Chapter 4

Small world graphs

4.1 Analysis of graph algorithms

The order of growth for a graph algorithm is usually expressed as a function of $|V|$, the number of vertices, and $|E|$, the number of edges.

The order of growth for BFS is $O(|V| + |E|)$, which is a convenient way to say that the run time grows in proportion to either $|V|$ or $|E|$, whichever is "bigger."

To see why, think about these four operations:

Adding a vertex to the queue: this happens once for each vertex, so the total cost is in $O(|V|)$.

Removing a vertex from the queue: this happens once for each vertex, so the total cost is in $O(|V|)$.

Marking a vertex "visited": this happens once for each vertex, so the total cost is in $O(|V|)$.

Checking whether a vertex is marked: this happens once each edge, so the total cost is in $O(|E|)$.

Adding them up, we get $O(|V| + |E|)$. If we know the relationship between $|V|$ and $|E|$, we can simplify this expression. For example, in a regular graph, the number of edges is in $O(|V|)$ so BFS is linear in $|V|$. In a complete graph, the number of edges is in $O(|V|^2)$ so BFS is quadratic in $|V|$.

Of course, this analysis is based on the assumption that all four operations—adding and removing vertices, marking and checking marks—are constant time.

Marking vertices is easy. You can add an attribute to the `Vertex` objects or put the marked ones in a set and use the `in` operator.

But making a first-in-first-out (FIFO) queue that can add and remove vertices in constant time turns out to be non-trivial.

4.2 FIFO implementation

A FIFO is a data structure that provides the following operations:

append: Add a new item to the end of the queue.

pop: Remove and return the item at the front of the queue.

There are several good implementations of this data structure. One is the **doubly-linked list**, which you can read about at `http://en.wikipedia.org/wiki/Doubly-linked_list`. Another is a circular buffer, which you can read about at `http://en.wikipedia.org/wiki/Circular_buffer`.

Exercise 4.1. *Write an implementation of a FIFO using either a doubly-linked list or a circular buffer.*

Yet another possibility is to use a Python dictionary and two indices: `nextin` keeps track of the back of the queue; `nextout` keeps track of the front. The dictionary maps from integer indices to values.

Here is an implementation based on Raymond Hettinger's recipe at `http://code.activestate.com/recipes/68436/`.

```
class DictFifo(object):

    def __init__(self):
        self.nextin = 0
        self.nextout = 0
        self.data = {}

    def append(self, value):
        self.data[self.nextin] = value
        self.nextin += 1

    def pop(self, n=-1):
        value = self.data.pop(self.nextout)
        self.nextout += 1
        return value
```

`append` stores the new item and increments `nextin`; both operations are constant time.

`pop` removes the last item and increments `nextout`. Again, both operations are constant time.

As yet another alternative, the Python `collections` module provides an object called a `deque`, which stands for "double-ended queue". It is supposed to be pronounced "deck," but many people say "deek." A Python deque can be adapted to implement a FIFO.

You can read about deques at `http://en.wikipedia.org/wiki/Deque` and get the details of the Python implementation at `docs.python.org/lib/deque-objects.html`.

Exercise 4.2. *The following implementation of a BFS contains two performance errors. What are they? What is the actual order of growth for this algorithm?*

```
def bfs(top_node, visit):
    """Breadth-first search on a graph, starting at top_node."""
    visited = set()
    queue = [top_node]
    while len(queue):
        curr_node = queue.pop(0)    # Dequeue
        visit(curr_node)            # Visit the node
        visited.add(curr_node)

        # Enqueue non-visited and non-enqueued children
        queue.extend(c for c in curr_node.children
                     if c not in visited and c not in queue)
```

Test this code with a range of graph sizes and check your analysis. Then use a FIFO implementation to fix the errors and confirm that your algorithm is linear.

4.3 Stanley Milgram

Stanley Milgram was an American social psychologist who conducted two of the most famous experiments in social science, the Milgram experiment, which studied people's obedience to authority (`http://en.wikipedia.org/wiki/Milgram_experiment`) and the Small World Experiment, which studied the structure of social networks (`http://en.wikipedia.org/wiki/Small_world_phenomenon`).

In the Small World Experiment, Milgram sent a package to several randomly-chosen people in Wichita, Kansas, with instructions asking them to forward an enclosed letter to a target person, identified by name and occupation, in Sharon, Massachusetts (which is the town near Boston where I grew up). The subjects were told that they could mail the letter directly to the target person only if they knew him personally; otherwise they were instructed to send it, and the same instructions, to a relative or friend they thought would be more likely to know the target person.

Many of the letters were never delivered, but for the ones that were the average path length—the number of times the letters were forwarded—was about six. This result was taken to confirm previous observations (and speculations) that the typical distance between any two people in a social network is about "six degrees of separation."

This conclusion is surprising because most people expect social networks to be localized—people tend to live near their friends—and in a graph with local connections, path lengths tend to increase in proportion to geographical distance. For example, most of my friends live nearby, so I would guess that the average distance between nodes in a social network is about 50 miles. Wichita is about 1600 miles from Boston, so if Milgram's letters traversed typical links in the social network, they should have taken 32 hops, not six.

4.4 Watts and Strogatz

In 1998 Duncan Watts and Steven Strogatz published a paper in *Nature*, "Collective dynamics of 'small-world' networks," that proposed an explanation for the small world phe-

nomenon. You can download it from `http://www.nature.com/nature/journal/v393/n6684/abs/393440a0.html`.

Watts and Strogatz started with two kinds of graph that were well understood: random graphs and regular graphs. They looked at two properties of these graphs, clustering and path length.

Clustering is a measure of the "cliquishness" of the graph. In a graph, a **clique** is a subset of nodes that are all connected to each other; in a social network, a clique is a set of friends who all know each other. Watts and Strogatz defined a clustering coefficient that quantifies the likelihood that two nodes that are connected to the same node are also connected to each other.

Path length is a measure of the average distance between two nodes, which corresponds to the degrees of separation in a social network.

Their initial result was what you might expect: regular graphs have high clustering and high path lengths; random graphs with the same size tend to have low clustering and low path lengths. So neither of these is a good model of social networks, which seem to combine high clustering with short path lengths.

Their goal was to create a **generative model** of a social network. A generative model tries to explain a phenomenon by modeling the process that builds or leads to the phenomenon. In this case Watts and Strogatz proposed a process for building small-world graphs:

1. Start with a regular graph with n nodes and degree k. Watts and Strogatz start with a ring lattice, which is a kind of regular graph. You could replicate their experiment or try instead a graph that is regular but not a ring lattice.

2. Choose a subset of the edges in the graph and "rewire" them by replacing them with random edges. Again, you could replicate the procedure described in the paper or experiment with alternatives.

The proportion of edges that are rewired is a parameter, p, that controls how random the graph is. With $p = 0$, the graph is regular; with $p = 1$ it is random.

Watts and Strogatz found that small values of p yield graphs with high clustering, like a regular graph, and low path lengths, like a random graph.

Exercise 4.3. *Read the Watts and Strogatz paper and answer the following questions:*

1. *What process do Watts and Strogatz use to rewire their graphs?*

2. *What is the definition of the clustering coefficient $C(p)$?*

3. *What is the definition of the average path length $L(p)$?*

4. *What real-world graphs did Watts and Strogatz look at? What evidence do they present that these graphs have the same structure as the graphs generated by their model?*

Exercise 4.4. *Create a file named* `SmallWorldGraph.py` *and define a class named* `SmallWorldGraph` *that inherits from* `RandomGraph`.

If you did Exercise 2.4 you can use your own `RandomGraph.py`*; otherwise you can download mine from* `thinkcomplex.com/RandomGraph.py`.

1. *Write a method called* `rewire` *that takes a probability* `p` *as a parameter and, starting with a regular graph, rewires the graph using Watts and Strogatz's algorithm.*

2. *Write a method called* `clustering_coefficient` *that computes and returns the clustering coefficient as defined in the paper.*

3. *Make a graph that replicates the line marked $C(p)/C(0)$ in Figure 2 of the paper. In other words, confirm that the clustering coefficient drops off slowly for small values of p.*

Before we can replicate the other line, we have to learn about shortest path algorithms.

4.5 Dijkstra

Edsger W. Dijkstra was a Dutch computer scientist who invented an efficient shortest-path algorithm (see `http://en.wikipedia.org/wiki/Dijkstra's_algorithm`). He also invented the semaphore, which is a data structure used to coordinate programs that communicate with each other (see `http://en.wikipedia.org/wiki/Semaphore_(programming)` and Downey, *The Little Book of Semaphores*).

Dijkstra is famous (and notorious) as the author of a series of essays on computer science. Some, like "A Case against the GO TO Statement," have had a profound effect on programming practice. Others, like "On the Cruelty of Really Teaching Computing Science," are entertaining in their cantankerousness, but less effective.

Dijkstra's algorithm solves the "single source shortest path problem," which means that it finds the minimum distance from a given "source" node to every other node in the graph (or at least every connected node).

We start with a simplified version of the algorithm that considers all edges the same length. The more general version works with any non-negative edge lengths.

The simplified version is similar to the breadth-first search in Section 2.4 except that instead of marking visited nodes, we label them with their distance from the source. Initially all nodes are labeled with an infinite distance. Like a breadth-first search, Dijkstra's algorithm uses a queue of discovered unvisited nodes.

1. Give the source node distance 0 and add it to the queue. Give the other nodes infinite distance.

2. Remove a vertex from the queue and assign its distance to d. Find the vertices it is connected to. For each connected vertex with infinite distance, replace the distance with $d + 1$ and add it to the queue.

3. If the queue is not empty, go back to Step 2.

The first time you execute Step 2, the only node in the queue has distance 0. The second time, the queue contains all nodes with distance 1. Once those nodes are processed, the queue contains all nodes with distance 2, and so on.

So when a node is discovered for the first time, it is labeled with the distance $d + 1$, which is the shortest path to that node. It is not possible that you will discover a shorter path

later, because if there were a shorter path, you would have discovered it sooner. That is not a proof of the correctness of the algorithm, but it sketches the structure of the proof by contradiction.

In the more general case, where the edges have different lengths, it is possible to discover a shorter path after you have discovered a longer path, so a little more work is needed.

Exercise 4.5. *Write an implementation of Dijkstra's algorithm and use it to compute the average path length of a SmallWorldGraph.*

Make a graph that replicates the line marked $L(p)/L(0)$ in Figure 2 of the Watts and Strogatz paper. Confirm that the average path length drops off quickly for small values of p. What is the range of values for p that yield graphs with high clustering and low path lengths?

Exercise 4.6. *A natural question about the Watts and Strogatz paper is whether the small world phenomenon is specific to their generative model or whether other similar models yield the same qualitative result (high clustering and low path lengths).*

To answer this question, choose a variation of the Watts and Strogatz model and replicate their Figure 2. There are two kinds of variation you might consider:

- *Instead of starting with a regular graph, start with another graph with high clustering. One option is a locally-connected graph where vertices are placed at random locations in the plane and each vertex is connected to its nearest k neighbors.*
- *Experiment with different kinds of rewiring.*

If a range of similar models yield similar behavior, we say that the results of the paper are **robust**.

Exercise 4.7. *To compute the average path length in a SmallWorldGraph, you probably ran Dijkstra's single-source shortest path algorithm for each node in the graph. In effect, you solved the "all-pairs shortest path" problem, which finds the shortest path between all pairs of nodes.*

1. *Find an algorithm for the all-pairs shortest path problem and implement it. Compare the run time with your "all-source Dijkstra" algorithm.*
2. *Which algorithm gives better order-of-growth run time as a function of the number of vertices and edges? Why do you think Dijkstra's algorithm does better than the order-of-growth analysis suggests?*

4.6 What kind of explanation is *that*?

If you ask me why planetary orbits are elliptical, I might start by modeling a planet and a star as point masses; I would look up the law of universal gravitation at `http://en.wikipedia.org/wiki/Newton's_law_of_universal_gravitation` and use it to write a differential equation for the motion of the planet. Then I would either derive the orbit equation or, more likely, look it up at `http://en.wikipedia.org/wiki/Orbit_equation`. With a little algebra, I could derive the conditions that yield an elliptical orbit. Then I would argue that the objects we consider planets satisfy these conditions.

People, or at least scientists, are generally satisfied with this kind of explanation. One of the reasons for its appeal is that the assumptions and approximations in the model seem

reasonable. Planets and stars are not really point masses, but the distances between them are so big that their actual sizes are negligible. Planets in the same solar system can affect each others' orbits, but the effect is usually small. And we ignore relativistic effects, again on the assumption that they are small.

This explanation is also appealing because it is equation-based. We can express the orbit equation in a closed form, which means that we can compute orbits efficiently. It also means that we can derive general expressions for the orbital velocity, orbital period, and other quantities.

Finally, I think this kind of explanation is appealing because it has the form of a mathematical proof. It starts from a set of axioms and derives the result by logic and analysis. But it is important to remember that the proof pertains to the model and not the real world. That is, we can prove that an idealized model of a planet yields an elliptic orbit, but we can't prove that the model pertains to actual planets (in fact, it does not).

By comparison, Watts and Strogatz's explanation of the small world phenomenon may seem less satisfying. First, the model is more abstract, which is to say less realistic. Second, the results are generated by simulation, not by mathematical analysis. Finally, the results seem less like a proof and more like an example.

Many of the models in this book are like the Watts and Strogatz model: abstract, simulation-based and (at least superficially) less formal than conventional mathematical models. One of the goals of this book it to consider the questions these models raise:

- What kind of work can these models do: are they predictive, or explanatory, or both?
- Are the explanations these models offer less satisfying than explanations based on more traditional models? Why?
- How should we characterize the differences between these and more conventional models? Are they different in kind or only in degree?

Over the course of the book I will offer my answers to these questions, but they are tentative and sometimes speculative. I encourage you to consider them skeptically and reach your own conclusions.

Chapter 5

Scale-free networks

5.1 Zipf's Law

Zipf's law describes a relationship between the frequencies and ranks of words in natural languages; see `http://en.wikipedia.org/wiki/Zipf's_law`. The "frequency" of a word is the number of times it appears in a body of work. The "rank" of a word is its position in a list of words sorted by frequency: the most common word has rank 1, the second most common has rank 2, etc.

Specifically, Zipf's Law predicts that the frequency, f, of the word with rank r is:

$$f = cr^{-s}$$

where s and c are parameters that depend on the language and the text.

If you take the logarithm of both sides of this equation, you get:

$$\log f = \log c - s \log r$$

So if you plot $\log f$ versus $\log r$, you should get a straight line with slope $-s$ and intercept $\log c$.

Exercise 5.1. *Write a program that reads a text from a file, counts word frequencies, and prints one line for each word, in descending order of frequency. You can test it by downloading an out-of-copyright book in plain text format from* `gutenberg.net`. *You might want to remove punctuation from the words.*

If you need some help getting started, you can download `thinkcomplex.com/Pmf.py`, *which provides an object named* `Hist` *that maps from value to frequencies.*

Plot the results and check whether they form a straight line. For plotting suggestions, see Section 3.6. Can you estimate the value of s?

You can download my solution from `thinkcomplex.com/Zipf.py`

5.2 Cumulative distributions

A distribution is a statistical description of a set of values. For example, if you collect the population of every city and town in the U.S., you would have a set of about 14,000 integers.

The simplest description of this set is a list of numbers, which would be complete but not very informative. A more concise description is a statistical summary like the mean and variation, but that is not a complete description because there are many sets of values with the same summary statistics.

One alternative is a histogram, which divides the range of possible values into "bins" and counts the number of values that fall in each bin. Histograms are common, so they are easy to understand, but it is tricky to get the bin size right. If the bins are too small, the number of values in each bin is also small, so the histogram doesn't give much insight. If the bins are too large, they lump together a wide range of values, which obscures details that might be important.

A better alternative is a **cumulative distribution function** (CDF), which maps from a value, x, to the fraction of values less than or equal to x. If you choose a value at random, $CDF(x)$ is the probability that the value you get is less than or equal to x.

For a list of n values, the simplest way to compute CDF is to sort the values. Then the CDF of the ith value (starting from 1) is i/n.

I have written a class called `Cdf` that provides functions for creating and manipulating CDFs. You can download it from `thinkcomplex.com/Cdf.py`.

As an example, we'll compute the CDF for the values {1,2,2,4,5}:

```
import Cdf
cdf = Cdf.MakeCdfFromList([1,2,2,4,5])
```

`MakeCdfFromList` can take any sequence or iterator. Once you have the `Cdf`, you can find the probability, $CDF(x)$, for a given value:

```
prob = cdf.Prob(2)
```

The result is 0.6, because 3/5 of the values are less than or equal to 2. You can also compute the value for a given probability:

```
value = cdf.Value(0.5)
```

The value with probability 0.5 is the median, which in this example is 2.

To plot the `Cdf`, you can use `Render`, which returns a list of value-probability pairs.

```
    xs, ps = cdf.Render()
    for x, p in zip(xs, ps):
        print x, p
```

The result is:

```
1 0.0
1 0.2
2 0.2
2 0.6
```

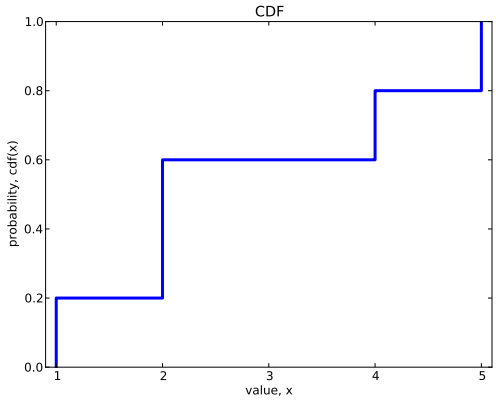

Figure 5.1: CDF of the values {1,2,2,4,5}.

```
4 0.6
4 0.8
5 0.8
5 1.0
```

Each value appears twice. That way when we plot the CDF, we get a stair-step pattern.

```
import matplotlib.pyplot as pyplot

    xs, ps = cdf.Render()

    pyplot.plot(xs, ps, linewidth=3)
    pyplot.axis([0.9, 5.1, 0, 1])
    pyplot.title('CDF')
    pyplot.xlabel('value, x')
    pyplot.ylabel('probability, CDF(x)')
    pyplot.show()
```

Figure 5.1 shows the cumulative distribution function (CDF), for the values $(1, 2, 2, 4, 5)$.

I drew vertical lines at each of the values, which is not mathematically correct. To be more rigorous, I should draw a discontinuous function.

Exercise 5.2. *Read the code in* `Cdf.py`*. What is the order of growth for* `MakeCdfFromList` *and the methods* `Prob` *and* `Value`*?*

5.3 Continuous distributions

The distributions we have seen so far are sometimes called **empirical distributions** because they are based on a dataset that comes from some kind of empirical observation.

An alternative is a **continuous distribution**, which is characterized by a CDF that is a continuous function. Some of these distributions, like the Gaussian or normal distribution,

are well known, at least to people who have studied statistics. Many real world phenomena can be approximated by continuous distributions, which is why they are useful.

For example, if you observe a mass of radioactive material with an instrument that can detect decay events, the distribution of times between events will most likely fit an exponential distribution. The same is true for any series where an event is equally likely at any time.

The CDF of the exponential distribution is:

$$CDF(x) = 1 - e^{-\lambda x}$$

The parameter, λ, determines the mean and variance of the distribution. This equation can be used to derive a simple visual test for whether a dataset can be well approximated by an exponential distribution. All you have to do is plot the **complementary distribution** on a log-y scale.

The complementary distribution (CCDF) is just $1 - CDF(x)$; if you plot the complementary distribution of a dataset that you think is exponential, you expect to see a function like:

$$y = 1 - CDF(x) \sim e^{-\lambda x}$$

If you take the log of both sides of this equation, you get:

$$\log y \sim -\lambda x$$

So on a log-y scale the CCDF should look like a straight line with slope $-\lambda$.

Exercise 5.3. *Write a function called* `plot_ccdf` *that takes a list of values and the corresponding list of probabilities and plots the CCDF on a log-y scale.*

To test your function, use `expovariate` *from the* `random` *module to generate 100 values from an exponential distribution. Plot the CCDF on a log-y scale and see if it falls on a straight line.*

5.4 Pareto distributions

The Pareto distribution is named after the economist Vilfredo Pareto, who used it to describe the distribution of wealth; see `http://en.wikipedia.org/wiki/Pareto_distribution`. Since then, people have used it to describe phenomena in the natural and social sciences including sizes of cities and towns, sand particles and meteorites, forest fires and earthquakes.

The Pareto distribution is characterized by a CDF with the following form:

$$CDF(x) = 1 - \left(\frac{x}{x_m}\right)^{-\alpha}$$

The parameters x_m and α determine the location and shape of the distribution. x_m is the minimum possible quantity.

Values from a Pareto distribution often have these properties:

Long tail: Pareto distributions contain many small values and a few very large ones.

80/20 rule: The large values in a Pareto distribution are so large that they make up a disproportionate share of the total. In the context of wealth, the 80/20 rule says that 20% of the people own 80% of the wealth.

Scale free: Short-tailed distributions are centered around a typical size, which is called a "scale." For example, the great majority of adult humans are between 100 and 200 cm in height, so we could say that the scale of human height is a few hundred centimeters. But for long-tailed distributions, there is no similar range (bounded by a factor of two) that contains the bulk of the distribution. So we say that these distributions are "scale-free."

To get a sense of the difference between the Pareto and Gaussian distributions, imagine what the world would be like if the distribution of human height were Pareto.

In Pareto World, the shortest person is 100 cm, and the median is 150 cm, so that part of the distribution is not very different from ours.

But if you generate 6 billion values from this distribution distribution, the tallest person might be 100 km—that's what it means to be scale-free!

There is a simple visual test that indicates whether an empirical distribution is well-characterized by a Pareto distribution: on a log-log scale, the CCDF looks like a straight line. The derivation is similar to what we saw in the previous section.

The equation for the CCDF is:

$$y = 1 - CDF(x) \sim \left(\frac{x}{x_m}\right)^{-\alpha}$$

Taking the log of both sides yields:

$$\log y \sim -\alpha(\log x - \log x_m)$$

So if you plot $\log y$ versus $\log x$, it should look like a straight line with slope $-\alpha$ and intercept $\alpha \log x_m$.

Exercise 5.4. *Write a version of* `plot_ccdf` *that plots the complementary CCDF on a log-log scale.*

To test your function, use `paretovariate` *from the* `random` *module to generate 100 values from a Pareto distribution. Plot the CCDF on a log-y scale and see if it falls on a straight line. What happens to the curve as you increase the number of values?*

Exercise 5.5. *The distribution of populations for cities and towns has been proposed as an example of a real-world phenomenon that can be described with a Pareto distribution.*

The U.S. Census Bureau publishes data on the population of every incorporated city and town in the United States. I wrote a small program that downloads this data and converts it into a convenient form. You can download it from `thinkcomplex.com/populations.py`.

Read over the program to make sure you know what it does and then write a program that computes and plots the distribution of populations for the 14593 cities and towns in the dataset.

Plot the CDF on linear and log-x scales so you can get a sense of the shape of the distribution. Then plot the CCDF on a log-log scale to see if it has the characteristic shape of a Pareto distribution.

What conclusion do you draw about the distribution of sizes for cities and towns?

5.5 Barabási and Albert

In 1999 Barabási and Albert published a paper in *Science*, "Emergence of Scaling in Random Networks," that characterizes the structure (also called "topology") of several real-world networks, including graphs that represent the interconnectivity of movie actors, world-wide web (WWW) pages, and elements in the electrical power grid in the western United States. You can download the paper from `http://www.sciencemag.org/content/286/5439/509`.

They measure the degree (number of connections) of each node and compute $P(k)$, the probability that a vertex has degree k; then they plot $P(k)$ versus k on a log-log scale. The tail of the plot fits a straight line, so they conclude that it obeys a **power law**; that is, as k gets large, $P(k)$ is asymptotic to $k^{-\gamma}$, where γ is a parameter that determines the rate of decay.

They also propose a model that generates random graphs with the same property. The essential features of the model, which distinguish it from the Erdős-Rényi model and the Watts-Strogatz model, are:

Growth: Instead of starting with a fixed number of vertices, Barabási and Albert start with a small graph and add vertices gradually.

Preferential attachment: When a new edge is created, it is more likely to connect to a vertex that already has a large number of edges. This "rich get richer" effect is characteristic of the growth patterns of some real-world networks.

Finally, they show that graphs generated by this model have a distribution of degrees that obeys a power law. Graphs that have this property are sometimes called **scale-free networks**; see `http://en.wikipedia.org/wiki/Scale-free_network`. That name can be confusing because it is the distribution of degrees that is scale-free, not the network.

In order to maximize confusion, distributions that obey the power law are sometimes called **scaling distributions** because they are invariant under a change of scale. That means that if you change the units the quantities are expressed in, the slope parameter, γ, doesn't change. You can read `http://en.wikipedia.org/wiki/Power_law` for the details, but it is not important for what we are doing here.

Exercise 5.6. *This exercise asks you to make connections between the Watts-Strogatz (WS) and Barabási-Albert (BA) models:*

1. *Read Barabási and Albert's paper and implement their algorithm for generating graphs. See if you can replicate their Figure 2(A), which shows $P(k)$ versus k for a graph with 150 000 vertices.*
2. *Use the WS model to generate the largest graph you can in a reasonable amount of time. Plot $P(k)$ versus k and see if you can characterize the tail behavior.*
3. *Use the BA model to generate a graph with about 1000 vertices and compute the characteristic length and clustering coefficient as defined in the Watts and Strogatz paper. Do scale-free networks have the characteristics of a small-world graph?*

5.6 Zipf, Pareto and power laws

At this point we have seen three phenomena that yield a straight line on a log-log plot:

Zipf plot: Frequency as a function of rank.

Pareto CCDF: The complementary CDF of a Pareto distribution.

Power law plot: A histogram of frequencies.

The similarity in these plots is not a coincidence; these visual tests are closely related.

Starting with a power-law distribution, we have:

$$P(k) \sim k^{-\gamma}$$

If we choose a random node in a scale free network, $P(k)$ is the probability that its degree equals k.

The cumulative distribution function, $CDF(k)$, is the probability that the degree is less than or equal to k, so we can get that by summation:

$$CDF(k) = \sum_{i=0}^{k} P(i)$$

For large values of k we can approximate the summation with an integral:

$$\sum_{i=0}^{k} i^{-\gamma} \sim \int_{i=0}^{k} i^{-\gamma} = \frac{1}{\gamma - 1}(1 - k^{-\gamma+1})$$

To make this a proper CDF we could normalize it so that it goes to 1 as k goes to infinity, but that's not necessary, because all we need to know is:

$$CDF(k) \sim 1 - k^{-\gamma+1}$$

Which shows that the distribution of k is asymptotic to a Pareto distribution with $\alpha = \gamma - 1$.

Similarly, if we start with a straight line on a Zipf plot, we have[1]:

$$f = cr^{-s}$$

Where f is the frequency of the word with rank r. Inverting this relationship yields:

$$r = (f/c)^{-1/s}$$

Now subtracting 1 and dividing through by the number of different words, n, we get

$$\frac{r-1}{n} = \frac{(f/c)^{-1/s}}{n} - \frac{1}{n}$$

Which is only interesting because if r is the rank of a word, then $(r-1)/n$ is the fraction of words with lower ranks, which is the fraction of words with higher frequency, which is the CCDF of the distribution of frequencies:

$$CCDF(x) = \frac{(f/c)^{-1/s}}{n} - \frac{1}{n}$$

To characterize the asymptotic behavior for large n we can ignore c and $1/n$, which yields:

$$CCDF(x) \sim f^{-1/s}$$

Which shows that if a set of words obeys Zipf's law, the distribution of their frequencies is asymptotic to a Pareto distribution with $\alpha = 1/s$.

So the three visual tests are mathematically equivalent; a dataset that passes one test will pass all three. But as a practical matter, the power law plot is noisier than the other two, because it is the derivative of the CCDF.

The Zipf and CCDF plots are more robust, but Zipf's law is only applicable to discrete data (like words), not continuous quantities. CCDF plots work with both.

For these reasons—robustness and generality—I recommend using CCDFs.

Exercise 5.7. *The Stanford Large Network Dataset Collection is a repository of datasets from a variety of networks, including social networks, communication and collaboration, Internet and road networks. See* `http://snap.stanford.edu/data/index.html`.

Download one of these datasets and explore. Is there evidence of small-world behavior? Is the network scale-free? What else can you discover?

[1]This derivation follows Adamic, "Zipf, power law and Pareto—a ranking tutorial," available at `www.hpl.hp.com/research/idl/papers/ranking/ranking.html`

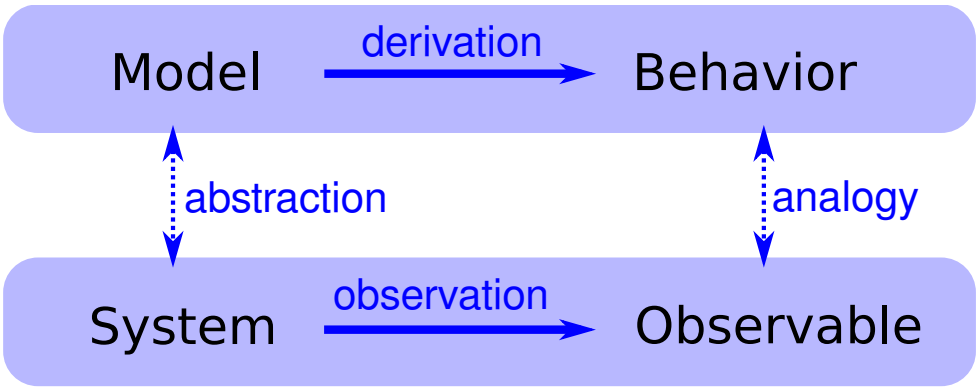

Figure 5.2: The logical structure of an explanatory model.

5.7 Explanatory models

We started the discussion of networks with Milgram's Small World Experiment, which shows that path lengths in social networks are surprisingly small; hence, "six degrees of separation".

When we see something surprising, it is natural to ask "Why?" but sometimes it's not clear what kind of answer we are looking for. One kind of answer is an **explanatory model** (see Figure 5.2). The logical structure of an explanatory model is:

1. In a system, S, we see something observable, O, that warrants explanation.
2. We construct a model, M, that is analogous to the system; that is, there is a correspondence between the elements of the model and the elements of the system.
3. By simulation or mathematical derivation, we show that the model exhibits a behavior, B, that is analogous to O.
4. We conclude that S exhibits O *because* S is similar to M, M exhibits B, and B is similar to O.

At its core, this is an argument by analogy, which says that if two things are similar in some ways, they are likely to be similar in other ways.

Argument by analogy can be useful, and explanatory models can be satisfying, but they do not constitute a proof in the mathematical sense of the word.

Remember that all models leave out, or "abstract away" details that we think are unimportant. For any system there are many possible models that include or ignore different features. And there might be models that exhibit different behaviors, B, B′ and B″, that are similar to O in different ways. In that case, which model explains O?

The small world phenomenon is an example: the Watts-Strogatz (WS) model and the Barabási-Albert (BA) model both exhibit small world behavior, but they offer different explanations:

- The WS model suggests that social networks are "small" because they include both strongly-connected clusters and "weak ties" that connect clusters (see `http://en.wikipedia.org/wiki/Mark_Granovetter`).
- The BA model suggests that social networks are small because they include nodes with high degree that act as hubs, and that hubs grow, over time, due to preferential attachment.

As is often the case in young areas of science, the problem is not that we have no explanations, but too many.

Exercise 5.8. *Are these explanations compatible; that is, can they both be right? Which do you find more satisfying as an explanation, and why?*

Is there data you could collect, or experiments you could perform, that would provide evidence in favor of one model over the other?

Choosing among competing models is the topic of Thomas Kuhn's essay, "Objectivity, Value Judgment, and Theory Choice." Kuhn was a historian of science who wrote The Structure of Scientific Revolutions *in 1962, and spent the rest of his life explaining what he meant to say.*

What criteria does Kuhn propose for choosing among competing models? Do these criteria influence your opinion about the WS and BA models? Are there other criteria you think should be considered?