{\ell}{\ell} \;=\; 1.$$

The left hand side here is the same as the sum in Eqn. (8.10) and so we've established that any x which belongs to exactly ℓ of the subsets X_j makes a net contribution of 1 to the sum on the right hand side of Eqn. (8.7). And as every $x \in X_1 \cup \cdots \cup X_n$ must belong to at least one of the X_j, this establishes the Principle of Inclusion/Exclusion.

8.5.3 Alternative proof

Students who like proofs that look more like calculations may prefer to reformulate the arguments from the previous section in terms of *characteristic functions* (sometimes also called *indicator functions*) of sets. If we define $\mathbb{1}_X : U \to \{0,1\}$ by

$$\mathbb{1}_X(s) = \begin{cases} 1 & \text{if } s \in X \\ 0 & \text{otherwise} \end{cases}$$

then we can calculate $|X|$ for a subset $X \subseteq U$ as follows:

$$\begin{aligned} |X| &= \sum_{x \in U} \mathbb{1}_X(x) \\ &= \left(\sum_{x \in X} \mathbb{1}_X(x)\right) + \left(\sum_{x \notin X} \mathbb{1}_X(x)\right) \\ &= \sum_{x \in X} \mathbb{1}_X(x) \end{aligned} \tag{8.11}$$

where, in passing from the second to third lines, I have dropped the second sum because all its terms are zero.

Then the Principle of Inclusion/Exclusion is equivalent to

$$\begin{aligned} \sum_{x \in X_1 \cup \cdots \cup X_n} \mathbb{1}_{X_1 \cup \cdots \cup X_n}(x) &= \sum_{I \subseteq \{1,\ldots,n\},\, I \neq \emptyset} (-1)^{|I|-1} \left| \bigcap_{i \in I} X_i \right| \\ &= \sum_{I \subseteq \{1,\ldots,n\},\, I \neq \emptyset} (-1)^{|I|-1} \sum_{x \in \bigcap_{i \in I} X_i} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) \\ &= \sum_{k=1}^{n} (-1)^{k-1} \sum_{I \subseteq \{1,\ldots,n\},\, |I| = k} \left(\sum_{x \in \bigcap_{i \in I} X_i} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) \right) \end{aligned}$$

which I have obtained by using of Eqn. (8.11) to replace terms in Eqn. (8.7) with the corresponding sums of values of characteristic functions.

We can then rearrange the expression on the right, first expanding the ranges of the sums over elements of k-fold intersections (this doesn't change the result since $\mathbb{1}_X(x) = 0$ for $x \notin X$) and then interchanging the order of summation so that the sum over elements comes first. This calculation proves that the Principle of Inclusion/Exclusion is equivalent to the following:

$$\begin{aligned}
&\sum_{x \in X_1 \cup \cdots \cup X_n} \mathbb{1}_{X_1 \cup \cdots \cup X_n}(x) \\
&\qquad = \sum_{k=1}^{n} (-1)^{k-1} \sum_{I \subseteq \{1,\ldots,n\},\, |I|=k} \left(\sum_{x \in X_1 \cup \cdots \cup X_n} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) \right) \\
&\qquad = \sum_{x \in X_1 \cup \cdots \cup X_n} \sum_{k=1}^{n} (-1)^{k-1} \sum_{I \subseteq \{1,\ldots,n\},\, |I|=k} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) \qquad (8.12)
\end{aligned}$$

Arguments similar to those in Section 8.5.2 then establish the following results, the last of which, along with Eqn. (8.12), proves Theorem 8.13.

Proposition 8.14. *If an element $x \in X_1 \cup \cdots \cup X_n$ belongs to exactly ℓ of the sets $\{X_j\}_{j=1}^n$ then for $k \leq \ell$ we have*

$$\sum_{I \subseteq \{1,\ldots,n\},\, |I|=k} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) = \binom{\ell}{k} = \frac{\ell!}{k!\,(\ell-k)!}$$

while if $k > \ell$

$$\sum_{I \subseteq \{1,\ldots,n\},\, |I|=k} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) = 0$$

Proposition 8.15. *For an element $x \in X_1 \cup \cdots \cup X_n$ we have*

$$\sum_{k=1}^{n} (-1)^{k-1} \sum_{I \subseteq \{1,\ldots,n\},\, |I|=k} \mathbb{1}_{\bigcap_{i \in I} X_i}(x) \;=\; \sum_{k=1}^{n} (-1)^{k-1} \binom{\ell}{k} \;=\; 1.$$

Lemma 8.16. *The characteristic function $\mathbb{1}_{X_1 \cup \cdots \cup X_n}$ of the set $X_1 \cup \cdots \cup X_n$ satisfies*

$$\mathbb{1}_{X_1 \cup \cdots \cup X_n}(x) \;=\; \sum_{k=1}^{n} (-1)^{k-1} \sum_{I \subseteq \{1,\ldots,n\},\, |I|=k} \mathbb{1}_{\bigcap_{i \in I} X_i}(x).$$

Lecture 9

Proof of Tutte's Matrix-Tree Theorem

The proof here is derived from a terse account in the lecture notes from a course on Algebraic Combinatorics taught by Lionel Levine at MIT in Spring 2011.[1] I studied them with Samantha Barlow, a former Discrete Maths student who did a third-year project with me in 2011-12.

Reading:
I don't know of any textbook accounts of the proof given here, but the intrepid reader might like to look at the following two articles, both of which make the connection between the Principle of Inclusion/Exclusion and Tutte's Matrix Tree theorem.

- J.B. Orlin (1978), Line-digraphs, arborescences, and theorems of Tutte and Knuth, *Journal of Combinatorial Theory, Series B*, **25**(2):187–198. DOI: 10.1016/0095-8956(78)90038-2
- S. Chaiken (1983), A combinatorial proof of the all minors matrix tree theorem, *SIAM Journal on Algebraic and Discrete Methods*, **3**:319–329. DOI: 10.1137/0603033

9.1 Single predecessor graphs

Before we plunge into the proof itself I'd like to define a certain family of graphs that includes, but is larger than, the family of spanning arborescences.

Definition 9.1. *A **single predecessor graph ("spreg") with distinguished vertex** v in a digraph $G(V, E)$ is a subgraph $T(V, E')$ (Note that T and G have the same vertex set) in which each vertex other than the distinguished vertex v has exactly one predecessor while v itself has no predecessors. Equivalently,*

$$\deg_{in}(v) = 0 \qquad \textit{and} \qquad \deg_{in}(u) = 1 \ \forall u \neq v \in V.$$

[1] Dr. Levine seems to have moved to a post at Cornell, but his notes were still available via the link above in January 2020.

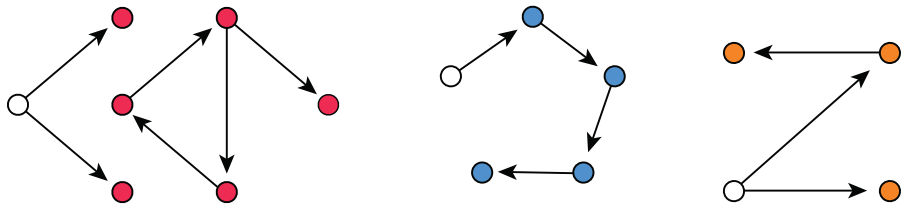


Figure 9.1: *Three examples of* ***s****ingle* ***pre****decessor* ***g****raphs (spregs). In each the distinguished vertex is white, while the other vertices, which all have* $\deg_{in}(u) = 1$*, are shaded in other colours. The example at left, has multiple weakly connected components, while the other two are arborescences.*

Figure 9.1 includes several examples of spregs, including two that are arborescences, which prompts the following proposition:

Proposition 9.2 (Spanning arborescences are spregs)**.** *If* $T(V, E')$ *is a spanning arborescence for* $G(V, E)$ *with root* v*, then it is also a spreg with distinguished vertex* v*.*

Proof. By definition, G and T share the same vertex set, so all we need check is that the vertices $u \neq v$ in T have a single predecessor. Recall that an arborescence rooted at v is a directed graph $T(V, E)$ such that

(i) Every vertex $u \neq v$ is accessible form v. That is, there is a directed path from v to every other vertex.

(ii) T becomes an ordinary, undirected tree if we ignore the directedness of the edges.

The proposition consists of two separate claims: that $\deg_{in}(v) = 0$ and that $\deg_{in}(u) = 1 \; \forall u \neq v \in V$. We'll prove both by contradiction.

Suppose that $\deg_{in}(v) > 0$: it's then easy to see that T must include a directed cycle. Consider one of v's predecessors—call is u_0. It is accessible from v, so there is a directed path from v to u_0. And u_0 is a predecessor of v, so there is also a directed edge $(u_0, v) \in E$. If we append this edge to the end of the path, we get a directed path from v back to itself. This contradicts the second property of an arborescence and so we must have $\deg_{in}(v) = 0$.

The proof for the second part of the proposition is illustrated in Figure 9.2. Suppose that $\exists \; u \neq v \in V$ such that $\deg_{in}(u) \geq 2$ and choose two distinct predecessors of u: call them v_1 and v_2 and note that one of them may be the root vertex v. Now consider the directed paths from v to v_1 and v_2. In the undirected version of T these paths, along with the edges (v_1, u) and (v_2, u), must include a cycle, which contradicts the second property of an arborescence. □

The examples in Figure 9.1 make it clear that there are other kinds of spregs besides spanning arborescences, but there aren't *that* many kinds:

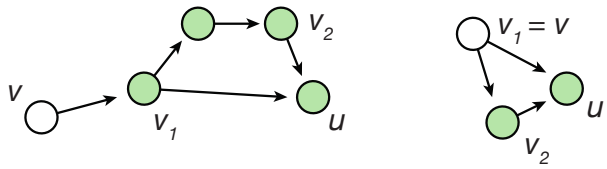


Figure 9.2: *Two examples to illustrate the second part of the proof that an arborescence is a spreg. If one ignores the directedness of the edges in the graphs above, both contain cycles.*

Proposition 9.3 (Characterising spregs)**.** *A spreg with distinguished vertex v consists of an arborescence rooted at v, plus zero or more disjoint weakly connected components, each of which contains a single directed cycle.*

Note that the arborescence mentioned in the Proposition is not necessarily a spanning one: the leftmost graph in Figure 9.1 consists of a small, non-spanning arborescence and a second component that contains a cycle.

The reasoning needed to prove this proposition is similar to that for the previous one and so is left to the Problem Sets.

This lemma is one of the key ingredients in the proof of Tutte's Matrix-Tree Theorem. The idea is to first note that a spanning arborescence is a spreg. We then count the spanning arborescences contained in a graph by first counting *all* the spregs, then use the Principle of Inclusion/Exclusion to count—and subtract away—those spregs that contain one or more cycles.

9.2 Counting spregs with determinants

Recall that we're trying to prove

Theorem 1 (Tutte's Directed Matrix-Tree Theorem, 1948)**.** *If $G(V, E)$ is a digraph with vertex set $V = \{v_1, \ldots, v_n\}$ and L is an $n \times n$ matrix whose entries are given by*

$$L_{ij} \;=\; \begin{cases} \deg_{in}(v_j) & \text{If } i = j \\ -1 & \text{If } i \neq j \text{ and } (v_i, v_j) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{9.1}$$

then the number N_j of spanning arborescences with root at v_j is

$$N_j \;=\; \det(\hat{L}_j)$$

where $\hat{L}_j$ is the matrix produced by deleting the j-th row and column from L.

First note that—because we can always renumber the vertices before we apply the theorem—it is sufficient to prove the result for the case with root vertex $v = v_n$. Now consider the representation of $\det(\hat{L}_n)$ as a sum over permutations:

$$\det(\hat{L}_n) \;\equiv\; \det(\mathcal{L}) \;=\; \sum_{\sigma \in S_{n-1}} \operatorname{sgn}(\sigma) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)}. \tag{9.2}$$

Predecessor of			Is a spanning
v_1	v_2	v_3	arboresence?
v_2	v_1	v_2	No
v_2	v_3	v_2	No
v_2	v_4	v_2	Yes
v_4	v_1	v_2	Yes
v_4	v_3	v_2	No
v_4	v_4	v_2	Yes

Table 9.1: *Each row here corresponds to one of the spregs in Figure 9.3.*

where I have introduced the notation $\mathcal{L} \equiv \hat{L}_n$ to avoid the confusion of having two kinds of subscripts on $\hat{L}_n$. This means that $\mathcal{L}$ is an $(n-1) \times (n-1)$ matrix in which

$$\mathcal{L}_{ij} = L_{ij},$$

where L_{ij} is the i, j entry in the matrix L defined by Eqn. (9.1) in the statement of Tutte's theorem.

9.2.1 Counting spregs

In this section we'll explore two examples that illustrate a connection between terms in the sum for $\det(\mathcal{L})$ and the business of counting various kinds of spregs.

The identity term: counting all spregs

In the case where $\sigma = \mathrm{id}$, so that $\sigma(j) = j$ for all j, we have $\mathrm{sgn}(\sigma) = 1$ and

$$\prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} = \prod_{j=1}^{n-1} \mathcal{L}_{jj} = \prod_{j=1}^{n-1} \deg_{in}(v_j). \tag{9.3}$$

This product is also equal to the total number of spregs in $G(V, E)$ that have distinguished vertex v_n. To see why, look back at the definition of a spreg and think about what we'd need to do if we wanted to write down a complete list of these spregs. We could specify a spreg by listing the single predecessor for each vertex other than v_n in a table like the one below

Vertex	v_1	v_2	v_3
Predecessor	v_2	v_1	v_2

which describes one of the spregs rooted at v_4 contained in the four-vertex graph shown in Figure 9.3. And if we wanted to list *all* the four-vertex spregs contained in this graph we could start by assembling the predecessor lists of all the vertices other than the distinguished vertex,

$$P_1 = \{v_2, v_4\},\ P_2 = \{v_1, v_3, v_4\} \text{ and } P_3 = \{v_2\},$$

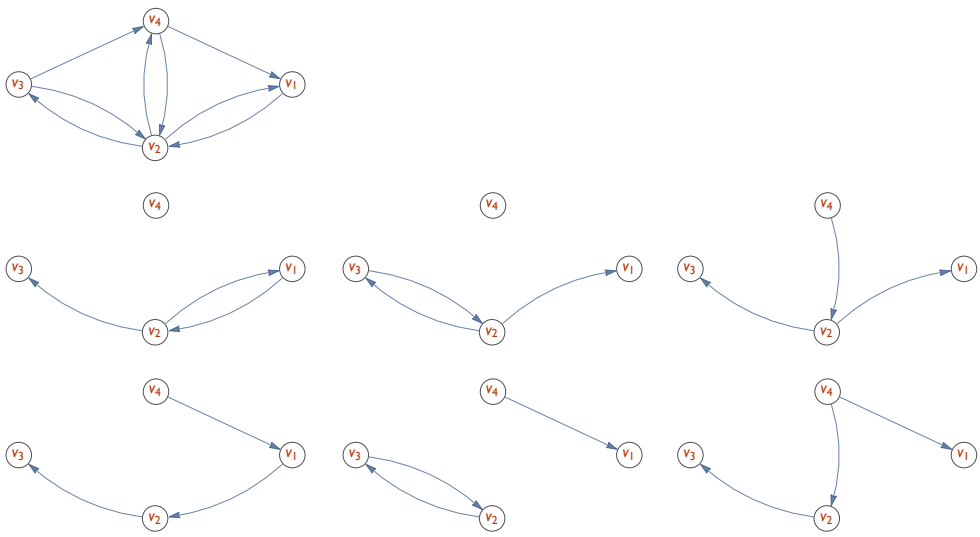


Figure 9.3: *The graph $G(V, E)$ at upper left contains six spregs with distinguished vertex v_4, all of which are shown in the two rows below. Three of them are spanning arborescences rooted at v_4, while the three others contain cycles.*

where P_j lists the predecessors of v_j. Then, to specify a spreg with distinguished vertex v_4 we would choose one entry from each of the predecessor lists, meaning that there are

$$|P_1| \times |P_2| \times |P_3| = \deg_{in}(v_1) \times \deg_{in}(v_2) \times \deg_{in}(v_3) = 2 \times 3 \times 1 = 6$$

such spregs in total. All six possibilities are listed in Table 9.1 and illustrated in Figure 9.3. The equation above also emphasises that $|P_j| = \deg_{in}(v_j)$ and so makes the connection with the product in Eqn. (9.3).

Terms that count spregs containing a single directed cycle

In the case where the permutation σ contains a single cycle of length ℓ, so that

$$\sigma = (i_1, \ldots, i_\ell),$$

we have $\mathrm{sgn}(\sigma) = (-1)^{\ell-1}$ and

$$\begin{aligned}\prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} &= \left(\prod_{j \in \mathrm{fix}(\sigma)} \mathcal{L}_{jj}\right) \times \left(\prod_{k=1}^{\ell} \mathcal{L}_{i_k i_{k+1}}\right) \\ &= \left(\prod_{j \in \mathrm{fix}(\sigma)} \deg_{in}(v_j)\right) \times \left(\prod_{k=1}^{\ell} \mathcal{L}_{i_k i_{k+1}}\right)\end{aligned}$$

where the indices i_k are to be understood periodically, so $i_{\ell+1} = i_1$. The factors $\mathcal{L}_{i_k i_{k+1}}$ in the second of the two products above are off-diagonal entries of $\mathcal{L} = \hat{L}_n$ and thus satisfy

$$\mathcal{L}_{i_k i_{k+1}} = \begin{cases} -1 & \text{if } (v_{i_k}, v_{i_{k+1}}) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Thus if one or more of the edges $(v_{i_k}, v_{i_{k+1}})$ is absent from the graph we have

$$\prod_{k=1}^{\ell} \mathcal{L}_{i_k i_{k+1}} = 0,$$

but if all the edges $(v_{i_k}, v_{i_{k+1}})$ are present we have can make the following observations:

- the graph contains a directed cycle given by the vertex sequence

$$(v_{i_1},\, \ldots,\, v_{i_\ell},\, v_{i_1});$$

- $\mathcal{L}_{i_k i_{k+1}} = -1$ for all $1 \leq k \leq \ell$ and so we have

$$\begin{aligned}
\operatorname{sgn}(\sigma) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} &= (-1)^{\ell-1} \left(\prod_{j \in \operatorname{fix}(\sigma)} \deg_{in}(v_j) \right) \times \prod_{k=1}^{\ell} (-1) \\
&= (-1)^{\ell-1} \left(\prod_{j \in \operatorname{fix}(\sigma)} \deg_{in}(v_j) \right) (-1)^{\ell} \\
&= (-1)^{2\ell-1} \prod_{j \in \operatorname{fix}(\sigma)} \deg_{in}(v_j) \\
&= - \prod_{j \in \operatorname{fix}(\sigma)} \deg_{in}(v_j). \qquad (9.4)
\end{aligned}$$

Arguments similar to those in the previous section then show that the product $\prod_{j \in \operatorname{fix}(\sigma)} \deg_{in}(v_j)$ in Eqn. (9.4) counts the number of ways to choose predecessors for those vertices that aren't part of the cycle. We can summarise all these ideas with the following pair of results:

Proposition 9.4. *For a permutation $\sigma \in S_{n-1}$ consisting of a single cycle*

$$\sigma = (i_1,\, \ldots,\, i_\ell)$$

define an associated directed cycle C_σ specified by the vertex sequence $(v_{i_1},\, \ldots,\, v_{i_\ell},\, v_{i_1})$. Then the term in $\det(\mathcal{L})$ corresponding to σ satisfies

$$\operatorname{sgn}(\sigma) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} = \begin{cases} -\displaystyle\prod_{j \in \operatorname{fix}(\sigma)} \deg_{in}(v_j) & \text{if } C_\sigma \subseteq G(V,E) \text{ and } \operatorname{fix}(\sigma) \neq \emptyset \\ -1 & \text{if } C_\sigma \subseteq G(V,E) \text{ and } \operatorname{fix}(\sigma) = \emptyset \\ 0 & \text{if } C_\sigma \not\subseteq G(V,E) \end{cases}$$

Corollary 9.5. *For σ and C_σ as in Proposition 9.4*

$$\left| \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} \right| = \left| \{ \textit{spregs containing } C_\sigma \} \right|.$$

σ	C_σ	$C_\sigma \subseteq G$?	$\left\lvert\prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)}\right\rvert$	Number of spregs containing C_σ
(1,2)	(v_1, v_2, v_1)	Yes	$\deg_{in}(v_3) = 1$	1
(1,3)	(v_1, v_3, v_1)	No	$\deg_{in}(v_2) \times 0$	0
(2,3)	(v_2, v_3, v_2)	Yes	$\deg_{in}(v_1) = 2$	2
(1,2,3)	(v_1, v_2, v_3, v_1)	No	0	0
(1,3,2)	(v_1, v_3, v_2, v_1)	No	0	0

Table 9.2: *The results of using Corollary 9.5 to count spregs containing the various cycles C_σ associated with the non-identity elements of S_3. The right column lists the number one gets by direct counting of the spregs shown in Figure 9.3.*

9.2.2 An example

Before pressing on to generalise the results of the previous section to arbitrary permutations, let's see what Corollary 9.5 allows us to say about the graph in Figure 9.3. There $G(V, E)$ is a digraph on four vertices, so the determinant that comes into Tutte's theorem is that of L_4, a three-by-three matrix. We've already seen that if $\sigma = \mathrm{id}$ the product $\prod_{j \in \mathrm{fix}(\sigma)} \deg_{in}(v_j)$ gives six, the total number of spregs contained in the graph. The results for the remaining elements of S_3 are listed in Table 9.2 and all are covered by Corollary 9.5, as all non-identity elements of S_3 are single cycles.

9.2.3 Counting spregs in general

Here we generalise the results from Section 9.2.1 to permutations that are the products of arbitrarily many cycles.

Lemma 9.6 (Counting spregs containing cycles)**.**
Suppose $\sigma \in S_{n-1}$ is the product of $k > 0$ disjoint cycles

$$\sigma = (i_{1,1}, \ldots, i_{1,\ell_1}) \ldots (i_{k,1}, \ldots, i_{k,\ell_k}),$$

where ℓ_j is the length of the j-th cycle. Associate the directed cycle C_j defined by the vertex sequence $(v_{i_{j,1}}, \ldots, v_{i_{j,\ell_j}}, v_{i_{j,1}})$ with the j-th cycle in the permutation and define

$$C_\sigma = \bigcup_{j=1}^{k} C_j.$$

Then the term in $\det(\mathcal{L})$ corresponding to σ satisfies

$$\mathrm{sgn}(\sigma) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} = \begin{cases} (-1)^k \prod_{j \in \mathrm{fix}(\sigma)} \deg_{in}(v_j) & \text{if } C_\sigma \subseteq G(V,E) \text{ and } \mathrm{fix}(\sigma) \neq \emptyset \\ (-1)^k & \text{if } C_\sigma \subseteq G(V,E) \text{ and } \mathrm{fix}(\sigma) = \emptyset \\ 0 & \text{if } C_\sigma \not\subseteq G(V,E) \end{cases}$$

Further,

$$\left|\prod_{j=1}^{n-1}\mathcal{L}_{j\sigma(j)}\right| \;=\; \left|\left\{\textit{spregs containing } C_\sigma = \textstyle\bigcup_{j=1}^{k} C_j\right\}\right|. \tag{9.5}$$

The proof of this result requires reasoning much like that used in Section 9.2.1 and so is left to the reader.

9.3 Proof of Tutte's theorem

Throughout this section I will continue to write $\mathcal{L}$ in place of $\hat{L}_n$ to avoid a confusing welter of subscripts.

Proof. As we argued at the beginning of Section 9.2, it is sufficient to prove that $\det(\hat{L}_n) = \det(\mathcal{L})$ is the number of spanning arborescences rooted at v_n. We'll do this with the Principle of Inclusion/Exclusion and so, to begin, we need to specify the universal set U and the subsets X_j. Begin by considering the set $\mathcal{C}$ of all possible directed cycles involving the vertices $v_1 \ldots v_{n-1}$. It's clearly a finite set and so we can declare that it has M elements and imagine that we've chosen some (arbitrary) numbering scheme so that we can list the set of cycles as

$$\mathcal{C} = \{C_1, \ldots, C_M\}.$$

We'll then choose the sets U and X_j as follows:

- U is the set of all spregs with distinguished vertex v_n. That is, U is the set of subgraphs of $G(V, E)$ in which

$$\deg_{in}(v_n) = 0 \qquad \text{and} \qquad \deg_{in}(v_j) = 1 \text{ for } 1 \le j \le (n-1).$$

- $X_j \subseteq U$ is the subset of U consisting of spregs containing the cycle C_j. This subset may, of course, be empty.

Proposition 9.3—the one about characterising spregs—tells us that a spreg that has distinguished vertex v_n is either a spanning arboresence rooted at v_n or a graph that contains one or more disjoint cycles. This means that

$$N_n \;=\; |\{\text{spanning arborescences rooted at } v_n\}| \;=\; |U| - \left|\bigcup_{j=1}^{M} X_j\right|.$$

and the Principle of Inclusion/Exclusion then says

$$\begin{aligned} N_n &= |U| - \left(\sum_{I\subseteq\{1,\ldots,M\},\, I\neq\emptyset} (-1)^{|I|-1}\left|\bigcap_{j\in I} X_j\right|\right) \\ &= |U| + \sum_{I\subseteq\{1,\ldots,M\},\, I\neq\emptyset} (-1)^{|I|}\left|\bigcap_{j\in I} X_j\right| \end{aligned} \tag{9.6}$$

As we know that spregs contain only disjoint cycles, we can say

$$|X_j \cap X_k| = 0 \text{ unless } C_j \cap C_k = \emptyset$$

and so can eliminate many of the terms in the sum over intersections in Eqn. (9.6), rewriting it as a sum over collections of disjoint cycles:

$$N_n = |U| + \sum_{\substack{I \subseteq \{1,\ldots,M\},\, I \neq \emptyset \\ C_j \cap C_k = \emptyset \ \forall j \neq k \in I}} (-1)^{|I|} \left| \bigcap_{j \in I} X_j \right|. \tag{9.7}$$

Then we can use the lemma from the previous section—Lemma 9.6, which relates non-identity permutations to numbers of spregs containing cycles—to rewrite Eqn. (9.7) in terms of permutations. First note that Eqn. (9.5) allows us to write

$$\left| \bigcap_{j \in I} X_j \right| = \left| \left\{ \text{spregs containing } \textstyle\bigcup_{j \in I} C_j \right\} \right| = \left| \prod_{k=1}^{n-1} \mathcal{L}_{k\sigma_I(k)} \right|.$$

Here $\sigma_I \in S_{n-1}$ is the permutation

$$\sigma_I = \prod_{j \in I} \sigma_{C_j}$$

whose cycle representation is the product of the permutations corresponding to the directed cycles C_j for $j \in I$. In the product above σ_{C_j} is the cycle permutation corresponding to the directed cycle C_j. The correspondence here somes from the bijection between permutations and unions of directed cycles that we discussed in Section 8.2.

Now, again using Lemma 9.6, we have

$$N_n = |U| + \sum_{\substack{I \subseteq \{1,\ldots,M\},\, I \neq \emptyset \\ C_j \cap C_k = \emptyset \ \forall j \neq k \in I}} (-1)^{|I|} \left| \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma_I(j)} \right|$$

$$N_n = |U| + \sum_{\substack{I \subseteq \{1,\ldots,M\},\, I \neq \emptyset \\ C_j \cap C_k = \emptyset \ \forall j \neq k \in I}} \operatorname{sgn}(\sigma_I) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma_I(j)} \tag{9.8}$$

As the sum in Eqn. (9.8) ranges over *all* collections of disjoint cycles, the permutations σ_I range over all non-identity permutations in S_{n-1} and so we have

$$N_n = |U| + \sum_{\sigma \neq \mathrm{id}} \operatorname{sgn}(\sigma) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)}. \tag{9.9}$$

Finally, from Eqn. (9.3) we know that

$$|U| = |\{\text{spregs containing all } v \in V \text{ with distinguished vertex } v_n\}| = \prod_{j=1}^{n-1} \deg_{in}(v_j)$$

which is the term in $\det(\mathcal{L})$ corresponding to the identity permutation. Combining this observation with Eqn. (9.9) gives us

$$N_n \;=\; \sum_{\sigma \in S_{n-1}} \operatorname{sgn}(\sigma) \prod_{j=1}^{n-1} \mathcal{L}_{j\sigma(j)} \;=\; \det(\mathcal{L}),$$

which is the result we sought.

$\square$

Part III

Eulerian and Hamiltonian Graphs

Lecture 10

Eulerian Multigraphs

This section of the notes revisits the Königsberg Bridge Problem and generalises it to explore *Eulerian multigraphs:* those that contain a closed walk that traverses every edge exactly once.

Reading:
The material in today's lecture comes from Section 1.3 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, (available online via SpringerLink),

though his proof is somewhat more terse.

In Lecture 1 we used a proof by contradiction to demonstrate that there is no solution to the Königsberg Bridge Problem, which is illustrated in Figure 10.1. That is, it's not possible to find a walk that (a) crosses each of the city's seven bridges exactly once and (b) starts and finishes in the same place. Today we'll generalise the problem, then find a number of equivalent conditions that tell us when the corresponding closed walk exists.

First, recall that a *multigraph* $G(V, E)$ has the same definition as a graph, except that we allow parallel edges. That is, we allow pairs of vertices (u, v) to appear more than once in E. Because of this, people sometimes speak of the *edge list* of a multigraph, as opposed to the *edge set*.

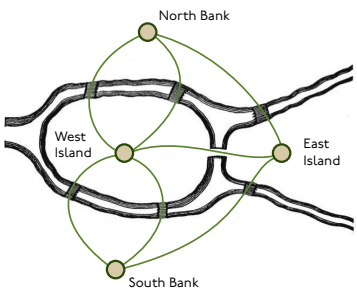


Figure 10.1: *We proved in the first lecture of the term that it is impossible to find a closed walk that traverses every edge in the graph above exactly once.*

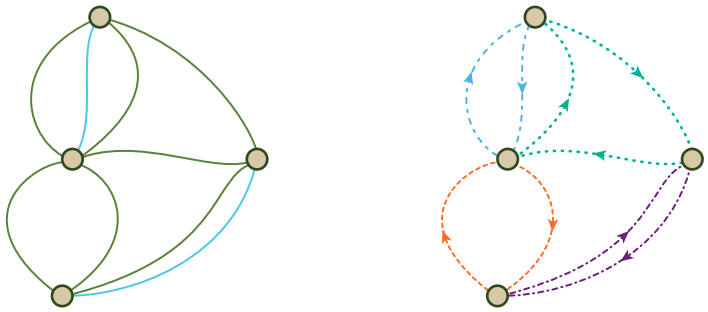


Figure 10.2: *The panel at left shows a graph produced by adding two edges (shown in blue) to the graph from the Königsberg Bridge Problem. These extra edges make the graph Eulerian and the panel at right illustrated the associated partition of the edge set into cycles*

The main theorem we'll prove today relies on the following definitions:

Definition 10.1. *An* ***Eulerian trail*** *in a multigraph $G(V, E)$ is a trail that includes each of the graph's edges exactly once.*

Definition 10.2. *An* ***Eulerian tour*** *in a multigraph $G(V, E)$ is an Eulerian trail that starts and finishes at the same vertex. Equivalently, it is a closed trail that traverses each of the graph's edges exactly once.*

Definition 10.3. *A multigraph that contains an Eulerian tour is said to be an* ***Eulerian multigraph****.*

Armed with these, it's then easy to formulate the following characterisation of Eulerian multigraphs:

Theorem 10.4 (Jungnickel's Theorem 1.3.1)**.** *Let G be a connected multigraph. Then the following statements are equivalent:*

(1) G is Eulerian.

(2) Each vertex of G has even degree.

(3) The edge set of G can be partitioned into cycles.

The last of these characterisations may be new to you: it means that it is possible to arrange the edges of G into a collection of disjoint cycles. Figure 10.2 shows an example of such a partition for a graph derived from the Königsberg Bridge multigraph by adding two extra edges, shown in blue at left. Adding these edges makes the graph Eulerian, and a decomposition of the edge set into cycles appears at right. Note that undirected multigraphs can contain cycles of length two that consist of a pair of parallel edges.

The proof of the theorem is simpler if one has the following lemma, whose proof I'll defer until after that of the main result. Note that the lemma, unlike the theorem, does not require the multigraph to be connected.

Lemma 10.5 (Vertices of even degree and cycles)**.** *If $G(V, E)$ is a multigraph with a nonempty edge set $E \neq \emptyset$ and the property that* $\deg(v)$ *is an even number for all $v \in V$, then G contains a cycle.*

Proof of Theorem 10.4. The theorem says these statements are all "equivalent", which encompasses a total of six implications[1] but we don't need to prove all of them: it's sufficient to prove, say, that (1) $\implies$ (2), (2) $\implies$ (3) and (3) $\implies$ (1). That is, it's sufficient to make a directed graph whose vertices are the statements and whose edges indicate implications. If this graph is strongly connected, so that one can get from any statement to any other by following a chain of implications, then the result is proven.

(1) $\implies$ (2):

Proof. We know G is Eulerian, so it has a closed trail that includes each edge exactly once. Imagine that this trail is specified by the following sequence of vertices

$$v_0, \ldots, v_m = v_0 \tag{10.1}$$

where $|E| = m$ and the v_j are the vertices encountered along the trail, so that some of them may appear more than once. In particular, $v_0 = v_m$ because the trail starts and finishes at the same vertex. As G is a connected multigraph, every vertex appears somewhere in the sequence (if not, the absent vertices would have degree zero and not be connected to any of the others).

Consider first some vertex $u \neq v_0$. It must appear one or more times in the sequence above and, each time, it appears in a pair of successive edges: if $u = v_j$ with $0 < j < m$, then these edges are (v_{j-1}, v_j) and (v_j, v_{j+1}). This means that $\deg(u)$ is a sum of 2's, with one term in the sum for each appearance of u in the sequence (10.1). A similar argument applies to v_0, save that the edge that forms a pair with (v_0, v_1) is $(v_{m-1}, v_m = v_0)$. □

(2) $\implies$ (3): The theorem requires this implication to hold for connected multigraphs, but this particular result is more general and applies to *any* multigraph in which all vertices have even degree. We'll prove this stronger version by induction on the number of edges. That is, we'll prove:

Proposition. *If $G(V, E)$ is a multigraph (whether connected or not) in which* $\deg(v)$ *is an even number for all vertices $v \in V$, then the edge set E can be partitioned into cycles.*

Proof. The base case is a multigraph with $|E| = 0$. Such a graph consists of one or more isolated vertices and, as the graph has no edges, $\deg(v) = 0$ (an even number) for all $v \in V$ and the (empty) edge set can clearly be partitioned into a union of zero cycles.

Now suppose the result is true for every multigraph $G(V, E)$ with $|E| \leq m_0$ edges whose vertices all have even degree. Consider such a multigraph with $|E| = m_0 + 1$: we need to demonstrate that the edge set of such a graph can be partitioned into

[1] (1) $\implies$ (2), (2) $\implies$ (1), (1) $\implies$ (3)...

cycles. We can use Lemma 10.5 to establish that we can find at least one cycle C contained in G. And then we can form a new graph $G'(V', E') = G \backslash C$ formed by removing C from G. This bit of graph surgery either leaves the degree of a vertex unchanged (if the vertex isn't part of C) or decreases it by two, but either way, all vertices in G' have even degree because the corresponding vertices in G do.

The cycle C will contain at least one edge (and, unless we permit self-loops, two or more) and so G' will have at most m_0 edges and so the inductive hypothesis will apply to it. This means that we can partition $E' = E \backslash C$ into cycles. But then we can add C to the partition of E' and so get a partition into cycles for E, completing the inductive step and so proving our result. □

(3) $\implies$ (1): Here we need to establish that if the edge set of a connected multigraph $G(V, E)$ consists of a union of cycles, then G contains an Eulerian tour. This result is trivial unless the partition of E involves at least two cycles, so we'll restrict attention to that case from now on.

The key observation is that we can always find two cycles that we can merge to produce a single, longer closed trail that includes all the edges from the two cycles. To see why, note that there must be a pair of cycles that share a vertex (if there weren't, all the cycles would all lie in distinct connected components, contradicting the connectedness of G). Suppose that the shared vertex is $v_\star$ and that the cycles are C_1 and C_2 given by the vertex sequences

$$C_1 = \{v_\star = v_0, v_1, \ldots, v_{\ell_1} = v_\star\} \qquad \text{and} \qquad C_2 = \{v_\star = u_0, u_1, \ldots, u_{\ell_2} = v_\star\}.$$

We can combine them, as illustrated in Figure 10.3 to make a closed trail given by the vertex sequence

$$\{v_\star = v_0, v_1, \ldots, v_{\ell_1} = v_\star = u_0, u_1, \ldots, u_{\ell_2} = v_\star\}.$$

Scrupulous readers may wish to use this observation as the basis of a proof by induction (on the number of elements in the partition of E) of the somewhat stronger result:

Proposition 10.6. *If $G(V, E)$ is a strongly-connected graph whose edge set can be partitioned as a union of disjoint, closed trails, then G is Eulerian.*

Then, as a cycle is a special case of a closed trail, we get the desired implication as an immediate corollary. □

I'd like to conclude by giving an algorithmic proof of Lemma 10.5. The idea is to choose some initial vertex u_0 and then construct a trail in the graph by following one of u_0's edges, then one of the edges of u_0's successor in the trail ... and so on until we revisit some vertex and thus discover a cycle. Provided that we can do as I say—always move on through the graph without ever tracing over some edge twice—this approach is bound to work because there are only finitely many vertices. The proof that follows formalises this approach by spelling out an explicit algortihm.

Proof of Lemma 10.5. Consider the following algorithmic process, which finds a cycle in a multigraph $G(V, E)$ for which $E \neq \emptyset$ and $\deg(v)$ is even for all $v \in V$.

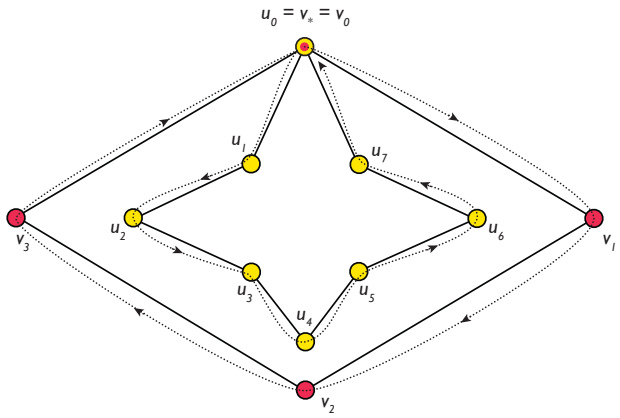


Figure 10.3: *The key step in the proof of the implication* (3) $\Longrightarrow$ (1) *in the proof of Theorem 10.4. The cycles* $C_1 = (v_\star = v_0, v_1, v_2, v_3, v_0)$, *whose vertices are shown in red, and* $C_2 = (v_\star = u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_0)$, *whose vertices are shown in yellow, may be merged to create the closed trail* $(v_\star = v_0, v_1, v_2, v_3, v_0 = v_\star = u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_0)$ *indicated by the dotted line.*

Algorithm 10.7 (Finding a cycle)**.**
Given a multigraph $G(V, E)$ *in which* $|E| > 0$ *and all vertices have even degree, construct a trail* T *given by a sequence of edges*

$$T = \{(u_0, u_1),\ (u_1, u_2),\ \ldots,\ (u_{\ell-1}, u_\ell)\}$$

that includes a cycle.

(1) Number the vertices, so that $V = \{v_1, \ldots, v_n\}$. *This is for the sake of concreteness: later in the algorithm, when we need to choose one of a set of vertices that have a particular property, we can choose the lowest-numbered one.*

(2) Initialize some things

- *Set a counter* $j \leftarrow 0$.
- *Choose the first vertex in the trail,* u_0, *to be the lowest-numbered vertex that has* $\deg(v_k) > 0$. *Such vertices exist, as we know* $|E| > 0$.
- *Initialise a list* A *(for "available") of edges that we have not yet included in* T. *At the outset we set* $A \leftarrow E$ *as we haven't used any edges yet.*

(3) Find the edge $(u_j, w) \in A$ *where* w *is the lowest-numbered neighbour of* u_j *whose edge we haven't yet used. The key to the algorithm's success is that this step is always possible. We chose* u_0 *with* $\deg(u_0) > 0$, *so this step is possible when* $j = 0$. *And when* $j > 0$, *the evenness of* $\deg(u_j)$ *means that if the trail we are constructing can arrive at* u_j, *then it must also be able to depart. The growing trail* T *either comes to a stop at* u_j *(see below) or uses a pair of the*

vertex's edges—one to arrive and another to depart—and so leaves an even number of unused edges behind.

We can thus always extend the trail T by one edge, modifying the list of unused edges A accordingly.

- $T \leftarrow T \cup \{(u_j, w)\}$
- $A \leftarrow A \backslash \{(u_j, w)\}$ (We've used (one copy of) the edge (u_j, w)).

(4) Are we finished? Does w already appear in the trail?

- *If yes, stop. The trail includes a cycle that starts and finishes at w.*
- *If no, set $u_{j+1} \leftarrow w$, then set $j \leftarrow j + 1$ and go to Step 3.*

The only way this process can stop is by revisiting a vertex and it must do this within $|V| = n$ steps. And once we've revisited a vertex, we've found a cycle and so are finished. $\square$

Lecture 11

Hamiltonian graphs and the Bondy-Chvátal Theorem

This lecture introduces the notion of a Hamiltonian graph and proves a lovely theorem due to J. Adrian Bondy and Vašek Chvátal that says—in essence—that if a graph has lots of edges, then it must be Hamiltonian.

Reading:
The material in today's lecture comes from Section 1.4 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, (available online via SpringerLink),

and is essentially an expanded version of the proof of Jungnickel's Theorem 1.4.1.

11.1 Hamiltonian graphs

In the last lecture we characterised Eulerian graphs, which are those that have a closed trail that includes every *edge* exactly once. It's then natural to wonder about graphs that have closed trails that include every *vertex* exactly once. Somewhat surprisingly, these turn out to be much, much harder to characterise. To begin with, let's make some definitions that parallel those for Eulerian graphs:

Definition 11.1. *A* ***Hamiltonian path*** *in a graph $G(V, E)$ is a path that includes all of the graph's vertices.*

Definition 11.2. *A* ***Hamiltonian tour*** *or* ***Hamiltonian cycle*** *in a graph $G(V, E)$ is a cycle that includes every vertex.*

Definition 11.3. *A graph that contains a Hamiltonian tour is said to be a* ***Hamiltonian graph****. Note that this implies that Hamiltonian graphs have $|V| \geq 3$, as otherwise they would be unable to contain a cycle.*

Generally speaking, it's difficult to decide whether a graph is Hamiltonian—there are no known efficient algorithms. There are, however, some special cases that are

easy: the cycle graphs C_n consist of nothing except one big Hamiltonian tour, and the complete graphs K_n with $n \geq 3$ obviously contain the Hamiltonian cycle

$$(v_1, v_2, \ldots, v_n, v_1)$$

obtained by numbering the vertices and visiting them in order. We'll spend most of the lecture proving results that say, more-or-less, that a graph with a lot of edges (where the point of the theorem is to make the sense of "a lot" precise) is Hamiltonian. Two of the simplest results of this kind are:

Theorem 11.4 (Dirac[1], 1952)**.** *Let G be a graph with $n \geq 3$ vertices. If each vertex of G has $\deg(v) \geq n/2$, then G is Hamiltonian.*

Theorem 11.5 (Ore, 1960)**.** *Let G be a graph with $n \geq 3$ vertices. If*

$$\deg(u) + \deg(v) \geq n$$

for every pair of non-adjacent vertices u and v, then G is Hamiltonian.

Dirac's theorem is a corollary of Ore's, but we will not prove either of these theorems directly. Instead, we'll obtain both as corollaries of a more general result, the Bondy-Chvátal Theorem. Before we can even formulate this mighty result, we need a somewhat involved new definition: the *closure* of a graph.

11.2 The closure a graph

Suppose G is a graph on n vertices. Then the ***closure*** of G, written $[G]$, is constructed by adding edges that connect pairs of non-adjacent vertices u and v for which

$$\deg(u) + \deg(v) \geq n. \tag{11.1}$$

One continues recursively, adding new edges according to (11.1) until all non-adjacent pairs u, v satisfy

$$\deg(u) + \deg(v) < n.$$

The graphs G and $[G]$ have the same vertex set—I'll call it V—but the edge set of $[G]$ may contain extra edges. In the next section I'll give an explicit algorithm that constructs the closure.

[1] This Dirac, Gabriel Andrew Dirac, was the adopted son of the Nobel prize winning theoretical physicist Paul A. M. Dirac, and the nephew of another Nobel prize winner, the physicist and mathematician Eugene Wigner. Wigner's sister Margit was visiting her brother in Princeton when she met Paul Dirac.

11.2.1 An algorithm to construct $[G]$

The algorithm below constructs a finite sequence of graphs

$$G = G_1(V, E_1),\ G_2(V, E_2),\ \ldots,\ G_K(V, E_K) = [G] \tag{11.2}$$

that all have the same vertex set V, but different edge sets

$$E = E_1,\ E_2,\ \ldots,\ E_K. \tag{11.3}$$

These edge sets form an increasing sequence in the sense that that $E_j \subset E_{j+1}$. In fact, E_{j+1} is produced by adding a single edge to E_j.

Algorithm 11.6 (Graph Closure)**.**
Given a graph $G(V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$, find $[G]$.

(1) Set an index j to one: $j \leftarrow 1$,
Also set E_1 to be the edge set of the original graph,
$E_1 \leftarrow E$.

(2) Given E_j, construct E_{j+1}, which contains, at most, one more edge than E_j. Begin by setting $E_{j+1} \leftarrow E_j$, so that E_{j+1} automatically includes every edge in E_j. Now work through every possible edge in the graph. For each one—let's call it $e = (v_r, v_s)$—there are three possibilities:

(i) the edge e is already present in E_j.

(ii) The edge $e = (v_r, v_s)$ is not in E_j, but the degrees of the vertices v_r and v_s are low in the sense that

$$\deg_{G_j}(v_r) + \deg_{G_j}(v_s) < n,$$

where the subscript G_j is meant to show that the degree is being calculated in the graph G_j, whose vertex set is V and whose edge set is E_j. In this case we do not *include e in E_{j+1}.*

(iii) the edge $e = (v_r, v_s)$ is not in E_j, but the degrees of the vertices v_r and v_s are high in the sense that

$$\deg_{G_j}(v_r) + \deg_{G_j}(v_s) \geq n. \tag{11.4}$$

Such an edge should be part of the closure, so we set

$$E_{j+1} \leftarrow E_j \cup \{e\}.$$

and then jump straight to step 3 below.

(3) Decide whether to stop: ask whether we added an edge during step 2.

- *If not, then stop: the closure $[G]$ has vertex set V and edge set E_j.*
- *Otherwise set $j \leftarrow j + 1$ and go back to step (2) to try to add another edge.*

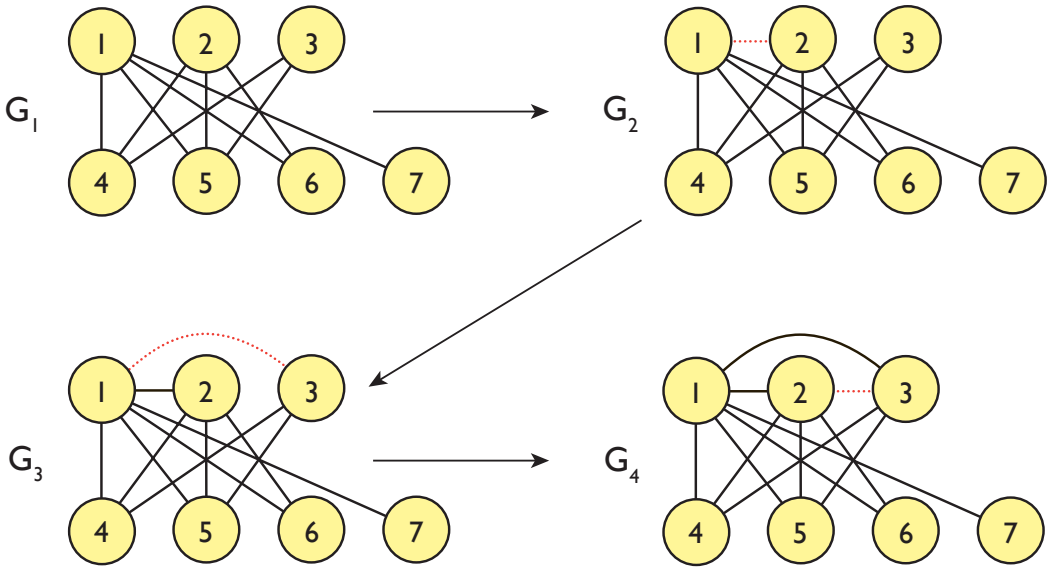


Figure 11.1: *The results of applying Algorithm 11.6 to the seven-vertex graph G_1. Each round of the construction (each pass through step 2 of the algorithm) adds a single new edge—shown with red, dotted curves—to the graph.*

11.2.2 An example

Figure 11.1 shows the result of applying Algorithm 11.6 to a graph with 7 vertices. The details of the process are discussed below.

Making G_2 from G_1
When constructing E_2 from E_1, notice that the vertex with highest degree, v_1, has $\deg_{G_1}(v_1) = 4$ and all the other vertices have lower degree. Thus, in step 2 of the algorithm we need only think about edges connecting v_1 to vertices of degree three. There are three such vertices—v_2, v_4 and v_5—but two of them are already adjacent to v_1 in G_1, so the only new edge we need to add at this stage is $e = (v_1, v_2)$.

Making G_3 from G_2
Now v_1 has $\deg_{G_2}(v_1) = 5$, so the closure condition (11.4) says that we should connect v_1 to any vertex whose degree is two or more, which requires us to add the edge (v_1, v_3).

Making G_4 from G_3
Now v_1 has degree $\deg_{G_3}(v_1) = 6$, so it is already connected to every other vertex in the graph and cannot receive any new edges. Vertex v_2 has $\deg_{G_3}(v_2) = 4$ and so should be connected to any vertex v_j with $\deg_{G_3}(v_j) \geq 3$. This means we need to add the edge $e = (v_2, v_3)$.

Conclusion: $[G] = G_4$
Careful study of G_4 shows that the rule (11.4) will not add any further edges, so the closure of the original graph is G_4.

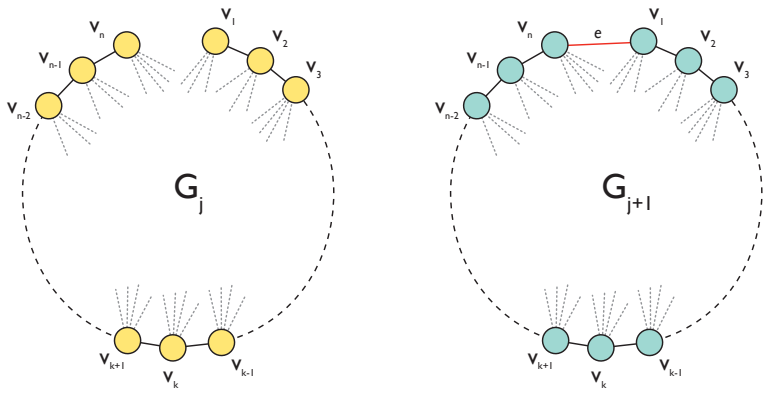


Figure 11.2: *The setup for the proof of the Bondy-Chvátal Theorem: adding the edge $e = (v_1, v_n)$ to G_j creates the Hamiltonian cycle $(v_1, \ldots, v_n, v_1)$ that's found in G_{j+1}. The dashed lines spraying off into the middles of the diagrams are meant to indicate that the vertices may have other edges besides those shown in black.*

11.3 The Bondy-Chvátal Theorem

The point of defining the closure is that it enables us to state the following lovely result:

Theorem 11.7 (Bondy and Chvátal, 1976)**.** *A graph G is Hamiltonian if and only if its closure $[G]$ is Hamiltonian.*

Before we prove this, notice that Dirac's and Ore's theorems are easy corollaries, for when $\deg(v) \geq n/2$ for all vertices (Dirac's condition) or when $\deg(u) + deg(v) \geq n$ for all non-adjacent pairs (Ore's condition), we have $[G] = K_n$, and, as we've seen, K_n is trivially Hamiltonian.

Proof. As the theorem is an if-and-only-if statement, we need to establish two things: (1) if G is Hamiltonian then $[G]$ is and (2) if $[G]$ is Hamiltonian then G is too. The first of these is easy in that the closure construction only *adds* edges to the graph, so in the sequence of edge sets (11.3) G has edge set E_1 and $[G]$ has edge set E_K with $K \geq 1$ and $E_K \supseteq E_1$. This means that any edges appearing in a Hamiltonian tour in G are automatically present in $[G]$ too, so if G is Hamiltonian, $[G]$ is also.

The second implication is harder and depends on an ingenious proof by contradiction. First notice that—by an argument similar to the one above—if some graph $G_{j_\star}$ in the sequence (11.2) is Hamiltonian, then so are all the other G_j with $j \geq j_\star$. This means that if the sequence is to begin with a non-Hamiltonian graph $G = G_1$ and finish with a Hamiltonian one $G_K = [G]$ there must be a single point at which the nature of the graphs in the sequence changes. That is, there must be some $j \geq 1$ such that G_j isn't Hamiltonian, but G_{j+1} is, even though G_{j+1} differs from G_j by only a single edge. This situation is illustrated in Figure 11.2, where I have numbered the vertices $v_1 \ldots v_n$ according to their position in the Hamiltonian cycle in G_{j+1} and arranged things so that the single edge whose addition creates the cycle is $e = (v_1, v_n)$.

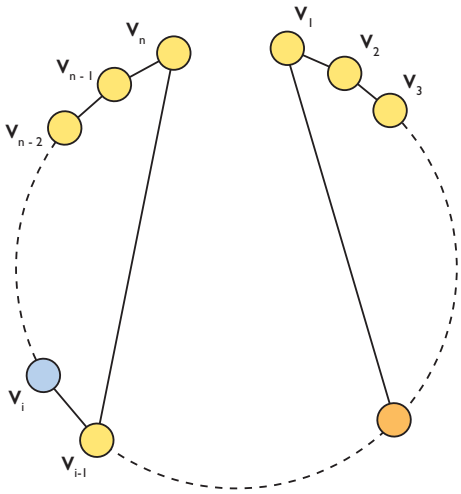


Figure 11.3: *Here the blue vertex, v_i, is in X because it is connected* indirectly *to v_n, through its predecessor v_{i-1}, while the orange vertex is in Y because it is connected* directly *to v_1.*

Let's now focus on G_j and define two interesting sets of vertices

$$X = \{v_i \mid (v_{i-1}, v_n) \in E_j \text{ and } 2 < i < n\}$$

and

$$Y = \{v_i \mid (v_1, v_i) \in E_j \text{ and } 2 < i < n\}.$$

The first set, X, consists of those vertices whose predecessor in the cycle has a direct connection to v_n, while the second set, Y, consists of vertices that have a direct connection to v_1: both sets are illustrated in Figure 11.3.

Notice that X and Y are defined to be subsets of $\{v_3, \ldots, v_{n-1}\}$, so they exclude v_1, v_2 and v_n. Thus X has $\deg_{G_j}(v_n) - 1$ members as it includes one element for each the neighbours of v_n except for v_{n-1}, while $|Y| = \deg_{G_j}(v_1) - 1$ as Y includes all neighbours of v_1 other than v_2. So then

$$\begin{aligned} |X| + |Y| &= \Big(\deg_{G_j}(v_n) - 1\Big) + \Big(\deg_{G_j}(v_1) - 1\Big) \\ &= \deg_{G_j}(v_n) + \deg_{G_j}(v_1) - 2 \\ &\geq n - 2 \end{aligned}$$

where the inequality follows because we know

$$\deg_{G_j}(v_n) + \deg_{G_j}(v_1) \geq n$$

as the closure construction is going to add the edge $e = (v_1, v_n)$ when passing from G_j to G_{j+1}. But then, both X and Y are drawn from the set of vertices $\{v_i \mid 2 < i < n\}$ which has only $n - 3$ members and so, by the pigeonhole principle, there must be some vertex v_k that is a member of both X and Y.

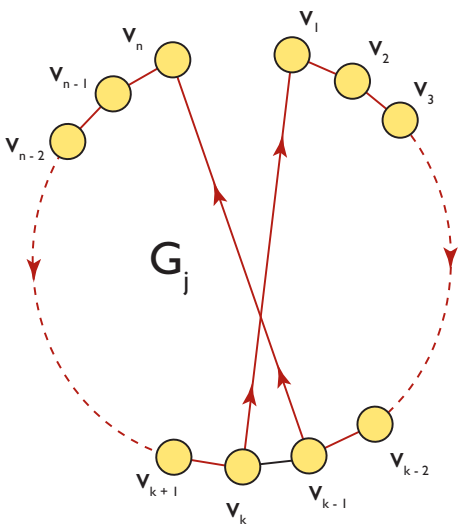


Figure 11.4: *The vertex v_k is a member of $X \cap Y$, which implies that there is, as shown above, a Hamiltonian cycle in G_j.*

The existence of such a v_k implies the presence of a Hamiltonian tour in G_j. As is illustrated in Figure 11.4, this tour:

- runs from v_1 to v_{k-1}, in the same order as the tour found in G_{j+1},
- then jumps from v_{k-1} to v_n: this is possible becase $v_k \in X$.
- The tour then continues, passing from v_n to v_{n-1} and on to v_k, visiting vertices in the opposite order from the tour in G_{j+1}
- and concludes with a jump from v_k to v_1, which is possible because $v_k \in Y$.

The existence of this tour contradicts our initial assumption that G_j is not Hamiltonian, but G_{j+1} is. This means no such G_j can exist: the sequence of graphs in the closure construction can never switch from non-Hamiltonian to Hamiltonian and so if $[G]$ is Hamiltonian, then G must be too. $\square$

11.4 Afterword

Students sometimes have trouble remembering the difference between Eulerian and Hamiltonian graphs and I'm not unsympathetic: after all, both are named after very famous, long-dead European mathematicians. One way out of this difficulty is to learn more about the two men. Leonhard Euler, who was born in Switzerland, lived longer ago (1707–1783) and was tremendously prolific, writing many hundreds of papers that made fundamental contributions to essentially all of 18th century mathematics. He also lived in a very alien scientific world in that he relied on royal patronage, first from the Russian emperor Peter the Great and then, later, from Frederick the Great of Prussia and then finally, toward the end of his life, from Catherine the Great of Russia.

By contrast William Rowan Hamilton, who was Irish, lived much more recently (1805–1865). He also made fundamental contributions across the whole of mathematics—the distinction between pure and applied maths didn't really exist then—but he inhabited a much more recognisable scientific community, first working as a Professor of Astronomy at Trinity College in Dublin and then, for the rest of his career, as the directory of Dunsink Observatory, just outside the city.

Alternatively, one can remember the distinction between Eulerian and Hamiltonian tours by noting that everything about Eulerian graphs starts with 'E': *Eulerian* tours go through every *edge* in a graph and are *easy* to find. On the other hand, *Hamiltonian* tours go through every vertex and are *hard* to find.

Part IV

Distance in Graphs and Scheduling

Lecture 12

Distance in Graphs

This lecture introduces the notion of a *weighted graph* and explains how some choices of weights permit us to define a notion of distance in a graph.

Reading:
The material in this lecture comes from Chapter 3 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, which is (available online via SpringerLink.

12.1 Adding weights to edges

The ideas and applications that will occupy us for the next few lectures involve both directed and undirected graphs and will include one of the most important applications in the course, which involves the scheduling of large complex projects. To begin with, we introduce the notion of *edge weights*.

Definition 12.1. *Given a graph $G(V, E)$, which may be either directed or undirected, we can associate* ***edge weights*** *with G by specifying a function $w : E \to \mathbb{R}$. We will write $G(V, E, w)$ to denote the graph $G(V, E)$ with edge weights given by w and we will call such a graph a* ***weighted graph****.*

We will write $w(a, b)$ to indicate the weight of the edge $e = (a, b)$ and if $G(V, E, w)$ is an undirected weighted graph we will require $w(a, b) = w(b, a)$ for all $(a, b) \in E$.

Note that Definition 12.1 allows the weights to be negative or zero. That's because, as we'll see soon, the weights can represent many things. If the vertices represent places, then we could define a weight function w so that, for an edge $e = (a, b) \in E$, the weight $w(e)$ is:

- the distance from a to b;
- the time it takes to travel from a to b, in which case it may happen that $w(a, b) \neq w(b, a)$;

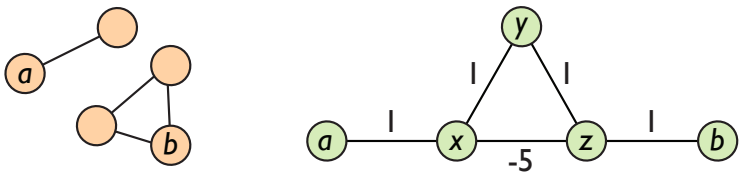


Figure 12.1: *In the graph at left there are no walks from a to b and so, by convention, we define $d(a,b) = \infty$. The graph at right, which has edge weights as indicated, illustrates a more serious problem. The cycle specified by the vertex sequence (x, y, z, x) has negative weight and so there is no minimal-weight path from a to b and hence no well-defined distance $d(a,b)$.*

- the profit made when we send a shipping container from a to b. This could easily be negative if we had to bring an empty container back from someplace we'd sent a shipment.

In any case, once we've defined weights for edges, it's natural to define the weight of a walk as follows.

Definition 12.2. *Given a weighted graph $G(V, E, w)$ and a walk from a to b defined by the vertex sequence*

$$a = v_0, \ldots, v_\ell = b,$$

so that the its edges are $e_j = (v_{j-1}, v_j)$, then the ***weight of the walk*** *is*

$$\sum_{j=1}^{\ell} w(e_j).$$

12.2 A notion of distance

Given two vertices a and b in a weighted graph $G(V, E, w)$, we might try to define a distance $d(a,b)$ from a to b as

$$d(a,b) = \min \{w(\omega) \,|\, \omega \text{ is a walk from } a \text{ to } b\},$$

but two issues, both of which are illustrated in Figure 12.1 present themselves immediately:

(1) What if there aren't any walks from a to b? In this case, by convention, we define $d(a,b) = \infty$.

(2) What if some cycle in G has negative weight? As we will see below, this leads to insurmountable problems and so we'll just have to exclude this possibility.

The problem with cycles of negative weight is illustrated at the right in Figure 12.1. The graph has $V = \{a, x, y, z, b\}$ and edge weights

$$w(a,x) = w(x,y) = w(y,z) = w(z,b) = 1 \qquad \text{and} \qquad w(x,z) = -5.$$

The cycle specified by the vertex sequence (z, x, y, z) thus has weight

$$w(z, x) + w(x, y) + w(y, z) = -5 + 1 + 1 = -3$$

and one can see that this presents a problem for our definition of $d(a, b)$ by considering the sequence of walks:

Walk	*Weight*	
(a, x, y, z, b)	$1 + 1 + 1 + 1 \;=$	4
(a, x, y, z, x, y, z, b)	$4 + (-5 + 1 + 1) \;=$	1
$(a, x, y, z, x, y, z, x, y, z, b)$	$4 - 2 \times 3 \;=$	-2
$\vdots$		
$(a, x, y, z, \underbrace{x, y, z, \ldots, x, y, z}_{k \text{ times around the cycle}}, b)$	$4 - k \times 3 \;=$	$4 - 3k.$

There *is* no walk of minimal weight from a to b: one can always find a walk of lower weight by tracing over the negative-weight cycle a few more times. We could escape this problem by defining $d(a, b)$ as the weight of a minimal-weight *path*[1], but instead we will exclude the problematic cases explicitly:

Definition 12.3. *Suppose $G(V, E, w)$ is a weighted graph that does not contain any cycles of negative weight. For vertices a and b we define the* ***distance function*** *$d : V \times V \to \mathbb{R}$ as follows:*

- *$d(a, a) = 0$ for all $a \in V$;*
- *$d(a, b) = \infty$ if there is no walk from a to b;*
- *$d(a, b)$ is the weight of a minimal-weight walk from a to b when such walks exist.*

A warning

The word "distance" in Definition 12.3 is potentially misleading in that it is perfectly possible to find weighted graphs in which $d(a, b) < 0$ for some (or even all) a and b. Further, it's possible that in a directed graph there may be vertices a and b such that $d(a, b) \neq d(b, a)$. If we want our distance function to have the all the properties that the word "distance" normally suggests, it's helpful to recall (or learn for the first time) the definition of a *metric on a set* X. It's a function $d : X \times X \to \mathbb{R}$ with the following properties:

Non-negativity $d(x, y) \geq 0 \; \forall x, y \in X$ and $d(x, y) = 0 \iff x = y$;

symmetry $d(x, y) = d(y, x) \; \forall x, y \in X$;

triangle inequality $d(x, y) + d(y, z) \geq d(x, z) \; \forall x, y, z \in X$.

If d is a metric on X we say that the pair (X, d) constitute a *metric space*. It's not hard to prove (see the Problem Sets) that if $G(V, E, w)$ is a weighted, undirected graph in which $w(e) > 0 \; \forall e \in E$, then the function $d : V \times V \to \mathbb{R}$ from Definition 12.3 is a metric on the vertex set V.

[1] A path cannot revisit a vertex and hence cannot trace over a cycle.

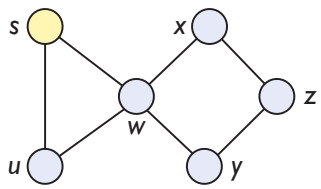


Figure 12.2: *A SSSP problem in which all the edge weights are 1 and the source vertex s is shown in yellow.*

12.3 Shortest path problems

Once one has a definition for distance in a weighted graph $G(V, E, w)$, two natural problems present themselves:

Single-Source Shortest Path (SSSP): Given a vertex s—the so-called *source vertex*—compute $d(s, v)\ \forall v \in V$.

All-Pairs Shortest Path: Compute $d(u, v)\ \forall u, v \in V$.

We will develop algorithms to solve the first of these, but not the second. Of course, if one has an algorithm for SSSP, one can also solve the second by applying the SSSP algorithm with each vertex as the source, though there are more efficient approaches as well.

12.3.1 Uniform weights & Breadth First Search

The simplest SSSP problems are those in undirected weighted graphs where all edges have the same weight, say, $w(e) = 1\ \forall e \in E$. In this case one can use an algorithm called *Breadth First Search (BFS)*, which is one of the fundamental tools of algorithmic graph theory. I'll present the algorithm twice, once informally, by way of an example, and then again in a sufficiently detailled way that one could, for example, implement it in MATLAB. As we're working on a single-source problem, it's convenient to define

$$d(v) \equiv d(s, v),$$

where s is the source vertex. Our goal is then to compute $d(v)$ for all vertices in the graph.

To illustrate the main ideas, we'll use BFS to compute $d(v)$ for all the vertices in the graph pictured in Figure 12.2:

- Set $d(s) = 0$.
- Set $d(v) = 1$ for all s's neighbours. That is, set $d(v) = 1$ for all vertices $v \in A_s = \{u, w\}$.
- Set $d(v) = 2$ for those vertices that (a) are adjacent to vertices t with $d(t) = 1$ and (b) have not yet had a values of $d(v)$ assigned.

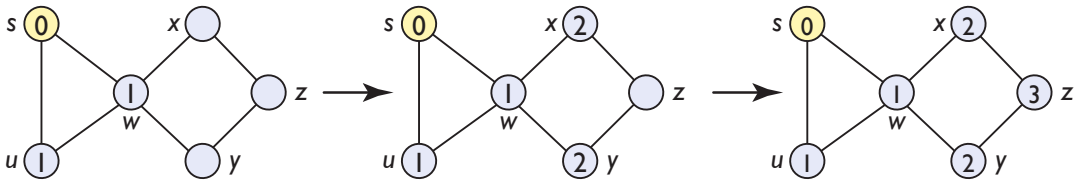


Figure 12.3: *The leftmost graph shows the result of the first two stages of the informal BFS algorithm: we set $d(s) = 0$ and and $d(v) = 1$ for all $v \in A_s$. In the second stage we set $d(v) = 2$ for neighbours of vertices t with $d(t) = 1$ $\cdots$ and so on.*

- Set $d(v) = 3$ for those vertices that (a) are adjacent to vertices t with $d(t) = 2$ and (b) have not yet had a values of $d(v)$ assigned.

This process is illustrated in Figure 12.3 and, for the current example, these four steps assign a value of $d(v)$ to every vertex. In Section 12.4 we will return to this algorithm and rewrite it in a more general way, but I'd like to conclude this section by discussing why this approach works.

12.3.2 Bellman's equations

BFS, and indeed all the shortest-path algorithms we'll study, work because of a characterisation of minimum-weight walks due to Richard Bellman. Suppose $G(V, E, w)$ is a weighted graph (either directed or undirected) on $|V| = n$ vertices. Specify a single-source shortest path problem by numbering the vertices so that the source vertex s comes first, $v_1 = s$, and assemble the edge weights into an $n \times n$ matrix w whose entries are given by

$$w_{k,j} = \begin{cases} w(v_k, v_j) & \text{if } (v_k, v_j) \in E \\ \infty & \text{otherwise} \end{cases}$$

Then we have the following theorem, which captures the idea that a minimal-weight path from v_1 to v_j consists of a minimal-weight path from v_1 to one of v_j's neighbours—say v_k such that $(v_k, v_j) \in E$—followed by a final step from v_k to v_j.

Theorem 12.4 (Bellman's equations)**.** *The quantities $u_j = d(v_1, v_j)$ satisfy the equations*

$$u_1 = 0 \qquad \textit{and} \qquad u_j = \min_{k \neq j}(u_k + w_{k,j}) \qquad \textit{for } 2 \leq j \leq n. \tag{12.1}$$

Further, if all cycles in $G(V, E, w)$ have positive weight, then the equations (12.1) have a unique solution.

12.4 Appendix: BFS revisited

The last two steps of our informal introduction to BFS had the general form

- Set $d(v) = j + 1$ for those vertices that (a) are adjacent to vertices t with $d(t) = j$ and (b) have not yet had a values of $d(v)$ assigned.

The main technical problem in formalising the algorithm is to find a systematic way of working our way outward from the source vertex. A data structure from computer science called a *queue* provides an elegant solution. It's an ordered list that we'll write from left-to-right, so that a queue containing vertices might look like

$$Q = \{x, z, u, w, \ldots, a\},$$

where x is the first entry in the queue and a is the last.

Just as in a well-behaved bus or bakery queue, we "serve" the vertices in order (left to right) and require that any new items added to the queue get added at the end (the right). There are two operations that one can perform on a queue:

push: add a new entry onto the end of the queue (i.e. at the right);

pop: remove the first (i.e. leftmost) entry and, typically, do something with it.

Thus if our queue is $Q = \{b, c, a\}$ and we push a vertex d onto it the result is

$$\{b, c, a\} \xrightarrow{\text{push } x \text{ onto } Q} \{b, c, a, x\},$$

while if we pop the queue we get

$$\{b, c, a\} \xrightarrow{\text{pop } Q} \{c, a\}.$$

We can use this idea to organise the order in which we visit the vertices in BFS. Our goal will be to compute $d(v) = d(s, v)$ for all vertices in the graph and we'll start by setting $d(s) = 0$ and

$$d(v) = \infty \; \forall v \neq s.$$

This has two advantages: first, $d(v) = \infty$ is the correct value for any vertex that is not reachable from s and second, it serves as a way to indicate that, as far as our algorithm is concerned, we have yet to visit vertex v and so $d(v)$ is yet-to-be-determined.

We'll then work our way through the vertices that lie in the same connected component as s by

- pushing a vertex v onto the end of the queue whenever we set $d(v)$, beginning with the source vertex s and

- popping vertices off the queue in turn, working through the adjacency list of the popped vertex u and examining its neighbours $w \in A_u$ in turn, setting $d(w) = d(u) + 1$ whenever $d(w)$ is currently marked as yet-to-be-determined.

Algorithm 12.5 (BFS for SSSP)**.** *Given an undirected graph $G(V, E)$ and a distinguished source vertex $s \in V$, assume uniform edge weights $w(e) = 1\ \forall e \in E$ and find the distances $d(v) = d(s, v)$ for all $v \in V$.*

(1) *Set things up:*

$d(v) \leftarrow \infty\ \ \forall v \neq s \in V$	*Set $d(v) = \infty$ to indicate that it is yet-to-be-determined*
$d(s) \leftarrow 0$	*Note that we know $d(s, s) = 0$*
$Q \leftarrow \{s\}$	*Get ready to process s's neighbours*

(2) *Main loop: continues until the queue is empty*

```
While ( Q ≠ ∅ ) {
        Pop a vertex u off the left end of Q.

        Examine each of u's neighbours
        For each  w ∈ A_u {
            If(  d(w) = ∞  ) then {
                Set d(w) and get ready to process w's neighbours
                d(w) ← d(u) + 1
                Push w onto the right end of Q.
            }
        }
}
```

Figure 12.4 illustrates the early stages of applying the algorithm to a small graph, while Table 12.1 provides a complete account of the computation: a set of PowerPoint-like slides illustrating this computation is available from the course web page.

Remarks

- If $d(u) = \infty$ when the algorithm finishes, then u and s lie in separate connected components.
- Because the computation works through adjacency lists, each edge gets considered at most twice and so the algorithm requires $O(|E|)$ steps, where a step consists of checking whether $d(u) = \infty$ and, if so, updating its value.
- It is possible to prove by induction on the lengths of the shortest paths, that BFS really does compute the distance $d(s, v)$: interested readers should see Jungnickel's Theorem 3.3.2.

u	$w \in A_u$	Action	Resulting Queue
–	–	*Start*	$\{S\}$
S	A	set $d(A) = 1$ and push A	$\{A\}$
S	C	set $d(C) = 1$ and push C	$\{A, C\}$
S	G	set $d(G) = 1$ and push G	$\{A, C, G\}$
A	B	set $d(B) = 2$ and push B	$\{C, G, B\}$
A	S	none, as $d(S) = 0$	$\{C, G, B\}$
C	D	set $d(D) = 2$ and push D	$\{G, B, D\}$
C	E	set $d(E) = 2$ and push E	$\{G, B, D, E\}$
C	F	set $d(F) = 2$ and push F	$\{G, B, D, E, F\}$
C	S	none	$\{G, B, D, E, F\}$
G	F	none	$\{B, D, E, F\}$
G	H	set $d(H) = 2$ and push H	$\{B, D, E, F, H\}$
G	S	none	$\{B, D, E, F, H\}$
B	A	none	$\{D, E, F, H\}$
D	C	none	$\{E, F, H\}$
E	C	none	$\{F, H\}$
E	H	none	$\{F, H\}$
F	C	none	$\{H\}$
F	G	none	$\{H\}$
H	E	none	$\{\}$
H	G	none	$\{\}$

Table 12.1: *A complete record of the execution of the BFS algorithm for the graph in Figure 12.4. Each row corresponds to one pass through the innermost loop of Algorithm 12.5, those steps that check whether $d(w) = \infty$ and act accordingly. The table is divided into sections—separated by horizontal lines—within which the algorithm works through the adjacency list of the most recently-popped vertex u.*

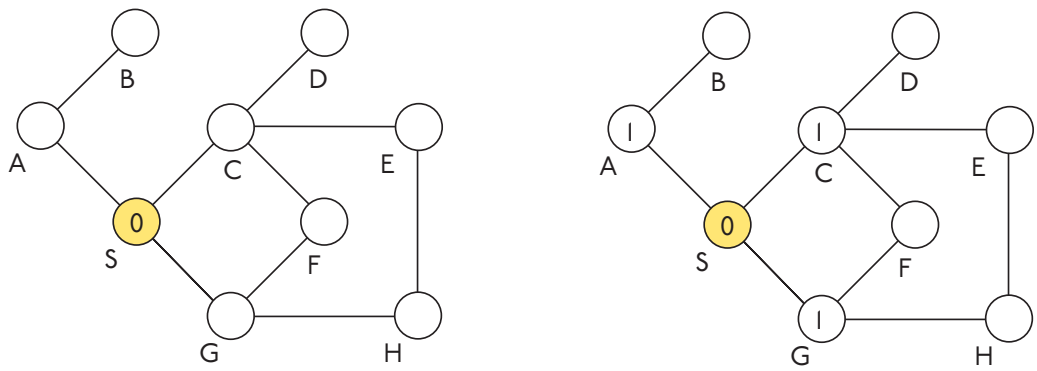


Figure 12.4: *In the graphs above the source vertex s is shown in yellow and a vertex v is shown with a number on it if, at that stage in the algorithm, $d(v)$ has been determined (that is, if $d(v) \neq \infty$). The graph at left illustrates the state of the computation just after the initialisation: $d(s)$ has been set to $d(s) = 0$, all other vertices have $d(v) = \infty$ and the queue is $Q = \{s\}$. The graph at right shows the state of the computation after we have popped s and processed its neighbours: $d(v)$ has been determined for A, C and G and they have been pushed onto the queue, which is now $Q = \{A, C, G\}$.*

Lecture 13

Tropical Arithmetic and Shortest Paths

This lecture introduces *tropical arithmetic*[1] and explains how to use it to calculate the lengths of all the shortest paths in a graph.

Reading:
The material here is not discussed in any of the main references for the course. The lecture is meant to be self-contained, but if you find yourself intrigued by tropical mathematics, you might want to look at a recent introductory article

> David Speyer and Bernd Sturmfels (2004), Tropical mathematics. Lecture notes from a Clay Mathematics Institute Senior Scholar Lecture, Park City, Utah, 22 July 2004. Available as preprint 0408099 from the arXiv preprint repository.

Very keen, mathematically sophisticated readers might also enjoy

> Diane Maclagan and Bernd Sturmfels (2015), *Introduction to Tropical Geometry*, Vol. 161 of *Graduate Studies in Mathematics*, American Mathematical Society, Providence, RI. ISBN: 978-0-8218-5198-2

while those interested in applications might prefer

> B. Heidergott, G. Olsder, and J. van der Woude (2006), *Max Plus at Work*, Vol. 13 of the *Princeton Series in Applied Mathematics*, Princeton Univ Press. ISBN: 978-0-6911-1763-8.

The book by Heidergott *et al.* includes a tropical model of the Dutch railway network and is more accessible than either the book by Maclagan and Sturmfels or the latter parts of the article by Speyer and Sturmfels.

[1] Maclagan & Sturmfels write: The adjective"tropical" was coined by French mathematicians, notably Jean-Eric Pin, to honor their Brazilian colleague Imre Simon, who pioneered the use of min-plus algebra in optimization theory. There is no deeper meaning to the adjective "tropical". It simply stands for the French view of Brazil.

13.1 All pairs shortest paths

In a previous lecture we used Breadth First Search (BFS) to solve the single-source shortest paths problem in a weighted graph $G(V, E, w)$ where the weights are trivial in the sense that $w(e) = 1 \ \forall e \in E$. Today we'll consider the problem where the weights can vary from edge to edge, but are constrained so that all cycles have positive weight. This ensures that Bellman's equations have a unique solution. Our approach to the problem depends on two main ingredients: a result about powers of the adjacency matrix and a novel kind of arithmetic.

13.2 Counting walks using linear algebra

Our main result is very close in spirit to the following, simpler one.

Theorem 13.1 (Powers of the adjacency matrix count walks)**.** *Suppose $G(V, E)$ is a graph (directed or undirected) on $n = |V|$ vertices and that A is its adjacency matrix. If we define A^ℓ, the ℓ-th matrix power of A, by*

$$A^{\ell+1} = A^\ell A \qquad \text{and} \qquad A^0 = I_n,$$

where $\ell \in \mathbb{N}$, then for $\ell > 0$,

$$A^\ell_{ij} = \text{the number of walks of length } \ell \text{ from vertex } i \text{ to vertex } j, \tag{13.1}$$

where A^ℓ_{ij} is the i, j entry in A^ℓ.

Proof. We'll prove this by induction on ℓ, the number of edges in the walk. The base case is $\ell = 1$ and so $A^\ell = A^1 = A$ and A_{ij} certainly counts the number of one-step walks from vertex i to vertex j: there is either exactly one such walk, or none.

Now suppose the result is true for all $\ell \leq \ell_0$ and consider

$$A^{\ell_0+1}_{ij} = \sum_{k=1}^{n} A^{\ell_0}_{ik} A_{kj}.$$

The only nonzero entries in this sum appear for those values of k for which both $A^{\ell_0}_{ik}$ and A_{kj} are nonzero. Now, the only possible nonzero value for A_{kj} is 1, which happens when the edge (k, j) is present in the graph. Thus we could also think of the sum above as running over vertices k such that the edge (k, j) is in E:

$$A^{\ell_0+1}_{ij} = \sum_{\{k|(k,j)\in E\}} A^{\ell_0}_{ik}.$$

By the inductive hypothesis, $A^{\ell_0}_{ik}$ is the number of distinct, length-ℓ_0 walks from i to k. And if we add the edge (k, j) to the end of such a walk, we get a walk from i to j. All the walks produced in this way are clearly distinct (those that pass through different intermediate vertices k are obviously distinct and even those that have the same k are, by the inductive hypothesis, different somewhere along the i to k segment). Further, every walk of length $\ell_0 + 1$ from i to j must consist of a length-ℓ_0 walk from i to some neighbour k of j, followed by a step from k to j, so we have completed the inductive step. $\square$

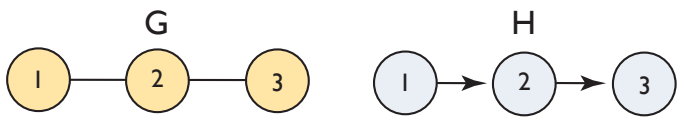


Figure 13.1: *In G, the graph at left, any walk from vertex 1 to vertex 3 must have even length while in H, the directed graph at right, there are no walks of length 3 or greater.*

Two examples

The graph at left in Figure 13.1 contains six walks of length 2. If we represent them with vertex sequences they're

$$(1,2,1),\ (1,2,3),\ (2,1,2),\ (2,3,2),\ (3,2,1),\ \text{ and } (3,2,3), \tag{13.2}$$

while the first two powers of A_G, G's adjacency matrix, are

$$A_G = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad A_G^2 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{13.3}$$

Comparing these we see that the computation based on powers of A_G agrees with the list of paths, just as Theorem 13.1 leads us to expect:

- $A^2_{1,1} = 1$ and there is a single walk, (1, 2, 1), from vertex 1 to itself;
- $A^2_{1,3} = 1$: counts the single walk, (1, 2, 3), from vertex 1 to vertex 3;
- $A^2_{2,2} = 2$: counts the two walks from vertex 2 to itself, (2,1,2) and (2,3,2);
- $A^2_{3,1} = 1$: counts the single walk, (3, 2, 1), from vertex 3 to vertex 1;
- $A^2_{3,3} = 1$: counts the single walk, (3, 2, 3), from vertex 3 to itself.

Something similar happens for the directed graph H that appears at right in Figure 13.1, but it has only a single walk of length two and none at all for lengths three or greater.

$$A_H = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad A_H^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad A_H^3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{13.4}$$

An alternative to BFS

The theorem we've just proved suggests a way to find all the shortest paths in the special case where $w(e) = 1\ \forall e \in E$. Of course, in this case the weight of a path is the same as its length.

(1) Compute the sequence of powers of the adjacency matrix A, A^2, $\cdots$, A^{n-1}.

(2) Observe that a shortest path has length at most $n-1$

(3) To find the length of a shortest path from v_i to v_j, look through the sequence of matrix powers and find the smallest ℓ such that $A^\ell_{ij} > 0$. This ℓ is the desired length.

In the rest of the lecture we'll generalise this strategy to graphs with arbitrary weights.

13.3 Tropical arithmetic

Tropical arithmetic acts on the set $\mathbb{R} \cup \{\infty\}$ and has two binary operations, $\oplus$ and $\otimes$, defined by

$$x \oplus y = \min(x, y) \qquad \text{and} \qquad x \otimes y = x + y \tag{13.5}$$

where, in the definition of $\otimes$, $x + y$ means ordinary addition of real numbers supplemented by the extra rule that $x \otimes \infty = \infty$ for all $x \in \mathbb{R} \cup \{\infty\}$. These novel arithmetic operators have many of the properties familiar from ordinary arithmetic. In particular, they are *commutative*. For all $a, b \in \mathbb{R} \cup \{\infty\}$ we have both

$$a \oplus b = \min(a, b) = \min(b, a) = b \oplus a$$

and

$$a \otimes b = a + b = b + a = b \otimes a.$$

The tropical arithmetic operators are also *associative*:

$$a \oplus (b \oplus c) = \min(a, \min(b, c)) = \min(a, b, c) = \min(\min(a, b), c) = (a \oplus b) \oplus c$$

and

$$a \otimes (b \otimes c) = a + b + c = (a \otimes b) \otimes c.$$

Also, there are distinct additive and multiplicative *identity elements*:

$$a \oplus \infty = \min(a, \infty) = a \qquad \text{and} \qquad b \otimes 0 = 0 + b = b.$$

Finally, the multiplication is *distributive*:

$$a \otimes (b \oplus c) = a + \min(b, c) = \min(a + b,\ a + c) = (a \otimes b) \oplus (a \otimes c).$$

There are, however, important differences between tropical and ordinary arithmetic. In particular, there are no additive inverses[2] in tropical arithmetic and so one cannot always solve linear equations. For example, there is no $x \in \mathbb{R} \cup \{\infty\}$ such that $(2 \otimes x) \oplus 5 = 11$. To see why, rewrite the equation as follows:

$$(2 \otimes x) \oplus 5 = (2 + x) \oplus 5 = \min(2 + x, 5) \leq 5.$$

[2]Students who did Algebraic Structures II might recognise that this collection of properties means that tropical arithmetic over $\mathbb{R} \cup \{\infty\}$ is a *semiring*.

13.3.1 Tropical matrix operations

Given two $m \times n$ matrices A and B whose entries are drawn from $\mathbb{R} \cup \{\infty\}$, we'll define the *tropical matrix sum* $A \oplus B$ by:

$$(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \min(A_{ij},\, B_{ij})$$

And for compatibly-shaped tropical matrices A and B we can also define the *tropical matrix product* by

$$(A \otimes B)_{ij} = \bigoplus_{k=1}^{n} A_{ik} \otimes B_{kj} = \min_{1 \le k \le n} (A_{ik} + B_{kj}).$$

Finally, if B is an $n \times n$ square matrix, we can define tropical matrix powers as follows:

$$B^{\otimes k+1} = B^{\otimes k} \otimes B \qquad \text{and} \qquad B^{\otimes 0} = \hat{I}_n. \tag{13.6}$$

where $\hat{I}_n$ is the $n \times n$ *tropical identity matrix*,

$$\hat{I}_n = \begin{bmatrix} 0 & \infty & & \dots & \infty \\ \infty & 0 & \infty & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \infty & 0 & \infty \\ \infty & \dots & & \infty & 0 \end{bmatrix}. \tag{13.7}$$

It has zeroes on the diagonal and ∞ everywhere else. It's easy to check that if A is an $m \times n$ tropical matrix then

$$\hat{I}_m \otimes A = A = A \otimes \hat{I}_n.$$

Example 13.2 (Tropical matrix operations)**.** *If we define two tropical matrices A and B by*

$$A = \begin{bmatrix} 1 & 2 \\ 0 & \infty \end{bmatrix} \qquad \textit{and} \qquad B = \begin{bmatrix} \infty & 1 \\ 1 & \infty \end{bmatrix}$$

then

$$A \oplus B \;=\; \begin{bmatrix} 1 \oplus \infty & 2 \oplus 1 \\ 0 \oplus 1 & \infty \oplus \infty \end{bmatrix} = \begin{bmatrix} \min(1, \infty) & \min(2, 1) \\ \min(0, 1) & \min(\infty, \infty) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & \infty \end{bmatrix}$$

and

$$\begin{aligned} A \otimes B &= \begin{bmatrix} (1 \otimes \infty) \oplus (2 \otimes 1) & (1 \otimes 1) \oplus (2 \otimes \infty) \\ (0 \otimes \infty) \oplus (\infty \otimes 1) & (0 \otimes 1) \oplus (\infty \otimes \infty) \end{bmatrix} \\ &= \begin{bmatrix} \min(1 + \infty,\, 2 + 1) & \min(1 + 1,\, 2 + \infty) \\ \min(0 + \infty,\, \infty + 1) & \min(0 + 1,\, \infty + \infty) \end{bmatrix} \\ &= \begin{bmatrix} 3 & 2 \\ \infty & 1 \end{bmatrix}. \end{aligned}$$

13.3.2 A tropical version of Bellman's equations

Recall Bellman's equations from Section 12.3.2. Given a weighted graph $G(V, E, w)$ in which all cycles have positive weight, we can find $u_j = d(v_1, v_j)$ by solving the system

$$u_1 = 0 \qquad \text{and} \qquad u_j = \min_{k \neq j} u_k + w_{k,j} \qquad \text{for } 2 \leq j \leq n,$$

where $w_{k,j}$ is an entry in a weight matrix w given by

$$w_{k,j} \;=\; \begin{cases} w(v_k, v_j) & \text{if } (v_k, v_j) \in E \\ \infty & \text{otherwise} \end{cases} . \tag{13.8}$$

We can rewrite Bellman's equations using tropical arithmetic

$$u_j \;=\; \min_{k \neq j} u_k + w_{k,j} \;=\; \min_{k \neq j} u_k \otimes w_{k,j} \;=\; \bigoplus_{k \neq j} u_k \otimes w_{k,j}$$

which looks almost like the tropical matrix product $u \otimes w$: we'll exploit this observation in the next section.

13.4 Minimal-weight paths in a tropical style

We'll now return to the problem of finding the weights of all the minimal-weight paths in a weighted graph. The calculations are very similar to those in Section 13.2, but now we'll take tropical powers of a weight matrix W whose entries are given by:

$$W_{k,j} \;=\; \begin{cases} 0 & \text{if } j = k \\ w(v_k, v_j) & \text{if } (v_k, v_j) \in E \\ \infty & \text{otherwise} \end{cases} . \tag{13.9}$$

Note that W is very similar to the matrix w defined by Eqn. (13.8): the two differ only along the diagonal, where $w_{ii} = \infty$ for all i, while $W_{ii} = 0$.

Lemma 13.3. *Suppose $G(V, E, w)$ is a weighted graph (directed or undirected) on n vertices. If all the cycles in G have positive weight and a matrix W is defined as in Eqn. (13.9), then the entries in $W^{\otimes \ell}$, the ℓ-th tropical power of W, are such that*

$$W_{ii}^{\otimes \ell} = 0 \textit{ for all } i$$

and, for $i \neq j$, either

$$W_{ij}^{\otimes \ell} = \textit{weight of a minimal-weight walk from } v_i \textit{ to } v_j \textit{ containing at most } \ell \textit{ edges}$$

when G contains such a walk or $W_{ij}^{\otimes \ell} = \infty$ if no such walks exist.

Proof. We proceed by induction on ℓ. The base case is $\ell = 1$ and it's clear that the only length-one walks are the edges themselves, while $W_{ii} = 0$ by construction.

Now suppose the result is true for all $\ell \leq \ell_0$ and consider the case $\ell = \ell_0 + 1$. We will first prove the result for the off-diagonal entries, those for which $i \neq j$. For these entries we have

$$W_{i,j}^{\otimes \ell_0+1} \;=\; \bigoplus_{k=1}^{n} W_{i,k}^{\otimes \ell_0} \otimes W_{k,j} \;=\; \min_{1\leq k\leq n} W_{ik}^{\otimes \ell_0} + W_{k,j} \tag{13.10}$$

and inductive hypothesis says that $W_{i,k}^{\otimes \ell_0}$ is either the weight of a minimal-weight walk from v_i to v_k containing ℓ_0 or fewer edges or, if no such walks exist, $W_{ik}^{\otimes \ell_0} = \infty$. $W_{k,j}$ is given by Eqn. (13.9) and so there are three possibilities for the terms

$$W_{i,k}^{\otimes \ell_0} + W_{k,j} \tag{13.11}$$

that appear in the tropical sum (13.10):

- They are infinite for all values of k, and so direct calculation gives $W_{i,j}^{\otimes \ell_0} = \infty$. This happens when, for each k, we have one or both of the following:
  - $W_{i,k}^{\otimes \ell_0} = \infty$, in which case the inductive hypothesis says that there are no walks of length ℓ_0 or less from v_i to v_k or
  - $W_{k,j} = \infty$ in which case there is no edge from v_k to v_j.

  And since this is true for all k, it implies that there are no walks of length $\ell_0 + 1$ or less that run from v_i to v_j. Thus the lemma holds when $i \neq j$ and $W_{ij}^{\otimes \ell_0+1} = \infty$.

- The expression in (13.11) is finite for at least one value of k, but not for $k = j$. Then we know $W_{ij}^{\otimes \ell_0} = \infty$ and so there are no walks of length ℓ_0 or less running from v_i to v_j. Further,

$$\begin{aligned} W_{i,j}^{\otimes \ell_0+1} &= \min_{k\neq j} W_{i,k}^{\otimes \ell_0} + W_{k,j} \\ &= \min_{\{k \,|\, (v_k,v_j)\in E\}} W_{ij}^{\otimes \ell_0} + w(v_k, v_j). \end{aligned} \tag{13.12}$$

  and reasoning such as we used when discussing Bellman's equations—a minimal-weight walk from v_i to v_j consists of a minimal weight walk from v_i to some neighbour (or, in a digraph, some predecessor) v_k of v_j, plus the edge (v_k, v_j)—means that the (13.12) gives the weight of a minimal-weight walk of length $\ell_0 + 1$ and so the lemma holds here too.

- The expression in (13.11) is finite for case $k = j$ and perhaps also for some $k \neq j$. When $k = j$ we have

$$W_{ik}^{\otimes \ell_0} + W_{k,j} \;=\; W_{ij}^{\otimes \ell_0} + W_{j,j} \;=\; W_{ij}^{\otimes \ell_0} + 0 \;=\; W_{ij}^{\otimes \ell_0},$$

  which, by the inductive hypothesis, is the weight of a minimal-weight walk of length ℓ_0 or less. If there are other values of k for which (13.11) is finite, then they give rise to a sum over neighbours (or, if G is a digraph, over predecessors) such as (13.12), which computes the weight of the minimal-weight walk of length $\ell_0 + 1$. The minimum of this quantity and $W_{ij}^{\otimes \ell_0}$ is then the minimal weight for any walk involving $\ell_0 + 1$ or fewer edges and so the lemma holds in this case too.

Finally, note that reasoning above works for $W_{ii}^{\otimes \ell}$ too: $W_{ii}^{\otimes \ell}$ is the weight of a minimal-weight walk from v_i to itself. And given that any walk from v_i to itself must contain a cycle and that all cycles have positive weight, we can conclude that the tropical sum

$$W_{i,i}^{\otimes \ell_0+1} \;=\; \min_k W_{i,k}^{\otimes \ell_0} + W_{k,i}$$

is minimised by $k = i$, when $W_{i,k}^{\otimes \ell_0} = W_{i,i}^{\otimes \ell_0} = 0$ (by the inductive hypothesis) and $W_{i,i} = 0$ (by construction) so

$$\min_k W_{i,k}^{\otimes \ell_0} + W_{k,i} \;=\; W_{i,i}^{\otimes \ell_0} + W_{i,i} \;=\; 0 + 0 \;=\; 0$$

and the theorem is proven for the diagonal entries too. □

Finally, we can state our main result:

Theorem 13.4 (Tropical matrix powers and shortest paths)**.** *If $G(V, E, w)$ is a weighted graph (directed or undirected) on n vertices in which all the cycles have positive weight, then $d(v_i, v_j)$, the weight of a minimal-weight path from v_i to v_j, is given by*

$$d(v_i, v_j) = W_{i,j}^{\otimes (n-1)} \tag{13.13}$$

Proof. First note that for $i \neq j$, any minimal-weight *walk* from v_i to v_j must actually be a minimal weight *path*. One can prove this by contradiction by noting that any walk that isn't a path must revisit at least one vertex. Say that $v_\star$ is one of these revisited vertices. Then the segment of the walk that runs from the first appearance of $v_\star$ to the second must have positive weight (it's a cycle and all cycles in G have positive weight) and so we can reduce the total weight of the walk by removing this cycle. But this contradicts our initial assumption that the walk had minimal weight.

Combining this with the previous lemma and the observation that a path in G contains at most $n - 1$ edges establishes the result. □

An example

The graph illustrated in Figure 13.2 is small enough that we can just read off the weights of its minimal-weight paths. If we assemble these results into a matrix D whose entries are given by

$$D_{ij} \;=\; \begin{cases} 0 & \text{if } i = j \\ d(v_i, v_j) & \text{if } i \neq j \text{ and } v_j \text{ is reachable from } v_i \\ \infty & \text{otherwise} \end{cases}$$

we get

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

To verify Theorem 13.4 we need only write down the weight matrix and its tropical square, which are

$$W = \begin{bmatrix} 0 & -2 & 1 \\ \infty & 0 & 1 \\ 2 & \infty & 0 \end{bmatrix} \qquad \text{and} \qquad W^{\otimes 2} = \begin{bmatrix} 0 & -2 & -1 \\ 3 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix}.$$

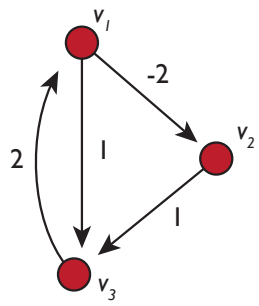


Figure 13.2: *The graph above contains two directed cycles,* (v_1, v_2, v_3, v_1)*, which has weight 1, and* (v_1, v_3, v_1)*, which has weight 2. Theorem 13.4 thus applies and we can compute the weights of minimal-weight paths using tropical powers of the weight matrix.*

The graph has $n = 3$ vertices and so we expect $W^{\otimes(n-1)} = W^{\otimes 2}$ to agree with the distance matrix D, which it does.

Lecture 14

Critical Path Analysis

This lecture applies ideas about distance in weighted graphs to solve problems in the scheduling of large, complex projects.

Reading:
The topic is discussed in Section 3.5 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, which is available online via SpringerLink,

but it is such an important application that it is also treated in many other places.

14.1 Scheduling problems

Suppose you are planning a dinner that involves a number of dishes, some of which have multiple components or stages of preparation, say, a roast with sauces or a pie with pastry and filling that have to be prepared separately. Especially for a novice cook, it can be difficult to arrange the cooking so that all the food is ready at the same moment. Somewhat surprisingly, graph theory can help in this, as well as in many more complex scheduling problems.

The key abstraction is a certain kind of directed graph constructed from a list of tasks such as the one in Table 14.1, which breaks a large project down into a list of smaller tasks and, for each one, notes (a) how long it takes to complete and (b) which other tasks are its immediate prerequisites. Here, for example, task A might be "wash and peel all the vegetables" while D and E—which have A as a prerequisite—might be "assemble the salad" and "fry the garlic briefly over very high heat."

14.1.1 From tasks to weighted digraphs

Figure 14.1 shows a directed graph associated with the project summarised in Table 14.1. It has:

- a vertex for each task;

Task	Time to Complete	Prerequisites
A	10	None
B	12	None
C	15	None
D	6	A & C
E	3	A & B
F	5	B & C
G	7	D & F
H	6	D & E
I	9	E & F

Table 14.1: *Summary of a modestly-sized project. The first column lists various tasks required for the completion of the project, while the second column gives the time (in minutes) needed to complete each task and the third column gives each task's immediate prerequisites.*

- edges that run from prerequisites to the tasks that depend on them. Thus for example, there is a directed edge (A, D), as task D has task A as a prerequisite.
- There are also two extra vertices, one called S (for "start") that is a predecessor for all the tasks that have no prerequisites and another, Z, that corresponds to finishing the project and is a successor of all tasks that are not prerequisites for any other task.
- There are edge weights that correspond to the time it takes to complete the task at the *tail* vertex. Thus—as task A takes 10 minutes to complete—all the edges coming out of the vertex A have weight 10.

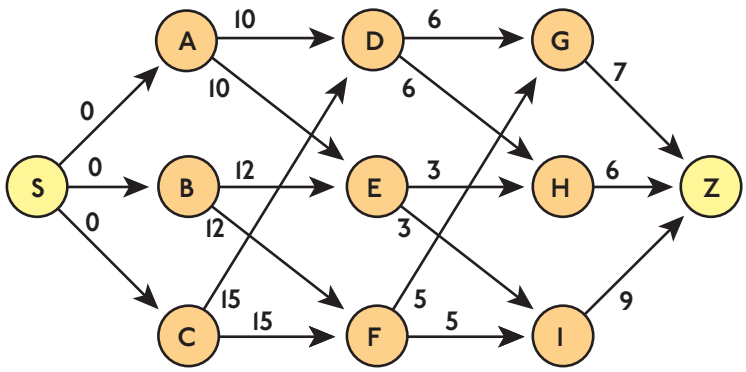


Figure 14.1: *The digraph associated with the scheduling problem from Table 14.1.*

Task	Prereq's
P: get a pink from	G
B: get a blue form	P
G: get a green form	B

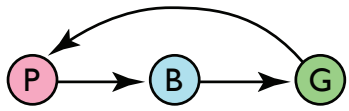


Figure 14.2: *An example showing why the digraph associated with a scheduling problem shouldn't contain cycles. It represents a bureaucratic nightmare in which one needs a pink form P in order to get a blue form B in order to get the green form G that one needs to get a pink form P.*

14.1.2 From weighted digraphs to schedules

Once we have a graph such as the one in Figure 14.1 we can answer a number of important questions about the project including:

(1) What is the shortest time in which we can complete the work?

(2) What is the earliest time (measured from the start of the project) at which we can start a given task?

(3) Are there any tasks whose late start would delay the whole project?

(4) For any tasks that don't need to be started as early as possible, how long can we delay their starts?

In the discussion that follows, we'll imagine that we have as many resources as we need (as many hands to help in the kitchen, as many employees and as much equipment as needed to pursue multiple tasks in parallel $\cdots$). In this setting, Lemma 14.1, proved below, provides a tool to answer all of these questions.

14.2 Graph-theoretic details

A directed graph representing a project that can actually be completed cannot contain any cycles. To see why, consider the graph in Figure 14.2. It tells us that we cannot start task G until we have completed its prerequisite, task B, which we cannot start before we complete its prerequisite, $P \cdots$ which we cannot start until we've completed G. This means we can never even start the project, much less finish it.

Thus any graph that describes a feasible project should be a directed, acyclic graph (often abbreviated *DAG*) with non-negative edge weights. From now on we'll restrict attention to such graphs and call them *task-dependency graphs*. We'll imagine that they are constructed from a project described by a list of tasks such as the one in Table 14.1 and that they look like the example in Figure 14.1. In particular, we'll require our graphs to have a *starting vertex* S which is the only vertex with $\deg_{in}(v) = 0$ and a *terminal vertex* Z, which is the only vertex with $\deg_{out}(v) = 0$.

Task	Time Needed	Prereq's
A	10 hours	None
B	6 hours	None

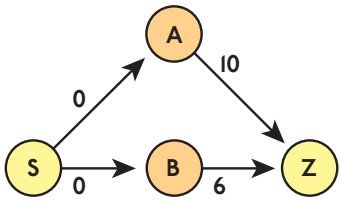


Figure 14.3: *An example showing why the shortest time in which one can complete a project corresponds to a* ***maximal-weight walk*** *from the start vertex S to the terminal vertex Z.*

14.2.1 Shortest times and maximal-weight paths

Now consider the very simple project illustrated in Figure 14.3. It involves just two tasks: A, which takes 10 hours to complete and B which takes 6 hours. Even if—as our assumptions allow—we start both tasks at the same time and work on them in parallel, the soonest we can possibly finish the project is 10 hours after we start. This is a special case of the following result, whose proof I'll only sketch briefly.

Lemma 14.1 (Shortest times and maximal weights)**.** *If $G(V, E, w)$ is a task-dependency graph that describes a scheduling problem, and if we start the work at $t = 0$, then the earliest time, t_v, at which we can start the task corresponding to vertex v is the weight of a maximal-weight walk from S to v.*

The proof of Lemma 14.1 turns on the observation that the times t_v satisfy equations that look similar to Bellman's Equations, except that they have a max() where Bellman's Equations have a min():

$$t_S = 0 \qquad \text{and} \qquad t_v = \max_{u \in P_v} (t_u + w(u, v)) \quad \forall\, v \neq S. \tag{14.1}$$

In the equation at right, P_v is v's predecessor list and $w(u, v)$ is the weight of the edge from u to v or, equivalently, the time it takes to complete the task corresponding to u.

Although the Bellman-like equations above provide an elegant characterisation of the t_v, they aren't necessarily all that practical as a way to *calculate* the t_v. The issue is that in order to use Eqn. (14.1) to compute t_v, we need t_u for all v's predecessors $u \in P_v$. And for each of them, we need t_w for $w \in P_u$ $\cdots$, and so on. Fortunately this problem has a simple resolution in DAGs, as we'll see below. The idea is to find a clever way to organise the computations so that the results we need when computing t_v are certain to be available.

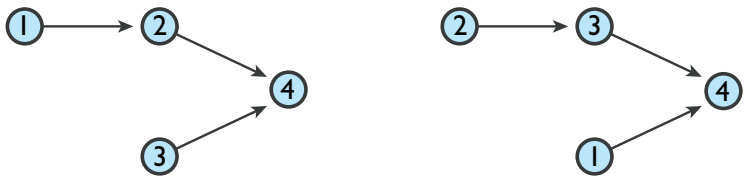


Figure 14.4: *A digraph with two distinct topological orderings.*

14.2.2 Topological ordering

Definition 14.2. *If $G(V, E)$ is a directed, acyclic graph with $|V| = n$, then a* ***topological ordering*** *(sometimes also called a* ***topological sorting****) of G is a map $\Phi : V \to \{1, 2, \ldots, n\}$ with the properties that*

- $\Phi(v) = \Phi(u) \;\Rightarrow\; u = v$;
- $(u, v) \in E \;\Rightarrow\; \Phi(u) < \Phi(v)$.

In other words, a topological ordering is a way of numbering the vertices so that the graph's directed edges always point from a vertex with a smaller index to a vertex with a bigger one.

Topological orderings are not, in general, unique, as is illustrated in Figure 14.4, but as the following results show, a DAG always has at least one.

Lemma 14.3 (DAGs contain sink vertices)**.** *If $G(V, E)$ is a directed, acyclic graph then it contains at least one vertex v with $\deg_{out}(v) = 0$. Such a vertex is sometimes called a* ***sink vertex****.*

Proof of Lemma 14.3. Construct a walk through $G(V, E)$ as follows. First choose an arbitrary vertex $v_0 \in V$. If $\deg_{out}(v_0) = 0$ we are finished, but if not choose an arbitrary successor of v_0, $v_1 \in S_{v_0}$. If $\deg_{out}(v_1) = 0$ we are finished, but if not, choose an arbitrary successor of v_1, $v_2 \in S_{v_1}$ $\cdots$ and so on. This construction can never revisit a vertex as G is acyclic. Further, as G has only finitely many vertices, the construction must come to a stop after at most $|V| - 1$ steps. But the only way for it to stop is to reach a vertex v_j such that $\deg_{out}(v_j) = 0$, which proves that such a vertex must exist. □

Theorem 14.4 (DAGs have topological orderings)**.** *A directed, acyclic graph $G(V, E)$ always has a topological ordering.*

Proof of Theorem 14.4. One can prove this by induction on the number of vertices. The base case is $|V| = 1$ and clearly, assigning the number 1 to the sole vertex gives a topological ordering.

Now suppose the result is true for all DAGs with $|V| \leq n_0$ and consider a DAG with $|V| = n_0 + 1$. Lemma 14.3 tells us that G contains a vertex w with $\deg_{out}(w) = 0$. Construct $G'(V', E') = G \backslash w$. It is a DAG (because G was one), but

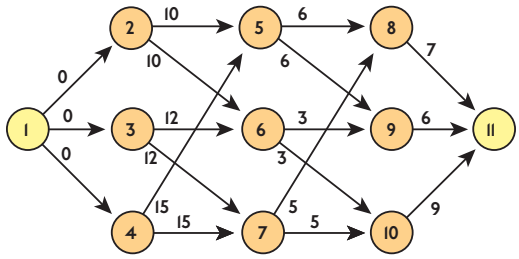


Figure 14.5: *A topological ordering for the digraph associated with the scheduling problem from Table 14.1 in which the vertex label v has been replaced by the value $\Phi(v)$ assigned by the ordering that's listed in Table 14.2.*

v	S	A	B	C	D	E	F	G	H	I	Z
$\Phi(v)$	1	2	3	4	5	6	7	8	9	10	11

Table 14.2: *The topological ordering illustrated in Figure 14.5.*

has only $|V'| = n_0$ vertices and so, by the inductive hypothesis, G' has a topological ordering $\Phi' : V' \to \{1, 2, \ldots, n_0\}$. We can extend this to a obtain a function $\Phi : V \to \{1, 2, \ldots, n_0 + 1\}$ by choosing

$$\Phi(v) = \begin{cases} \Phi'(v) & \text{if } v \neq w \\ n_0 + 1 & \text{if } v = w \end{cases}$$

Further, this Φ is clearly a topological ordering because all predecessors $u \in P_w$ of w have

$$\Phi(u) = \Phi'(u) \leq n_0 < n_0 + 1 = \Phi(w)$$

and, by construction, w has no successors. This concludes the inductive step and establishes that all DAGs have at least one topological ordering. □

14.3 Critical paths

Figure 14.5 shows a topological ordering for the graph from Figure 14.1. The reason we're interested in such orderings is that they provide a way to solve Eqns (14.1) in a task-dependency graph. By construction, the starting vertex S is the only one that has no predecessors and so any topological ordering must have $\Phi(S) = 1$. Similarly, the terminal vertex Z is the only one that has no successors, so for a project with n tasks, $\Phi(Z) = n + 2$.

By convention, we start the project at $t_S = 0$. If we then use Eqns (14.1) to compute the rest of the t_v, working through the vertex list in the order assigned by the topological ordering, it will always then be true that when we want to compute

$$t_v = \max_{u \in P_v} (t_u + w(u, v)) ,$$

we will have all the t_u for $u \in P_v$ available.

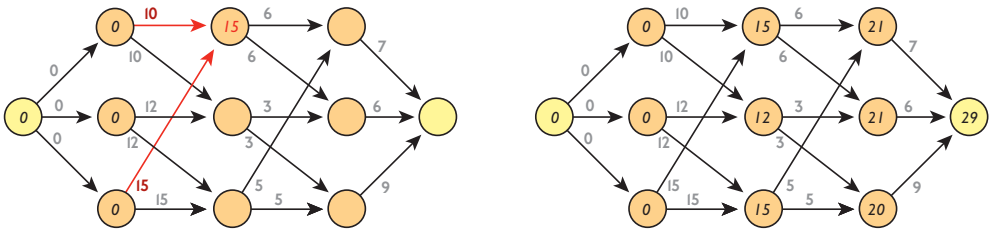


Figure 14.6: *In the graphs above the vertex labels have been replaced with values of t_j, the earliest times at which the corresponding task can start. The graph at left shows the edges that enter into Eqn.* (14.1) *for the computation of t_4 while the graph at right shows all the t_j.*

14.3.1 Earliest starts

For the digraph in Figure 14.5, we get

$$\begin{aligned}
t_S &= t_1 = 0 \\
t_A &= t_2 = t_1 + w(1,2) = 0 + 0 \\
&\vdots \\
t_D &= t_5 = \max(t_2 + w(2,5),\, t_4 + w(4,5)) = \max(0+10,\, 0+15) = 15 \qquad (14.2) \\
&\vdots
\end{aligned}$$

Figure 14.6 illustrates both the computation of t_4 and the complete set of t_j. As $t_Z = t_{11} = 29$, we can conclude that it takes a minimum of 29 minutes to complete the project.

14.3.2 Latest starts

The t_v that we computed in the previous section is the earliest time at which the task for vertx v *could* start, but it may be possible to delay the task without delaying the whole project. Consider, for example, task H in our main example. It could start as early as $t_H = t_9 = 21$, but since it only takes 6 minutes to complete, we could delay its start a bit without disrupting the project. If we define T_v to be the time by which task v *must* start if the project is not to be delayed, then it's clear that $T_H = T_Z - 6 = 23$. More generally, the latest time at which a task can start depends on the latest starts of its successors, so that

$$T_v = \min_{u \in S_v} \left(T_u - w(v,u)\right). \qquad (14.3)$$

This expression, along with the observation that $T_Z = t_Z$, allows us to find T_v for all tasks by working backwards through the DAG. Figure 14.7 illustrates this for our main example, while Table 14.3 lists t_v and T_v for all vertices $v \in V$.

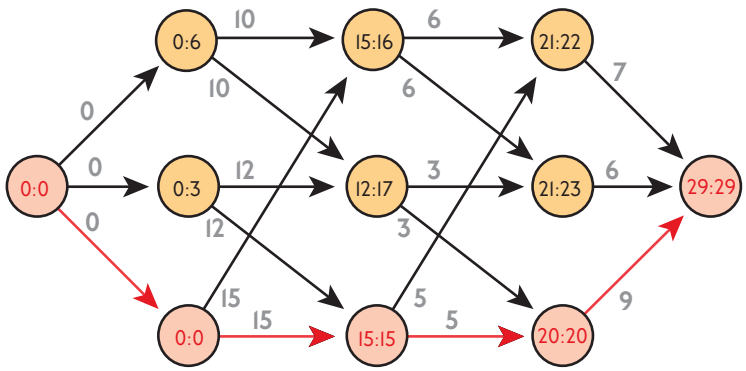


Figure 14.7: *Here a vertex v is labelled with a pair $t_v : T_v$ that shows both the earliest time t_v at which the corresponding task* ***could*** *start and T_v, the latest time by which the task* ***must*** *start if the whole project is not to be delayed. This project has only a single critical path, (S, C, F, I, Z), which is highlighted in red.*

v	S	A	B	C	D	E	F	G	H	I	Z
t_v	0	0	0	0	15	12	15	21	21	20	29
T_v	0	6	3	0	16	17	15	22	23	20	29

Table 14.3: *The earliest starts t_v and latest starts T_v for the main example.*

14.3.3 Critical paths

Notice that some of the vertices in Figure 14.7 have $t_v = T_v$. This happens because they lie on a maximal-weight path from S to Z and so a delay to any one of them will delay the whole project. Such maximal-weight paths play a crucial role in project management and so there is a term to describe them:

Definition 14.5 (Critical path)**.** *A maximal-weight path from S to Z in a task-dependency graph $G(V, E)$ is called a* ***critical path*** *and G may contain more than one of them.*

Vertices (and hence tasks) that lie off the critical path have $T_v > t_v$ and so do not require such keen supervision.

Part V

Planar Graphs

Lecture 15

Planar Graphs

This lecture introduces the idea of a planar graph—one that you can draw in such a way that the edges don't cross. Such graphs are of practical importance in, for example, the design and manufacture of integrated circuits as well as the automated drawing of maps. They're also of mathematical interest in that, in a sense we'll explore, there are really only two non-planar graphs.

Reading:
The first part of our discussion is based on that found in Chapter 10 of

> J. A. Bondy and U. S. R. Murty (2008), *Graph Theory*, Vol. 244 of Springer *Graduate Texts in Mathematics*, Springer Verlag,

but in subsequent sections I'll also draw on material from Section 1.5 of

> Dieter Jungnickel (2013), *Graphs, Networks and Algorithms*, 4th edition, which is available online via SpringerLink.

15.1 Drawing graphs in the plane

A graph G is said to be *planar* if it is possible to draw it in such a way that the edges intersect only at their end points (the vertices). Such a drawing is also called a *planar diagram* for G or a *planar embedding* of G. Indeed, it is possible to think of such a drawing—call it $\tilde{G}$—as a graph isomorphic to G. Recall our original definition of a graph: it involved only a vertex set V and a set E of pairs of vertices. Take the vertex set of $\tilde{G}$ to be the set of end points of the arcs in the drawing and say that the edge set consists of pairs made up of the two of end points of each arc.

15.1.1 The topology of curves in the plane

To give a clear treatment of this topic, it's helpful to use some ideas from plane topology. That takes us outside the scope of this module and so, in this subsection, I'll give some definitions and state one main result without proof. If you find this material interesting (and it *is* pretty interesting, as well as beautiful and useful) you might consider doing MATH31052, Topology.

Definition 15.1. *A* ***curve*** *in the plane is a continuous image of the unit interval. That is, a curve is a set of points*

$$C = \left\{\gamma(t) \in \mathbb{R}^2 \mid 0 \leq t \leq 1\right\}$$

traced out as t varies across the closed unit interval. Here $\gamma(t) = ((x(t), y(t))$, where $x(t) : [0,1] \to \mathbb{R}$ and $y(t) : [0,1] \to \mathbb{R}$ are continuous functions. If the curve does not intersect itself (that is, if $\gamma(t_1) = \gamma(t_2) \Rightarrow t_1 = t_2$) then it is a ***simple curve****.*

Definition 15.2. *A* ***closed curve*** *is a continuous image of the unit circle or, equivalently, a curve in which $\gamma(0) = \gamma(1)$. If a closed curve doesn't intersect itself anywhere other than at $\gamma(0) = \gamma(1)$, then it is a* ***simple closed curve****.*

Figure 15.1 and Table 15.1 give examples of these two definitions, while the following one, which is illustrated in Figure 15.2, sets the stage for this section's key result.

Figure 15.1: *From left to right: γ_1, a simple curve; γ_2, a curve that has an intersection, so is not simple; γ_3, a simple closed curve and γ_4, a closed curve with an intersection. Explicit formulae for the curves and their intersections appear in Table 15.1.*

Definition 15.3. *A set $S \subset \mathbb{R}^2$ is* ***arcwise-connected*** *if, for every pair of points $x, y \in S$, there is a curve $\gamma(t) : [0,1] \to S$ with $\gamma(0) = x$ and $\gamma(1) = y$.*

Theorem 15.4 (The Jordan Curve Theorem)**.** *A simple closed curve C in the plane divides the rest of the plane into two disjoint, arcwise-connected, open sets. These two open sets are called the* ***interior*** *and* ***exterior*** *of C, often denoted $\mathrm{Int}(C)$ and $\mathrm{Ext}(C)$, and any curve joining a point $x \in \mathrm{Int}(C)$ to a point $y \in \mathrm{Ext}(C)$ intersects C at least once.*

This is illustrated in Figure 15.3.

Curve	$x(t)$	$y(t)$
$\gamma_1(t)$	$2t$	$24t^3 - 36t^2 + 14t - 1$
$\gamma_2(t)$	$24t^3 - 36t^2 + 14t - 1$	$8t^2 - 8t + 1$
$\gamma_3(t)$	$\cos(2\pi t)$	$\sin(2\pi t)$
$\gamma_4(t)$	$\sin(4\pi t)$	$\sin(2\pi t)$

Table 15.1: *Explicit formulae for the curves appearing in Figure 15.1. The intersection in γ_2 occurs at $\gamma_2\left(\frac{1}{2} - \sqrt{\frac{1}{6}}\right) = \gamma_2\left(\frac{1}{2} + \sqrt{\frac{1}{6}}\right) = \left(0, \frac{1}{3}\right)$, while the one for γ_4 happens where $\gamma_4(0) = \gamma_4\left(\frac{1}{2}\right) = (0, 0)$.*

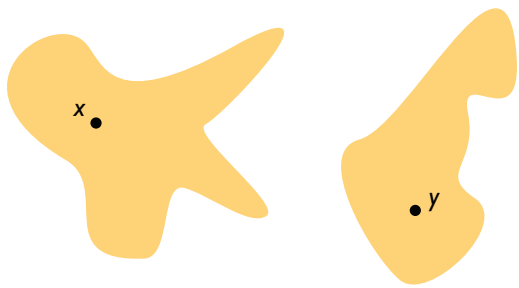


Figure 15.2: *The two shaded regions below are, individually, arcwise connected, but their union is not: any curve connecting x to y would have to pass outside the shaded regions.*

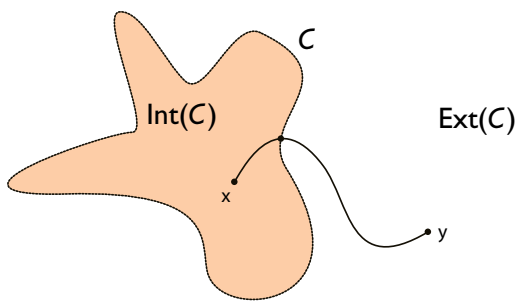


Figure 15.3: *An illustration of the Jordan Curve Theorem.*

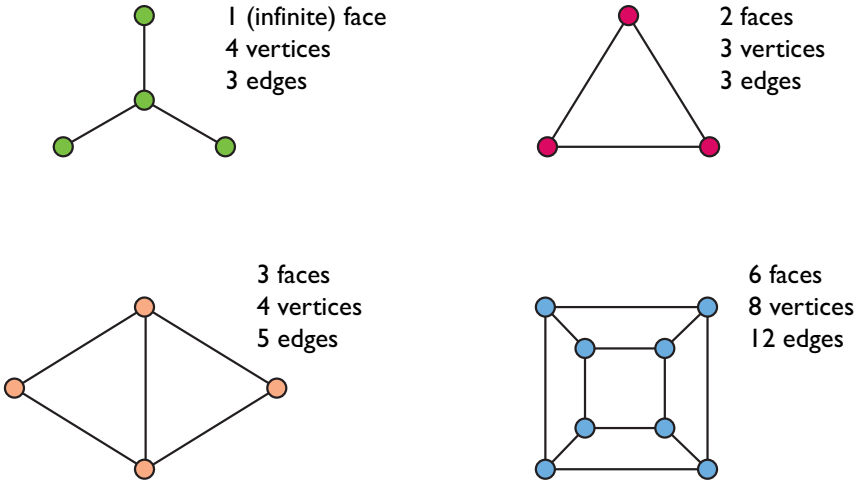


Figure 15.4: *Four examples of planar graphs, with numbers of faces, vertices and edges for each.*

15.1.2 Faces of a planar graph

The definitions in the previous section allow us to be a bit more formal about the definition of a planar graph:

Definition 15.5. *A* ***planar diagram*** *for a graph $G(V, E)$ with edge set $E = \{e_1, \ldots, e_m\}$ is a collection of simple curves $\{\gamma_1, \ldots, \gamma_m\}$ that represent the edges and have the property that the curves γ_j and γ_k corresponding to two distinct edges e_j and e_k intersect if and only if the two edges are incident on the same vertex and, in this case, they intersect only at the endpoints that correspond to their common vertex.*

Definition 15.6. *A graph G is* ***planar*** *if and only if it has a planar diagram.*

If a planar graph G contains cycles then the curves that correspond to the edges in the cycles link together to form simple closed curves that divide the plane into finitely many disjoint open sets called *faces*. Even if the graph has no cycles, there will still be one *infinite face*: see Figure 15.4.

15.2 Euler's formula for planar graphs

Our first substantive result about planar graphs is:

Theorem 15.7 (Euler's formula)**.** *If $G(V, E)$ is a connected planar graph with $n = |V|$ vertices and $m = |E|$ edges, then any planar diagram for G has $f = 2+m-n$ faces.*

Before giving a full proof, we begin with an easy special case:

Lemma 15.8 (Euler's formula for trees)**.** *If $G(V, E)$ is a tree then $f = 2 + m - n$.*

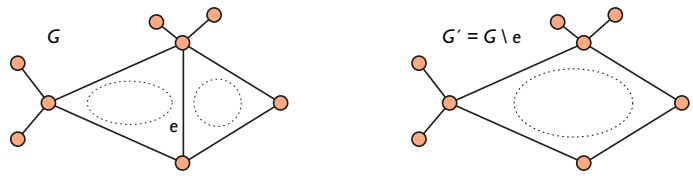


Figure 15.5: *Deleting the edge e causes two adjacent faces in G to merge into a single face in G'.*

Proof of the lemma about trees: As G is a tree we know $m = n - 1$, so

$$2 + m - n \;=\; 2 + (n-1) - n \;=\; 2 - 1 \;=\; 1 \;=\; f,$$

where the last equality follows because every planar diagram for a tree has only a single, infinite face. □

Proof of Euler's formula in the general case: We'll prove the result for arbitrary connected planar graphs by induction on m, the number of edges.

Base case The smallest connected planar graph contains only a single vertex, so has $n = 1$, $m = 0$ and $f = 1$. Thus

$$2 + m - n = 2 + 0 - 1 = 1 = f$$

just as Euler's formula demands.

Inductive step Suppose the result is true for all $m \leq m_0$ and consider a connected planar graph $G(V, E)$ with $|E| = m = m_0 + 1$ edges. Also suppose that G has n vertices and a planar diagram with f faces. Then one of the following things is true:

- G is a tree, in which case Euler's formula is true by the lemma proved above;
- G contains at least one cycle.

If G contains a cycle, choose an edge $e \in E$ that's part of that cycle and form $G' = G \backslash e$, which has $m' = m_0$ edges, $n' = n$ vertices and $f' = f - 1$ faces. This last follows because breaking a cycle merges two adjacent faces, as is illustrated in Figure 15.5.

As G' has only m_0 edges, we can use the inductive hypothesis to say that $f' = m' - n' + 2$. Then, again using unprimed symbols for quantities in G, we have:

$$\begin{aligned} f' &= m' - n' + 2 \\ f - 1 &= m_0 - n + 2 \\ f &= (m_0 + 1) - n + 2 \\ f &= m - n + 2, \end{aligned}$$

which establishes Euler's formula for graphs that contain cycles.

□

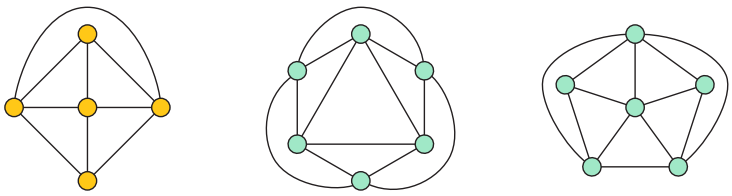


Figure 15.6: *Planar graphs with the maximal number of edges for a given number of vertices. The graph with the yellow vertices has $n = 5$ and $m = 9$ edges, while those with the blue vertices have $n = 6$ and $m = 12$*

15.3 Planar graphs can't have many edges

To set the scene for our next result, consider graphs on $n \in \{1, 2, \ldots, 5\}$ vertices and, for each n, try to draw a planar graph with as many edges as possible. At first this is easy: it's possible to find a planar diagram for each of the complete graphs K_1, K_2, K_3 and K_4, but, as we will prove below, K_5 is not planar and the the best one can do is to find a planar graph with $n = 5$ and $m = 9$. For $n = 6$ there are two non-isomorphic planar graphs with $m = 12$ edges, but none with $m \geq 12$. Figure 15.6 shows examples of planar graphs having the maximal number of edges.

Larger planar graphs (those with $n \gg 5$) tend to be even *sparser*, which means that they have many fewer edges than they could. The relevant comparison for a graph on n vertices is $n(n-1)/2$, the number of edges in the complete graph K_n, so we'll say that a graph is *sparse* if

$$|E| \ll \frac{n(n-1)}{2} \qquad \text{or} \qquad s \equiv \frac{|E|}{n(n-1)/2} \ll 1 \tag{15.1}$$

Table 15.2 makes it clear that when $n > 5$ the planar graphs become increasingly sparse[1].

15.3.1 Preliminaries: bridges and girth

The next two definitions will help us to formulate and prove our main result, a somewhat technical theorem that gives a precise sense to the intuition that a planar graph can't have very many edges.

Definition 15.9. *An edge e in a connected graph $G(V, E)$ is a* ***bridge*** *if the graph $G' = G \backslash e$ formed by deleting e has more than one connected component.*

Definition 15.10. *If a graph $G(V, E)$ contains one or more cycles then the* ***girth*** *of G is the length of a shortest cycle.*

These definitions are illustrated in Figures 15.7 and 15.8.

[1] I wrote software to compute the first few rows of this table myself, but got the counts for $n > 9$ from the *On-Line Encyclopedia of Integer Sequences*, entries A003094 and A001349.

n	m_{max}	s	Number of non-isomorphic, connected graphs that are ... planar, with $m = m_{max}$	planar	planar or not
5	9	0.9	1	20	21
6	12	0.8	2	99	112
7	15	0.714	5	646	853
8	18	0.643	14	5,974	11,117
9	21	0.583	50	71,885	261,080
10	24	0.583	?	1,052,805	11,716,571
11	27	0.533	?	17,449,299	1,006,700,565
12	30	0.491	?	313,372,298	164,059,830,476

Table 15.2: *Here m_{max} is the maximal number of edges appearing in a planar graph on the given number of vertices, while the column labelled s lists the measure of sparsity given by Eqn. 15.1 for connected, planar graphs with m_{max} edges. The remaining columns list counts of various kinds of graphs and make the point that as n increases, planar graphs with $m = m_{max}$ become rare in the set of all connected planar graphs and that this family itself becomes rare in the family of connected graphs.*

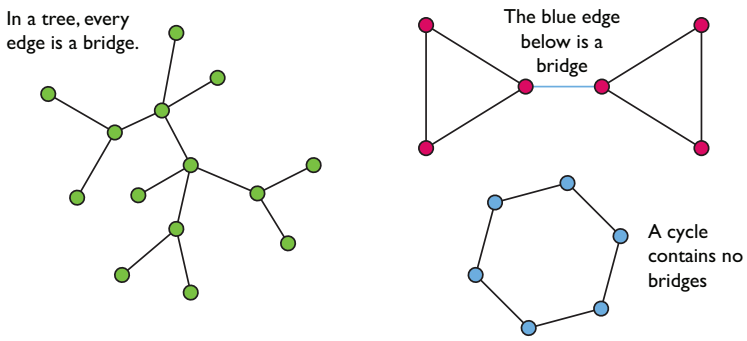


Figure 15.7: *Several examples of bridges in graphs.*

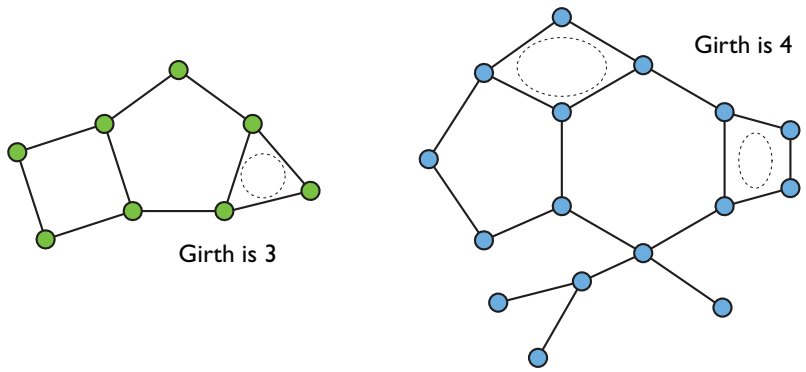


Figure 15.8: *The girth of a graph is the length of a shortest cycle.*

Remark 15.11. *A graph with n vertices has girth in the range $3 \leq g \leq n$. The lower bound arises because all cycles include at least three edges and the upper one because the longest possible cycle occurs when G is isomorphic to C_n.*

15.3.2 Main result: an inequality relating n and m

We are now in a position to state our main result:

Theorem 15.12 (Jungnickel's 1.5.3)**.** *If $G(V, E)$ is a connected planar graph with $n = |V|$ vertices and $m = |E|$ edges then either:*

A: G is acyclic and $m = n - 1$;

B: G has at least one cycle and so has a well-defined girth g. In this case

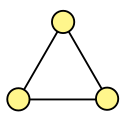
$$m \leq \frac{g(n-2)}{g-2}. \tag{15.2}$$

Outline of the Proof. We deal first with the case where G is acyclic and then move on to the harder, more general case:

A: G is connected, so if it has no cycles it's a tree and we've already proved (see Theorem 6.13) that trees have $m = n - 1$.

B: When G contains one or more cycles, we'll prove the inequality 15.2 mainly by induction on n, but we'll need several sub-cases. To see why, let's plan out the argument.

Base case: $n = 3$
There is only a single graph on three vertices that contains a cycle, it's K_3,



which has girth $g = 3$ and $n = 3$, so our theorem says

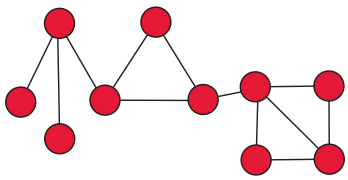
$$\begin{aligned} m &\leq \frac{g(n-2)}{g-2} \\ &\leq \frac{3 \times (3-2)}{3-2} \\ &\leq 3 \end{aligned}$$

which is obviously true.

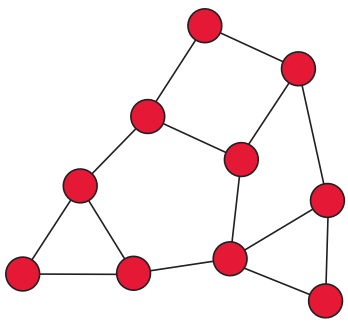
Inductive hypothesis:
Assume the result is true for all connected, planar graphs that contain a cycle and have $n \leq n_0$ vertices.

Inductive step:
Now consider a connected, planar graph $G(V, E)$ with $n_0 + 1$ vertices that contains a cycle. We need, somehow, to reduce this graph to one for which we can exploit the inductive hypothesis and so one naturally thinks of deleting something. This leads to two main sub-cases, which are illustrated[2] below.

B.1 G contains at least one bridge. In this case the road to a proof by induction seems clear: we'll delete the bridge and break G into two smaller graphs.



B.2 G does not contains any bridges. Equivalently, every edge in G is part of some cycle. Here it's less clear how to handle the inductive step and so we will use an altogether different, non-inductive approach.



We'll deal with these cases in turn, beginning with **B.1**.

As mentioned above, a natural approach is to delete a bridge and break G into two smaller graphs—say, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$—then apply the inductive

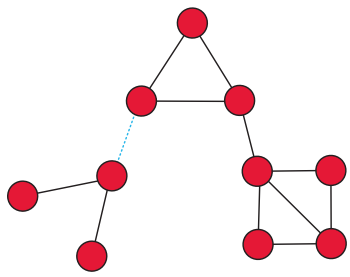
[2]The examples illustrating cases **B.1** and **B.2** are meant to help the reader follow the argument, but are *not* part of the logic of the proof.

hypothesis to the pieces. If we define $n_j = |V_j|$ to be the number of vertices in G_j and $m_j = |E_j|$ to be the corresponding number of edges, then we know

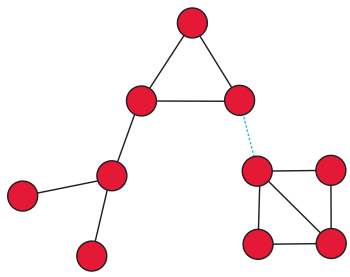
$$n_1 + n_2 = n \qquad \text{and} \qquad m_1 + m_2 = m - 1. \tag{15.3}$$

But we need to take a little care as deleting a bridge leads to two further sub-cases and we'll need a separate argument for each. Given that the original graph G contained at least one cycle—and noting that removing a bridge can't break a cycle—we know that at least one of the two pieces G_1 and G_2 contains a cycle. Our two sub-cases are thus:

B.1a Exactly one of the two pieces contains a cycle. We can assume without loss of generality that it's G_2, so that G_1 is a tree.



B.1b Both G_1 and G_2 contain cycles.



Thus we can complete the proof of Theorem 15.12 by producing arguments (full details below) that cover the following three possibilities

B.1a G contains a bridge and at least one cycle. Deleting the bridge leaves two subgraphs, a tree G_1 and a graph, G_2, that contains a cycle: we handle this possibility in Case 15.13 below.

B.1b G contains a bridge and at least two cycles. Deleting the bridge leaves two subgraphs, G_1 and G_2, each of which contains at least one cycle: see Case 15.14.

B.2 G contains one or more cycles, but no bridges: see Case 15.15.

□

15.3.3 Gritty details of the proof of Theorem 15.12

Before we plunge into the Lemmas, it's useful to make a few observations about the ratio $g/(g-2)$ that appears in Eqn. (15.2). Recall (from Remark 15.11) that if a graph on n vertices contains a cycle, then the girth is well-defined and lies in the range $3 \leq g \leq n$.

- For $g > 2$, the ratio $g/(g-2)$ is a monotonically decreasing function of g and so
$$g_1 > g_2 \;\Rightarrow\; \left(\frac{g_1}{g_1-2}\right) < \left(\frac{g_2}{g_2-2}\right). \tag{15.4}$$
- The monotonicity of $g/(g-2)$, combined with the fact that $g \geq 3$, implies that $g/(g-2)$ is bounded from above by 3:
$$g \geq 3 \;\Rightarrow\; \left(\frac{g}{g-2}\right) \leq \left(\frac{3}{3-2}\right) = 3. \tag{15.5}$$
- And at the other extreme, $g/(g-2)$ is bounded from below (strictly) by 1:
$$g \leq n \;\Rightarrow\; \left(\frac{g}{g-2}\right) \geq \left(\frac{n}{n-2}\right) > 1. \tag{15.6}$$

The three cases

The cases below are all part of an inductive argument in which, $G(V,E)$ is a connected planar graph with $|V| = n_0 + 1$ and $|E| = m$. It also contains at least one cycle and so has a well-defined girth, g. Finally, we have an inductive hypothesis saying that Theorem 15.12 holds for all trees and for all connected planar graphs with $|V| \leq n_0$.

Case 15.13 (Case **B.1a** of Theorem 15.12)**.** *Here G contains a bridge and deleting this bridge breaks G into two connected planar, subgraphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, one of which is a tree.*

Proof. We can assume without loss that G_1 is the tree and then argue that every cycle that appears in G is also in G_2 (we've only deleted a bridge), so the girth of G_2 is still g. Also, $n_1 \geq 1$, so $n_2 \leq n_0$ and, by the inductive hypothesis, we have

$$m_2 \;\leq\; \frac{g(n_2-2)}{g-2}.$$

But then, because G_1 is a tree, we know that $m_1 = n_1 - 1$. Adding this to both sides of the inequality yields

$$m_1 + m_2 \;\leq\; (n_1 - 1) + \frac{g(n_2-2)}{g-2}$$

or, equivalently,

$$m_1 + m_2 + 1 \;\leq\; n_1 + \frac{g(n_2 - 2)}{g - 2}.$$

Finally, noting that $m = m_1 + m_2 + 1$, we can say

$$\begin{aligned} m &\leq n_1 + \frac{g(n_2 - 2)}{g - 2} \\ &\leq \left(\frac{g}{g-2}\right) n_1 + \frac{g(n_2 - 2)}{g - 2} \\ &\leq \frac{g(n_1 + n_2 - 2)}{g - 2} \\ &\leq \frac{g(n - 2)}{g - 2}, \end{aligned}$$

which is the result we sought. Here the step from the first line to the second follows because $1 < g/(g-2)$ (recall Eqn. (15.6)), so

$$n_1 < \left(\frac{g}{g-2}\right) n_1$$

and the last line follows because $n = n_1 + n_2$. □

Case 15.14 (Case **B.1b** of Theorem 15.12)**.** *This case is similar to the previous one in that here again G contains a bridge, but in this case deleting the bridge breaks G into two connected planar, subgraphs, each of which contains at least one cycle (and so has a well defined-girth).*

Proof. We'll say that G_1 has girth g_1 and G_2 has girth g_2 and note that, as the girth is defined as the length of a *shortest* cycle—and as every cycle that appears in the original graph G must still be present in one of the G_j—we know that

$$g \leq g_1 \qquad \text{and} \qquad g \leq g_2. \tag{15.7}$$

Now, $n = n_0 + 1$ and $n = n_1 + n_2$ so as we know that $n_j \geq 3$ (the shortest possible cycle is of length 3 and the G_j contain cycles), it follows that we have both $n_1 < n_0$ and $n_2 < n_0$. This means that the inductive hypothesis applied to both G_j and so we have

$$m_1 \leq \frac{g_1(n_1 - 2)}{g_1 - 2} \qquad \text{and} \qquad m_2 \leq \frac{g_2(n_2 - 2)}{g_2 - 2}.$$

Adding these together yields:

$$\begin{aligned} m_1 + m_2 &\leq \frac{g_1(n_1 - 2)}{g_1 - 2} + \frac{g_2(n_2 - 2)}{g_2 - 2} \\ &\leq \frac{g(n_1 - 2)}{g - 2} + \frac{g(n_2 - 2)}{g - 2} \\ &\leq \frac{g(n_1 + n_2 - 4)}{g - 2}, \end{aligned}$$

where the step from the first line to the second follows from Eqn. 15.7 and the monotonicity of the ratio $g/(g-2)$ (recall Eqn. (15.4)). If we again note that $1 < g/(g-2)$ we can conclude that

$$\begin{aligned} m_1 + m_2 + 1 &\leq \frac{g(n_1+n_2-4)}{g-2} + 1 \\ &\leq \frac{g(n_1+n_2-4)}{g-2} + \frac{g}{g-2} \\ &\leq \frac{g(n_1+n_2-3)}{g-2} \end{aligned}$$

and so

$$m = m_1 + m_2 + 1 \;\leq\; \frac{g(n_1+n_2-3)}{g-2} \;\leq\; \frac{g(n_1+n_2-2)}{g-2}$$

or, as $n = n_1 + n_2$,

$$m \leq \frac{g(n-2)}{g-2},$$

which is the result we sought. □

Case 15.15 (Case **B.2** of Theorem 15.12)**.** *In the final case $G(V,E)$ does not contain any bridges, which implies that every edge in E is part of some cycle. This makes it harder to see how to use the inductive hypothesis (we'd have to delete two or more edges to break G into disconnected pieces . . .) and so we will use an entirely different argument based on Euler's Formula (Theorem 15.7).*

Proof. First, define f_j to be the number of faces whose boundary has j edges, making sure to include the infinite face: Figures 15.9 illustrates this definition. Then, as each edge appears in the boundary of exactly two faces we have both

$$\sum_{j=g}^{n} f_j = f \qquad \text{and} \qquad \sum_{j=g}^{n} j \times f_j = 2m.$$

Note that both sums start at g, the girth, as we know that there are no cycles of shorter length. But then

$$2m \;=\; \sum_{j=g}^{n} j \times f_j \;\geq\; \sum_{j=g}^{n} g \times f_j \;=\; g \sum_{j=g}^{n} f_j \;=\; gf,$$

where we obtain the inequality by replacing the length of the cycle j in $j \times f_j$ with g, the length of the shortest cycle (and hence the smallest value of j for which f_j is nonzero). Thus we have

$$2m \geq gf \qquad \text{or} \qquad f \leq 2m/g.$$

If we now use Euler's Theorem to say that $f = m - n + 2$, we have

$$m - n + 2 \;\leq\; \frac{2m}{g} \qquad \text{or} \qquad m - \frac{2m}{g} \;\leq\; n - 2.$$

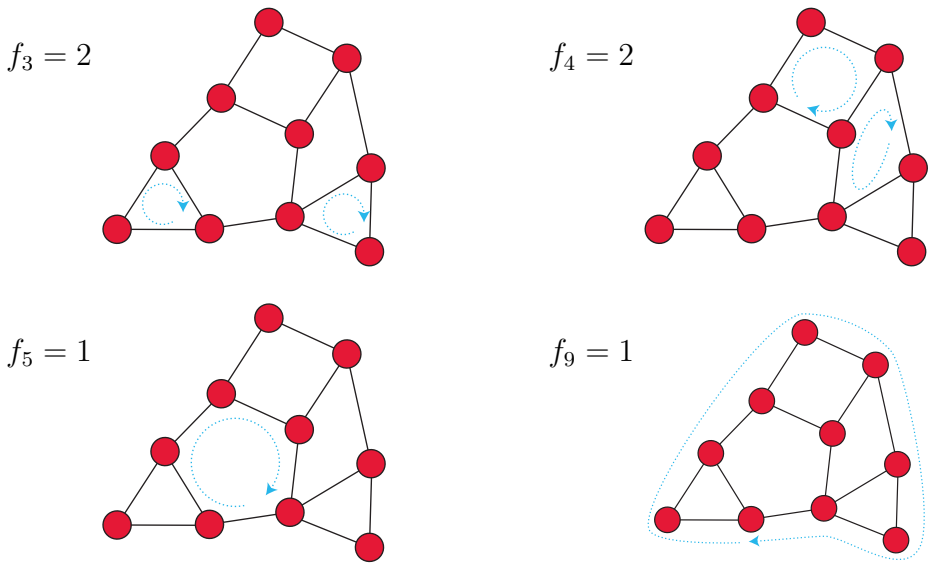


Figure 15.9: *The example used to illustrate case* ***B.2*** *of Theorem 15.12 has* $f_3 = 2$, $f_4 = 2$, $f_5 = 1$ *and* $f_9 = 1$ *(for the infinite face): all other* f_j *are zero.*

And then, finally,

$$\frac{gm}{g} - \frac{2m}{g} \leq n - 2 \qquad \text{so} \qquad \frac{(g-2)m}{g} \leq n - 2 \qquad \text{and} \qquad m \leq \frac{g(n-2)}{g-2}$$

which is the result we sought.

□

15.3.4 The maximal number of edges in a planar graph

Theorem 15.12 has an easy corollary that gives a simple bound on the maximal number of edges in a graph with $|V| = n$.

Corollary 15.16. *If* $G(V, E)$ *is a connected planar graph with* $n = |V| \geq 3$ *vertices and* $m = |E|$ *edges then* $m \leq 3n - 6$.

Proof. Either G is a tree, in which case $m = n - 1$ and the bound in the Corollary is certainly satisfied, or G contains at least one cycle. In the latter case, say that the girth of G is g. We know $3 \leq g \leq n$ and our main result says

$$m \leq \left(\frac{g}{g-2}\right)(n-2).$$

Thus, recalling that $g/(g-2) \leq 3$, the result follows immediately. □

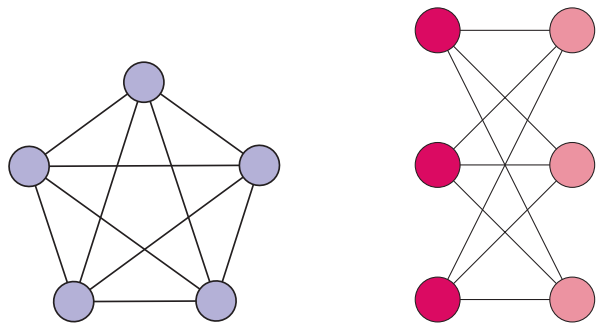


Figure 15.10: *Both K_5 and $K_{3,3}$ are non-planar.*

15.4 Two non-planar graphs

The hard-won inequalities from the previous section—which both say something like "G planar implies m small"—cannot be used to prove that a graph *is* planar[3], but can help establish that a graph *isn't*. The idea is to use the contrapositives, which are statements like "If m is too big, then G can't be planar."

To illustrate this, we'll use our inequalities to prove that neither of the graphs in Figure 15.10—K_5 at left and $K_{3,3}$ at right—is planar. Let's begin with K_5: it has $n = 5$ so Corollary 15.16 says that if it is planar,

$$m \leq 3n - 6 = 3 \times 5 - 6 = 15 - 6 = 9,$$

but K_5 actually has $m = 10$ edges, which is one too many for a planar graph. Thus K_5 can't have a planar diagram.

$K_{3,3}$ isn't planar either, but Corollary 15.16 isn't strong enough to establish this. $K_{3,3}$ has $n = 6$ and $m = 3 \times 3 = 9$. Thus it easily satisfies the bound from Corollary 15.16, which requires only that $m \leq 3 \times 6 - 6 = 12$. But if we now apply our main result, Theorem 15.12, we'll see that $K_{3,3}$ can't be planar. The relevant inequality is

$$\begin{aligned} m &\leq \frac{g(n-2)}{g-2} \\ &\leq \frac{4 \times (6-2)}{4-2} \\ &\leq \frac{16}{2} \\ &\leq 8 \end{aligned}$$

where, in passing from the first line to the second, I've used the fact that the girth of $K_{3,3}$ is $g = 4$. To see this, first note that any cycle in a bipartite graph has even

[3]There is an $O(n)$ algorithm that determines whether a graph on n vertices is planar and, if it is, produces a planar diagram. We don't have time to discuss it, but interested readers might like to look at John Hopcroft and Robert Tarjan (1974), Efficient Planarity Testing, *Journal of the ACM*, **21**(4):549–568. DOI: 10.1145/321850.321852

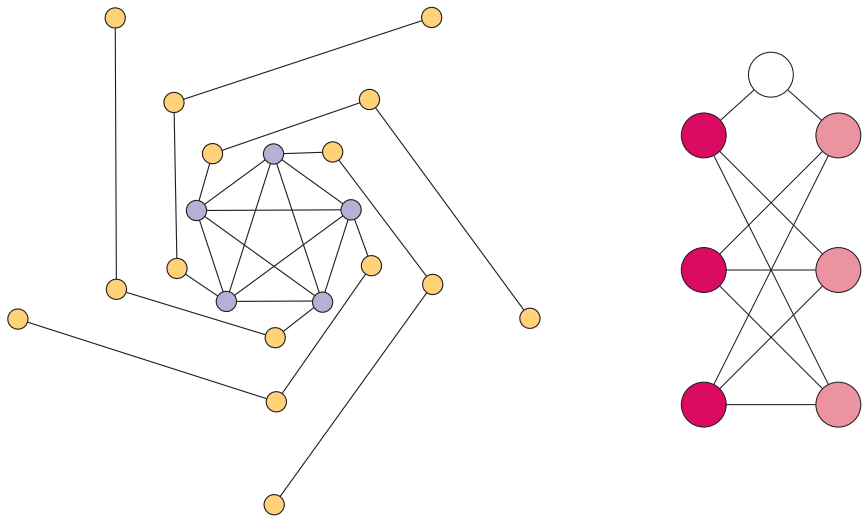


Figure 15.11: *Knowing that K_5 and $K_{3,3}$ are non-planar makes it clear that these two graphs can't be planar either, even though neither violates the inequalities from the previous section (check this).*

length, so the shortest possible cycle in $K_{3,3}$ has length 4, and then find such a cycle (there are lots).

Once we know that $K_{3,3}$ and K_5 are nonplanar, we can see immediately that many other graphs must be non-planar too, even when this would not be detected by either of our inequalities: Figure 15.11 shows two such examples. The one on the left has K_5 as a subgraph, so even though it satisfies the bound from Theorem 15.12, it can't be planar. The example at right is similar in that any planar diagram for this graph would obviously produce a planar diagram for $K_{3,3}$, but the sense in which this second graph "contains" $K_{3,3}$ is more subtle: we'll clarify and formalise this in the next section, then state a theorem that says, essentially, that *every* non-planar graph contains K_5 or $K_{3,3}$.

15.5 Kuratowski's Theorem

We begin with a pair of definitions designed to capture the sense in which the graph at right in Figure 15.11 contains $K_{3,3}$.

Definition 15.17. *A* ***subdivision*** *of a graph $G(V, E)$ is a graph $H(V', E')$ formed by (perhaps repeatedly) removing an edge $e = (a, b) \in E$ from G and replacing it with a path*

$$\{(a, v_1),\ (v_1, v_2),\ \ldots, (v_k, b)\}$$

containing of some number $k > 0$ of new vertices $\{v_1, \ldots, v_k\}$, each of which has degree 2.

Figure 15.12 shows a couple examples of subdivisions, including one at left that gives an indication of where the name comes from: the extra vertices can be thought of as dividing an existing edge into smaller ones.

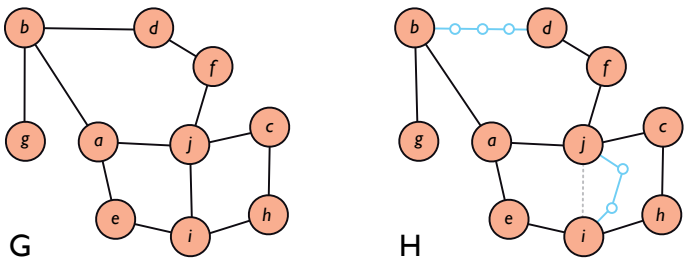


Figure 15.12: *H at right is a subdivision of G. The connection between b and d, which was a single edge in G, becomes a blue path in H: one can imagine that the original edge (b, d) has had three new, white vertices inserted into it, "sub-dividing" it. The other deleted edge, (i, j) is shown as a pale grey, dashed line (to indicate that it's not part of H), while the new path that replaces it is again shown in blue and white.*

Definition 15.18. *Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are said to be* ***homeomorphic*** *if they are isomorphic to subdivisions of the same graph.*

That is, we say G_1 and G_2 are homeomorphic if there is some third graph—call it G_0—such that both G_1 and G_2 are subdivisions of G_0. Figure 15.13 shows several graphs that are homeomorphic to K_5. Homeomorphism is an equivalence relation on graphs[4] and so all the graphs in Figure 15.13 are homeomorphic to each other as well as to K_5.

The notion of homeomorphism allows us to state the following remarkable result:

Theorem 15.19 (Kuratowski's Theorem (1930))**.** *A graph G is planar if and only if it does not contain a subgraph homeomorphic to K_5 or $K_{3,3}$.*

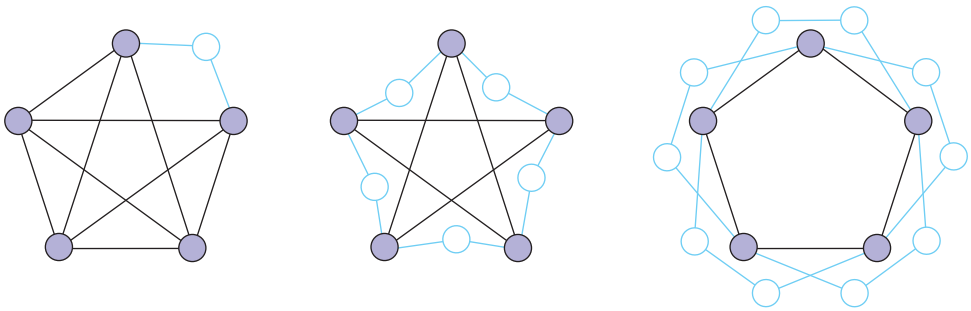


Figure 15.13: *These three graphs are homeomorphic to K_5, and hence also to each other.*

[4]The keen reader should check this for herself.

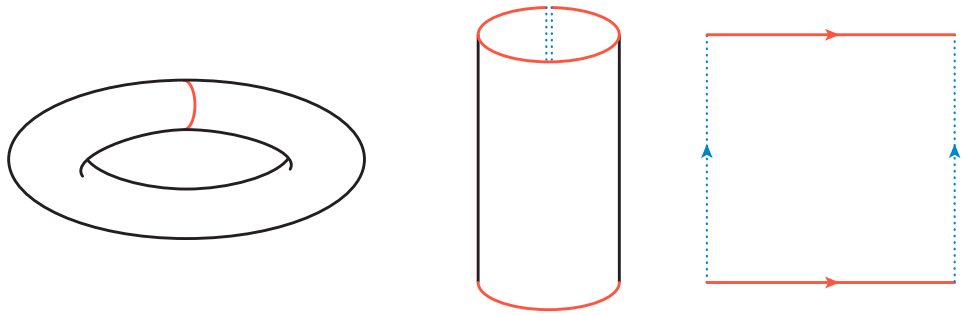


Figure 15.14: *The two-torus cut twice and flattened into a square.*

15.6 Afterword

The fact that there are, in a natural sense, only two non-planar graphs is one of the main reasons we study the topic. But this turns out to be the easiest case of an even more amazing family of results that I'll discuss briefly. These other theorems have to do with drawing graphs on arbitrary surfaces (spheres, tori ...)—it's common to refer to this as *embedding* the graph in the surface—and the process uses curves similar to those discussed in Section 15.1.1, except that now we want, for example, curves $\gamma : [0, 1] \to S_2$, where S_2 is the two-sphere, the surface of a three-dimensional unit ball.

Embedding a graph in the sphere turns out to be the same as embedding it in the plane: you can imagine drawing the planar diagram on a large, thin, stretchy sheet and then smoothing it onto a big ball in such a way that the diagram lies in the northern hemisphere while the edges of the sheet are all drawn together in a bunch at the south pole. Similarly, if we had a graph embedded in the sphere we could get a planar diagram for it by punching a hole in the sphere. Thus a graph can be embedded in the sphere unless it contains—in the sense of Kuratowski's Theorem—a copy of K_5 or $K_{3,3}$. For this reason, these two graphs are called *topological obstructions* to embedding a graph in the plane or sphere. They are also sometimes referred to as *forbidden subgraphs*.

But if we now consider the torus, the situation for K_5 and $K_{3,3}$ is different. To make drawings, I'll use a standard representation of the torus as a square: you should imagine this square to have been peeled off a more familiar torus-as-a-doughnut, as illustrated in Figure 15.14. Figure 15.15 then shows embeddings of K_5 ad $K_{3,3}$ in the torus—these are analogous to planar diagrams in that the arcs representing the edges don't intersect except at their endpoints.

There are, however, graphs that one cannot embed in the torus and there is even an analog of Kuratowski's Theorem that says that there are finitely many forbidden subgraphs and that all non-toroidal[5] graphs include at least one of them. In fact, something even more spectacular is true: early in an epic series[6] of papers,

[5] By analogy with the term non-planar, a graph is said to be *non-toroidal* if it cannot be embedded in the torus.

[6] The titles all begin with the words "Graph Minors". The series began in 1983 with "Graph Minors. I. Excluding a forest" (DOI: 10.1016/0095-8956(83)90079-5) and seems

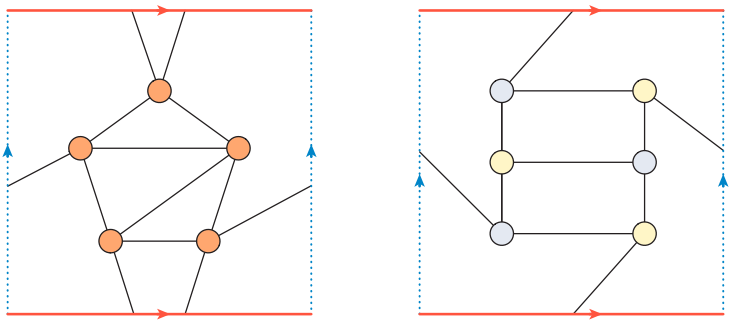


Figure 15.15: *Embeddings of K_5 (left) and $K_{3,3}$ (right) in the torus. Edges that run off the top edge of the square return on the bottom, while those that run off the right edge come back on the left.*

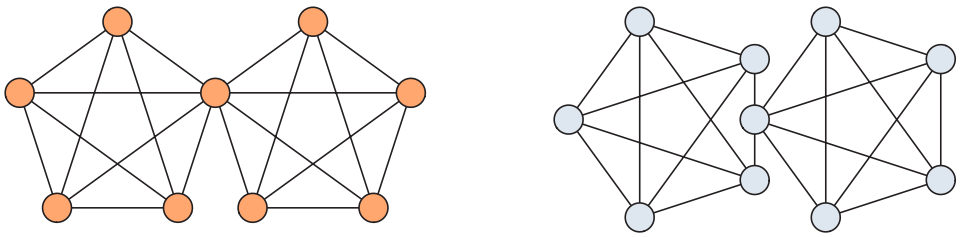


Figure 15.16: *Neither of these graphs can be embedded in the two-torus. These examples come from Andrei Gargarin, Wendy Myrvold and John Chambers (2009), The obstructions for toroidal graphs with no $K_{3,3}$'s,* Discrete Mathematics, ***309****(11):3625–3631. DOI: 10.1016/j.disc.2007.12.075*

Neil Robertson and Paul D. Seymour proved that *every* surface (the sphere, the torus, the torus with two holes...) has a Kuratowski-like theorem with a finite list of forbidden subgraphs: two of those for the torus are shown in Figure 15.16. One shouldn't, however, draw too much comfort from the word "finite". In her recent MSc thesis[7] Ms. Jennifer Woodcock developed a new algorithm for embedding graphs in the torus and tested it against a database that, although known to be incomplete, includes 239,451 forbidden subgraphs.

to have concluded with "Graph Minors. XXIII. Nash-Williams' immersion conjecture" in 2010 (DOI: 10.1016/j.jctb.2009.07.003). The result about embedding graphs in surfaces appeared in 1990 in "Graph Minors. VIII. A Kuratowski theorem for general surfaces" (DOI: 10.1016/0095-8956(90)90121-F).

[7] Ms. Woodcock's thesis, *A Faster Algorithm for Torus Embedding*, is lovely and is the source of much of the material in this section.