Problems in Mathematical Modeling

John Burkardt
Department of Scientific Computing
Florida State University
.....
http://people.sc.fsu.edu/~jburkardt/presentations/model_problems.pdf

August 8, 2013

Abstract

Scientific research involves studying some phenomenon, forming a mathematical model, and using the model to answer interesting questions. The modeling step is crucial; whether you are looking at atoms, hurricanes or zebras, you need to find a way to express the interesting aspects of the problem in terms of numbers and formulas. We pose a sequence of simple problems, and urge you to think about how you would come up with an answer.

Contents

1 Introduction

While working on a variety of software projects, I've often stumbled across minor tasks that were interesting problems by themselves.

These little problems may also give an idea of how logic, mathematics, geometry, computer science, and plain old puzzle solving actually end up getting used in real programs.

2 The Fish on the Ceiling

One part of a science show at my local planetarium involves displaying a computer-generated model of a fish on the planetarium's hemispherical ceiling. A 3D model of a fish was generated on a PC as a simple line drawing. The information in the picture was boiled down to a sequence of lines, each of which is "drawn" on the ceiling by the planetarium's laser.

The data describing the 3D fish model consists of pairs of lines which have the form:

P x_1 y_1 z_1 *intensity*
L x_2 y_2 z_2 *intensity*

where **P** for "point" indicates that we should move to the point (x_1, y_1, z_1) and display it with the given intensity, and **L** for "line" means we should then draw a line from that point to (x_2, y_2, z_2), with the given intensity. For our case, the intensity will always be 1.0, because the laser at the planetarium is only capable of full strength or nothing.

A tiny sample of the file looks like this:

```
P   80.37       -2.115        3.682        1.000
L   80.37       -2.137        3.684        1.000
P   80.37       -2.137        3.684        1.000
L   68.08       -8.638       -5.089        1.000
P   69.20       -1.986       -8.112        1.000
L   80.37       -2.137        3.684        1.000
```

(The full fish data file, by the way, is available at http://people.sc.fsu.edu/~jburkardt/data/vla/fish_lines.vla)

Now the laser system at our planetarium takes this data file and figures out how to "draw" a corresponding image on the ceiling. That is, it takes these 3D numbers that describe a tiny fish, and somehow makes a corresponding drawing on the curved ceiling by moving the laser around.

Suppose we are given the data file, describing points and lines on a 3D fish, and we are given the dimensions of the planetarium ceiling. For each (x, y, z) point in the fish data file, can we figure out the (X, Y, Z) coordinate of the corresponding point on the dome that should be illuminated as part of the drawing?

It's important to know where the laser is located. Since the ceiling is half of a sphere, we will assume for convenience that the laser is located at the center of that sphere (this is the typical arrangement.)

To think about how this might be done, suppose that, instead of a data file, we had the fish itself, suspended five feet above the laser. If we use the laser to trace the outline of the fish, what points on the ceiling will light up?

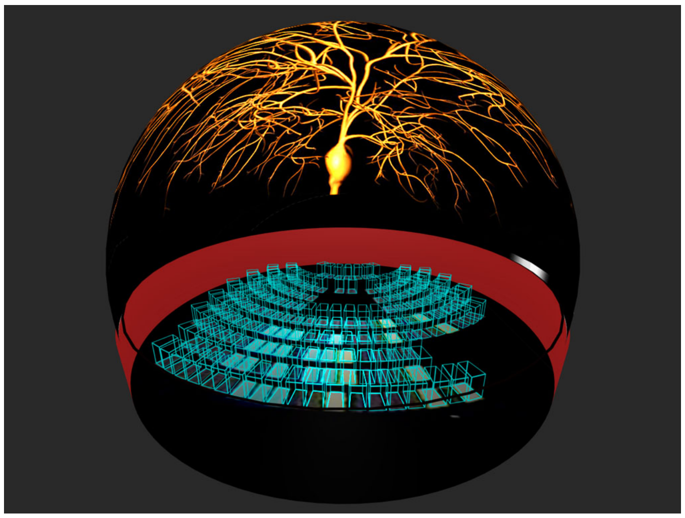


Figure 1: **The Neuron Projection**
In the same science show, a brain neuron image was projected on the ceiling.

3 If It's 3 O'Clock, This Must be Chicago

The Global Positioning System (GPS) is used to locate anybody, anywhere. To use it, you need a receiver, and a very accurate clock. There is a network of GPS satellites, and at any time, there are at least three of them within range. Each satellite sends out a signal, every 30 seconds, containing the current time, and the satellite's position. How does this tell you where you are?

Suppose you receive a signal from one satellite. Since the satellite is some distance from you, the signal will take a certain time (probably a fraction of a second) to reach you. So since your own clock is very accurate, the time on your clock when you receive the signal will differ from the time in the satellite signal, and this difference is the amount of time it took the signal to travel to you. Knowing the speed of light, (300,000 kilometers/second), you now know that you are so many kilometers away from the position (X,Y,Z) of the satellite.

Let's pretend for now that the (X,Y,Z) coordinate system used has its origin at the center of the earth, with the Z axis sticking out the North Pole, the X axis sticking through the 0 degree longitude line at the equator, and the Y axis through the 90 degree east longitude line at the equator.

Assuming you are just walking around on earth, your position is known, up to your location on the surface of a sphere. Getting a distance reading from a single satellite narrows your possible locations down to points on a circle. The signal from a second satellite gives you a second possible circle. These two circles intersect at just two points on the surface of the earth, and you can either live with this uncertainty, or get a third satellite's signal to settle your position for sure.

(If we can't assume that you're standing on the surface of the earth, then we actually need *four* satellite signals in order to determine your location. Two spheres can have a circle in common. Three spheres can have a pair of points in common. Four spheres, in general, can only have one point in common.)

To keep things simple, let's assume that each GPS satellite simply sends you a message telling you how far away it is from you. Suppose, then, that at a certain time, you get distance readings from three different satellites. Moreover, let's assume that the satellites are in orbit 12,000 miles above the surface of the earth, or 16,000 miles above the center of the earth, and that their locations form a sort of coordinate axes, with longitude and latitude $(0^o, 0^o)$, $(90^o, 0^o)$ and $(0^o, 90^o)$.

Based on the distance readings from the three satellites, can you determine your own location in (X,Y,Z) coordinates, and in longitude and latitude?

Figure 2: **GPS Satellites**
The GPS satellite network allows us to determine our position.

It might help to think about the problem in one or two dimensions first.

4 The Sneaky Flight Plan

A plane must be sent across enemy territory without being detected. The enemy has a number of detection sites in place. Each detection site has a visibility radius of 50 miles.

The navigator for the flight comes to you with a flight plane, which consists of a sequence of points marked on the map. The plane flies on straight line segments, from point to point, to complete its mission. The navigator asks you to approve this flight plan. Can you? To do so, we need to determine that the plane never gets closer than 50 miles to any detector.

One way to solve to think about solving this problem is to imagine computing the location of the plane at itevery instant in time, and measuring the distance to each detector. This might be a possible approach, but very tedious, even if we only computed the position every minute.

Another way would be to take a map, draw the flight plan, and then simply draw a strip around the flight plan, 50 miles on either side, and see if any detector falls within the strip. But a computer can't do this, can it?

Here's another approach. Pick a detection site. Compute the distance from the detection site to the line segment representing the first leg of the flight plan, and then for all the other line segments. Repeat this for all the detection sites. If you keep track of the minimum distance you encounter, that's the closest the plane gets to a detector.

In order to carry out such an approach, you need to think about some geometric questions of increasing difficulty, namely, how to compute the distance from a point to:

- another point;
- a line;
- a (finite) line segment.

Sometime soon, the enemy may develop mobile detectors, still with the 50 mile visibility radius, but able to move around the country. Suppose we manage to find out the scheduled locations of these movable detectors. How do we check whether our flight plan is safe?

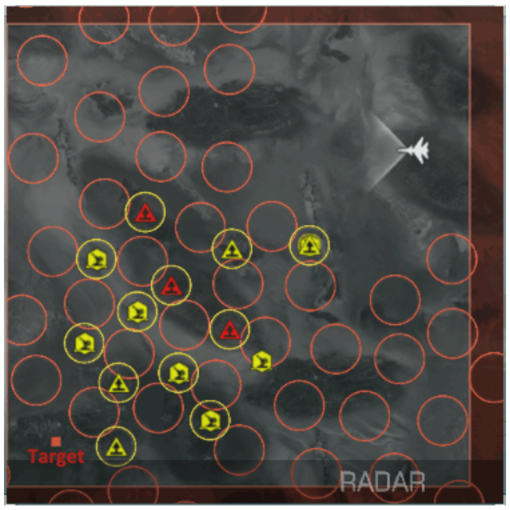


Figure 3: **The Sneaky Flight Plan**
What is the closest our flight will come to the enemy radar sites?

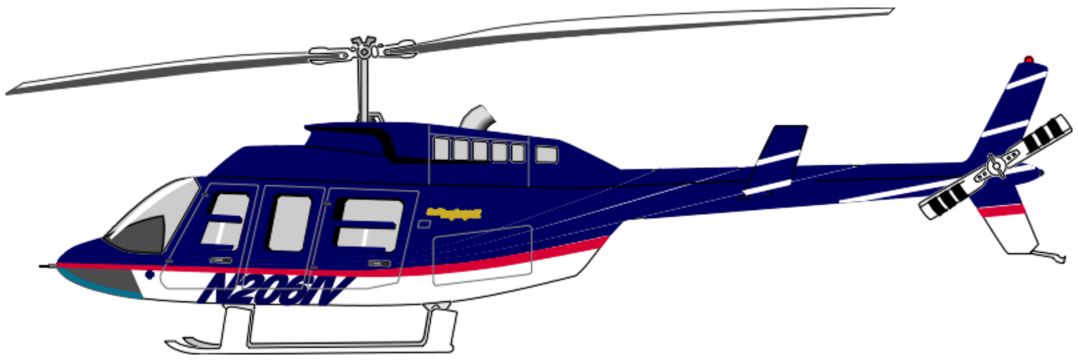


Figure 4: **The Bell Helicopter Model 206**
Will any part of the helicopter touch the nearby building?

5 The Careful Crash

One of the tests that the FAA requires a new helicopter to undergo is the "sudden loss of power" test. The helicopter takes off from the roof of a building, flies straight up 50 feet, and then cuts off power. Surprisingly, the helicopter will not plummet to the ground, although it will fall rapidly; the rotors turn during the fall, and actually provide a certain amount of lift and control. During the test, the pilot has to be able to use this minimal amount of control to guide the craft to a safe landing. Moreover, the helicopter must never come closer than 15 feet to the building at any time!

Suppose that during an execution of this test, the helicopter takes two minutes from power off to landing, and that the flight recorder registers the time and location at every second, as a set of 4 numbers (T,X,Y,Z). Thus, the record of the flight would consiste of 121 such sets.

Suppose that the building is shaped like a box, and that we have the (X,Y,Z) coordinates of its corners.

Using these two sets of data, can we determine how close the helicopter came to the building during the test? Did the heliopter pass the test?

You could start this problem by thinking about in two dimensions. Draw a curve near a square, mark a few points on the curve that represent the location of the helicopter second by second, and figure out the closest point. Could you tell a computer how to do this? The closest point will *never* be inside the square, unless something strange happens. The closest point is usually one of the corners of the square, except in special cases. When is that? What points in the plane are closest to a corner, and what points closest to a side?

To work on the full problem, you need to know some tools from geometry, including how to compute the distance from a point (X,Y,Z) to:

- another point;
- a line;
- a (finite) line segment;
- a plane;
- a (finite) rectangle;
- the surface of a box (made of 6 rectangles).

This problem is like the the "Sneaky Flight Plan", but in three dimensions. I ran across this problem while working at Bell Helicopter.

6 Building 3D Countries

A brain scanner, which can measure the density of tissue over a small region, can be used to search for tumors. A tumor has a higher density than normal brain tissue. While a brain may naturally have some regions of high density, a tumor is distinguished by the fact that it has a significantly greater volume than such naturally occurring irregularities.

The results of a brain scan have been returned to us. The scanner made measurements over a regular 64 by 64 by 26 grid, which we can think of as a collection of cubes, also called *voxels* (that is, volume elements, just as a "pixel" is a picture element). Thus, our data is a list of 64x64x26 sets of data, including a location and a density.

We concentrate on those voxels which have a high density, and ignore the others. To determine whether a tumor is formed, we need to know whether enough voxels are touching each other that they form a typical tumor mass. We will assume that a collection of 50 or more contiguous high-density voxels correspond to a tumor.

The information is all there in the data, but how can we see it? If we are lucky, we can get a graphical image of the data and examine it. But can a computer, working just from the data, give us the answer we need?

In essence, our problem assumes that we are given an array of 64 by 64 by 26 sets of data, representing the location and density of each voxel. We are told the density value above which a given voxel is "suspicious", and we are told that 50 or more contiguous suspicious voxels are probably a tumor. Our problem is to automate the process of tumor detection on a computer.

Once again, it might be easier to think about how to solve this problem by starting with a much simpler version, either in 1D or 2D. In 2D, for example, we might suppose that we have placed some checkers on a checkerboard, and now want to know how many "countries" have been formed, assuming that a country starts out as one square containing a checker, but includes occupied neighbors, and their neighbors, if any, and so on.

For a look at some representative brain data, see the Pittsburgh Supercomputing Center's Function Magnetic Resonance Imaging page at http://www.psc.edu/science/goddard.html.

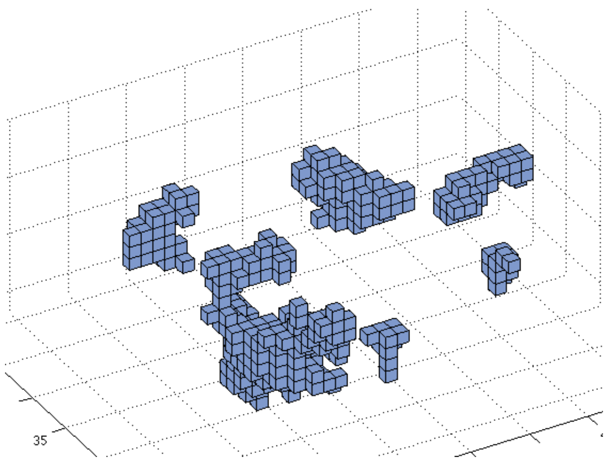


Figure 5: **A Sample 3D "Map"**
A brain scan noticed many small regions of high density.
Are many regions clumped together to form a tumor?

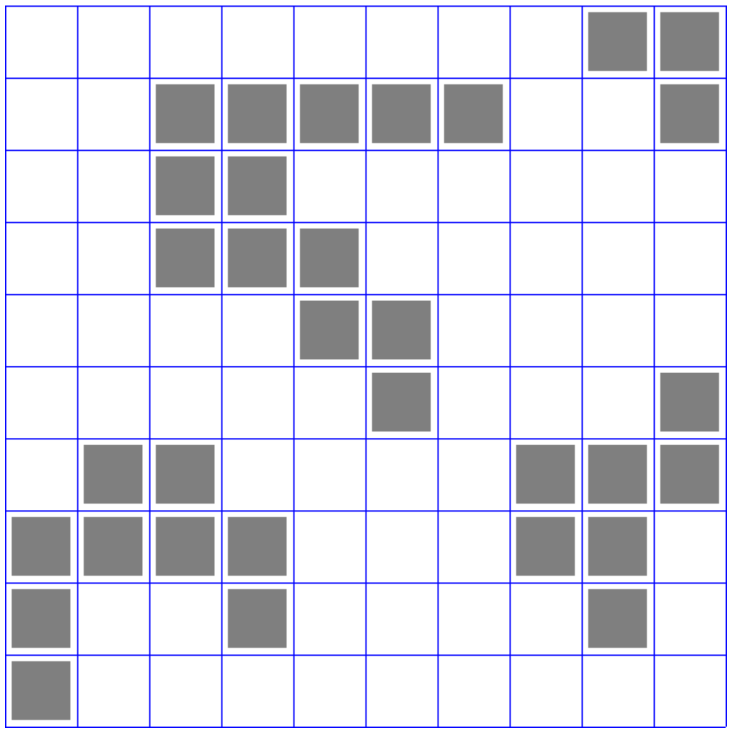


Figure 6: **A Tiny 2D "Map"**
In 2D, it may be easier to see how to determine the "countries".

Figure 7: **Lewis Carroll (Charles Lutwidge Dodgson)**
Lewis Carroll invented the game of Word Golf.

7 Lewis Carroll's Word Golf

Lewis Carroll, who suffered from insomnia his whole life, invented various puzzles to keep himself amused while trying to fall asleep. One such game he called "Word Golf" or "Word Ladders" or "Doublets". The problem was to take two words with the same number of letters, and try to transform one word into the other by changing a single letter at a time, with the requirement that each change resulted in a legal English word.

For instance, you can change *word* into *golf* in 7 steps, using the sequence:

```
WORD, WARD, HARD, CARD, CORD, COLD, GOLD, GOLF
```

Under those rules, can you change:

- HATE to LOVE?
- GOOD to EVIL?
- HORSE to WAGON?
- OUNCE to POUND?

You can't always get from one word to another. Sometimes you can do it only with the help of unusual words. For instance, after months of effort, I finally got from *ounce* to *pound* in 1967, but I had to use a few Scottish dialect words, which I only found after long desperate searched through the dictionary.

```
ounce
|
dunce
    |
dunch   (Scottish dialect for a jab with the elbow)
|
bunch
    |
bunco
   |
bunko
    |
bunks
  |
bulks
 |
balks
|
calks
   |
calls
  |
cauls
    |
cauld  (Scottish dialect for ''cold'')
 |
could
|
would
   |
wound
|
pound
```

To play this game, it helps to have an agreed list of legal words. Donald Knuth, for instance, came up with a list of 5,678 common five letter words for his Stanford Graph Base. (A copy of this list is available at http://people.sc.fsu.edu/∼jburkardt/datasets/words/knuth_words.txt).

If we agree to restrict ourselves to Knuth's list, we can ask the question, can any five letter word on the list be changed into any other five letter word? Or are some words aloof, refusing to be transformed?

To think about this problem, you could start by making a table of all the two letter words you can think of. It's easy to organize such an effort by using a piece of graph paper, marking 26 columns A through Z, and doing the same for rows. Put a check mark in every box that represents a word. This would give you a small sample database of two letter words to work on.

Of course, if we can't find a path from one word to another, that doesn't mean there isn't one. The only way to be sure would be to try absolutely every possibility. Such a task brings humans to tears, but is exactly the sort of thing that computers do very well at. But if we want to turn our problem over to a computer, we need to think more carefully about how we solve such problems in our head.

What would a computer need, in order to be able to find a sequence of transformations taking one word to another, or to determine that there was no such sequence?

How is this problem related to the problem of building 3D countries?

This is a hard, but interesting problem. One way to approach it is to imagine a graph of all the five letter words, with a link between pairs of words which are one change apart. Now start at the first word on the list, and do a "breadth-first" traversal of the graph, until you reach the target word, or have no more choices.

Software to handle this problem is available in the program **ladders**, which is part of Donald Knuth's Stanford Graph Base, described at http://www-cs-faculty.stanford.edu/∼knuth/sgb.html.

8 Six Degrees of Kevin Bacon

Stanley Milgram, a researcher at Harvard, was studying social networks. Start by considering any person, who we will consider to be at level 0; now consider all the friends of that person, whom we can designate as level 1 persons. Each of these level 1 people has friends as well; some of them may already be in the level

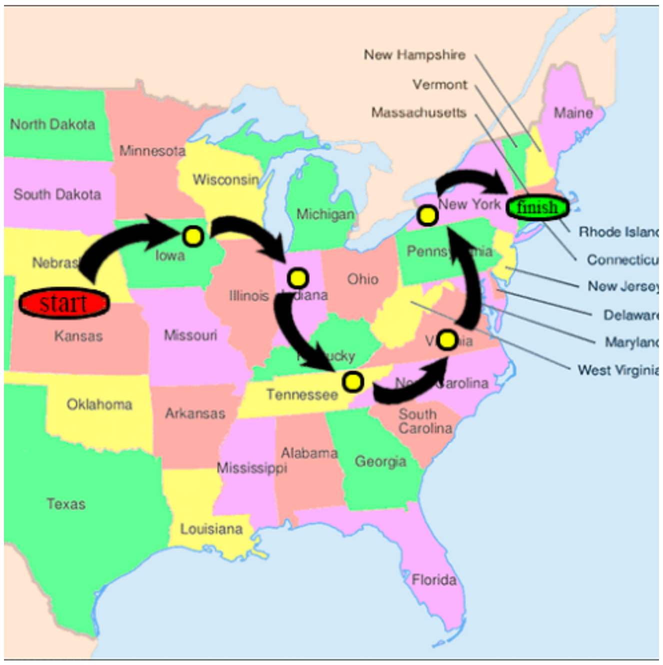


Figure 8: **The Milgram Experiment**
A sample journey of a letter from Omaha to Boston in Milgram's Small World experiment.

1 group, of course, but any new people will be assigned to level 2. The process can be repeated until there are no new additions are found.

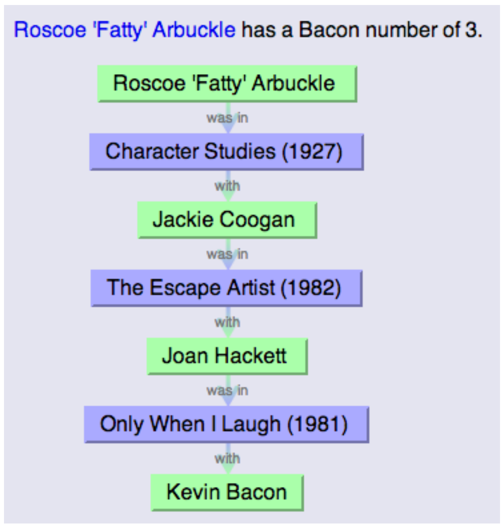
Some questions naturally arise:

- What are the chances that the process stops before the whole world is connected?
- Assuming everyone in the US is connected by friendship to everyone else, how many steps would it take, on average, to go from any one person to another?
- What is the distribution of loners (few friends) and super social people (many friends)?

Milgram tried out his ideas by choosing people in Omaha, Nebraska, or Wichita, Kansas and asking them to transmit a letter to a person in Boston, Massachusetts whom they did not know. Milgram suggested that they simply forward the letter to one of their friends whom they thought might be more closely connected to the final destinination. Surprisingly, in most cases the letter reached its destination, often in just three steps.

Are these results surprising? To answer that question, we might begin by asking how many friends the typical person has; relatives, neighbors, co-workers, schoolmates, and other people one knows by name. A value of 500 might seem reasonable. But if we take such a value as typical, then how quickly does everyone in the US become connected starting at you? Your level 0 contains 1 person, your level 1 contains 500 people. But level 2 contains 500x500 = 250,000 people. Level 3 will contain $500^3 = 125,000,000$ people which is getting near the entire US population, and level 4 is about 62 billion, which far exceeds the world population! Even though we have made some very optimistic assumptions here, it is not unreasonable to think that a very large proportion of the world is only 4 friends away from you!

Since we can't really go out and collect lists of friends, is there any way to study social networks like this on the computer? One way to do this relies on the extensive data collected about movies, in particular, the list of cast members. Suppose we consider any movie actor to be "friends" with all other actors who have appeared in the same movie.


Roscoe 'Fatty' Arbuckle has a Bacon number of 3.

Roscoe 'Fatty' Arbuckle
was in
Character Studies (1927)
with
Jackie Coogan
was in
The Escape Artist (1982)
with
Joan Hackett
was in
Only When I Laugh (1981)
with
Kevin Bacon

Figure 9: **Roscoe "Fatty" Arbuckle**
It's hard to find an actor with a Bacon number greater than 2!

Using the movie data, we can then ask questions such as how many steps it would take to get from one actor to another, using movies as links. In fact, at one time there was a trivia game called "Six Degrees of Kevin Bacon", which posed the challenge of finding such a sequence of movie links between any given named actor and Kevin Bacon.

For example, given Matthew Broderick, we can find a chain of movies linking him to Kevin Bacon as follows:

- Matthew Broderick was in *The Cable Guy* with Jim Carrey;
- Jim Carrey was in *Batman Forever* with Val Kilmer;
- Val Kilmer was in *Top Gun* with Tom Cruise;
- Tom Cruise was in *A Few Good Men* with Kevin Bacon.

Thus, Matthew Broderick can reach Kevin Bacon in just 4 steps. There might be a better solution involving fewer steps. The length of the shortest such solution is known as an actors *Bacon Number*, and you might be surprised to find out that Matthew Broderick's Bacon number is actually 1, although you might be hard-pressed to determine the movie they both appeared in.

So, although we'd really like to study real people's friendship networks, we can start with the movie link idea. Assuming we have access to a database of movie casts, we therefore ask: *Given the name of any movie actor, how can we find a sequence of movies that link this actor to Kevin Bacon?*

Can you see why this problem is related in some way to the problem of building 3D countries? Just as two actors are linked by movies, two voxels are linked if they are physically touching.

To see a program that can produce the Kevin Bacon Number for any actor, try *The Oracle of Bacon*, whose website is http://oracleofbacon.org/.

9 As the Worm Turns

A very fussy two dimensional worm decides to have his burrow built by an architect. The worm draws a sketch of his ideal burrow, which consists of 10 square rooms arranged in a straight line. "I want something

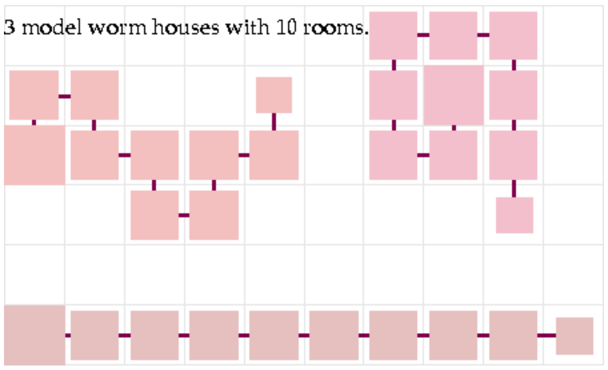


Figure 10: **Sample Worm Homes**
How many shapes are formed by 10 connected squares?

like this," he says, "with 10 rooms, no more, no less." The architect feels put off by this boring design. "Must your burrow actually be straight?", she asks. "Come to think of it, that's not important," says the worm, "but the rooms must be square, and arranged so that you can enter the front room and walk from one room to the next until you reach the last room."

The architect pulls out a checkerboard, and begins shuffling around 10 checkers. After a while, she says, "You know, I think there will be quite a *lot* of possibilities to choose from now!"

To answer questions about the possible worm homes, you have to find some way of organizing them in your mind. A good way might be to imagine that you always put the first room at the same location, and once you know where that room is, you have four choices about where to add on room number 2. After you add a room, you have to decide how the next room will be joined to the current one. Often, you will have three sides of the room to choose from, but if the house curls up against itself, some sides may no longer be available. And it's even possible that you come to a dead end and have to abandon a partial design.

If you can come up with an organized way of thinking about the house designs possible using 10 rooms, you can try to answer some questions:

- Can you determine how many possible burrow designs there will be?
- Can you determine an orderly method of listing every house design?
- Can you create a "random" house design?

There's actually some mathematics associated with this problem. A *walk* on a grid is a sequence of locations $p_1, p_2, ..., p_k$ on the grid with the property that, for any index i, p_i and p_{i+1} are neighbors. A *self avoiding walk* is a walk which has the additional property that no grid location occurs more than once in the sequence. Designing the worm's house is the same as constructing a self-avoiding walk of 10 steps.

You may have heard of *pentominoes*, which are the 12 distinct connected shapes that can be made up of 5 connected squares. Which of the pentomino shapes cannot be thought of as self-avoiding walks? Can you think of a procedure by which a computer could handle questions like this?

10 Origami With Proteins

Origami is the art of folding a sheet of paper to create an interesting shape, such as a bird or a frog. Surprisingly, proteins, which contain information about the design of real birds and frogs, also are created by folding up string of atoms into a clump that has a particular shape. It's not hard to determine the string of atoms making up a particular protein, but knowing this information is not enough to predict the shape into which the string of atoms will fold up.

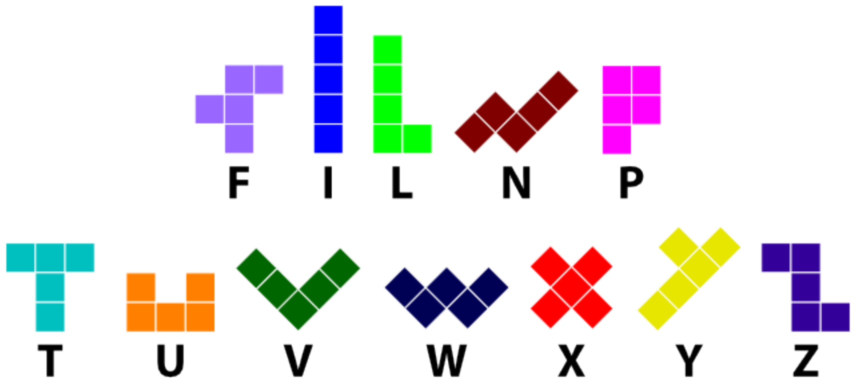


Figure 11: **Pentominoes**
Pentominoes are made of 5 squares.
Their shapes are suggested by the letters FILNP and TUVWXYZ.

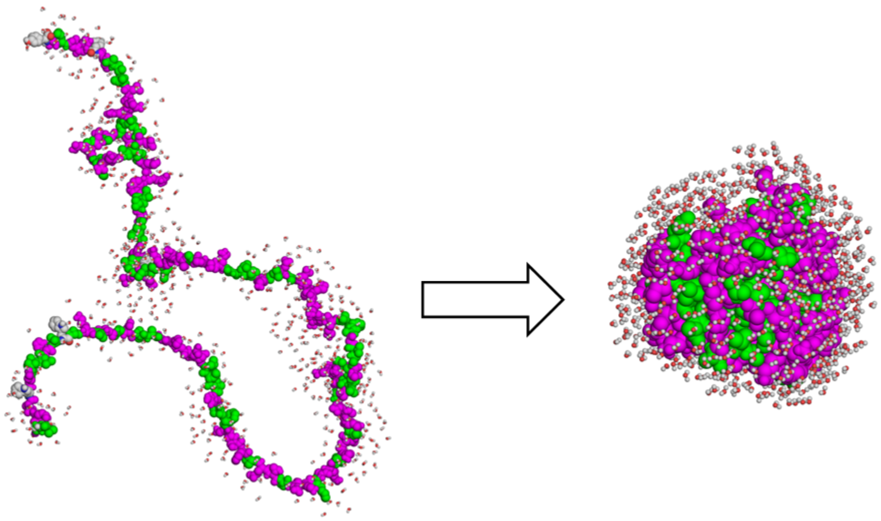


Figure 12: **Protein Folding**
A protein string folds into an energy-reducing shape.

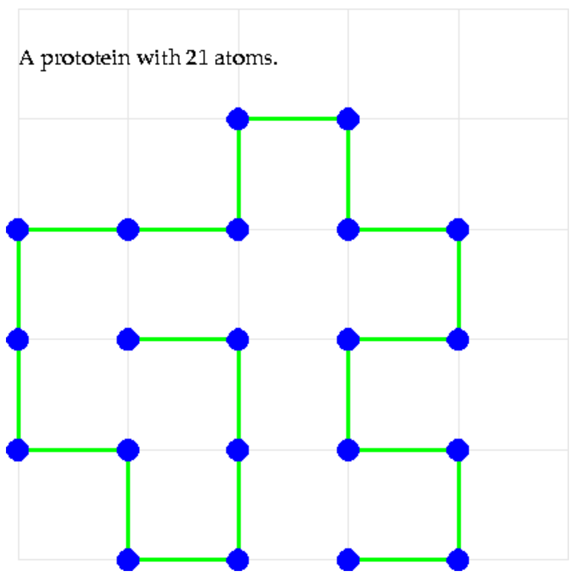


Figure 13: **Prototeins**
A prototein with 21 atoms might fold up like this.

In living things, a protein is constructed as a sort of string of beads, adding one component after another. Even while the protein is being constructed, the part that is completed will begin to fold around itself, because of the varying distribution of electrical charges on the components. The final shape of the protein is actually an important feature; occasionally, a protein will not assume the standard shape, and in consequence, will not be able to carry out its function.

Computational biologists want to be able to start from the formula for a protein, and work out on the computer how the protein will fold in response to the pattern of electrical charges.

We will look at a greatly simplified problem. Our model for a protein will be a string of numbers, 0, 1, and -1, representing neutral, positive, and negative regions. The "formula" for a protein will be its particular string of numbers. Our string proteins will live on sheets of graph paper, where they can bend and twist in very limited ways. We will assume that the first "atom" in a protein is associated with one cell of the graph paper. The next atom goes in one of the four unoccupied neighboring cells, and so on.

We can imagine the protein initially being "written" out straight on the paper. Now we expect to see some folding. Now let's suppose that the negative and positive atoms prefer to be near each other, and that the protein is willing to bend in various ways to allow this to happen. What is the configuration that is most comfortable for the protein?

In order to compare configurations, we'll have to come up with a scoring system. A crude but simple score is as follows: every charged atom that is not the neighbor of an oppositely charged atom adds 1 to the score.

Our modeling problem now assumes that we are given a formula for a prototein, that is, a sequence of values of -1, 0 and +1. We now lay out any shape we want for the prototein, where a shape is created by filling in neighboring boxes of a graph paper grid with the prototein formula values. Once we have chosen a shape, we can evaluate the prototein's score. Using this model for protein folding, our task is, *for a given prototein formula, find the shape (or shapes?) with the lowest possible score.*

For a short prototein formula, this problem is not very hard, because we can easily consider and evaluate every possible prototein shape. But we already saw in the "worm" problem, that as few as 10 squares can be arranged on a graph paper grid into an enormous number of distinct shapes. So if we start to talk about prototeins of length 20 or more, there is no chance of simply checking every possible shape.

In a case like this, we may have to be satisfied with an estimate of the lowest possible charge. We can

Figure 14: **The Tweedle Collection**
These 10 paintings are to be fairly divided by two heirs.

proceed by simply generating sample shapes at random, and keeping track of the lowest charge we encounter. To do this, we would go back to the "worm" problem. Instead of trying to construct every possible self-avoiding walk, we would need to figure out how to create a self-avoiding walk at random. If we can do this repeatedly, we can then compute the energy of each configuration and give a rough prediction of the best prototein shape.

If we want to do better than this simple procedure, we would have to figure out some way that we could take a random prototein with a good score, and try to "improve" it. From a diagram, we might see a simple adjustment to the shape that would improve the score. The trick would be to think of how to tell the computer to spot such opportunities automatically!

11 The Jealous Heirs

You are a lawyer, responsible for carrying out the late Aunt Tweedle's will, in which she specified that her art collection was to be divided, as equally as possible, between her two nephews, Dum and Dee. Aunt Tweedle specified that the artworks could not be sold.

As a first step, you've had the artworks appraised. The 10 works of art were valued, in thousands of dollars, as

```
771, 121, 281, 854, 885, 734, 486, 1003, 83, 62
```

You meet the nephews and give them a copy of the appraisals, and suggest that they agree on a fair division. Naturally, this only begins a bitter dispute. Neither nephew cares about art, but each one cares very much about money!

Nephew Dum suggests that he will take just three paintings, numbers 4, 5, and 8, while Dee can have seven. But Dee points out that Dum's art will be worth $2,742,000 while his will be only $2,538,000. Dee suggests that Dum take pictures 5, 6 and 8 instead, but this means that Dum gets $2,622,000 while Dee picks up $2,658,000, a solution which is closer, but not ideal.

As the lawyer, you have to wonder whether Aunt Tweedle actually knew whether there was a solution; and if there is one, you need to find it before the heirs start lawsuits against each other.

To test your skills in resolving wills, consider the following data, which may be a little easier to work with:

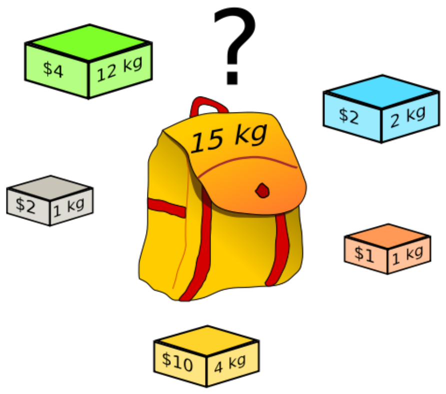


Figure 15: **The Knapsack Problem**
What's the best way to fill a knapsack?

```
  2, 10, 3, 8, 5, 7, 9, 5, 3, 2
```

Can you split these numbers into two groups, which have the same sum? Obviously, one way is to simply try every possible division of these numbers into two groups but that's probably not what you did!

The problem of breaking up a set of items of differing values into groups of equal or nearly equal value is known as the *partitioning problem*. The problem would be easy if the items (and their values) could be divided as easily as numbers can be. But, as King Solomon himself noted, some things just are not divisible in this way.

Faced with the problem of dividing the art works, a natural procedure might be to go back and forth between Dee and Dum, allowing each to pick one item at a time. If they always pick the most valuable remaining item, then Dee will get paintings 3, 4, 6, 8 and 9, with a value of $2,955,000, leaving Dum with $2,325,000. It should be clear that, in this procedure, the first person can always get more (or, at least, never less) than the second person, no matter how many items are to be divided.

Can you think of a simple modification of this procedure that would seem more fair, and would sometimes result in the second person begin able to get the most value?

Suppose that, before the will is carried out, a third heir appears unexpectedly. Surely, it's less likely that the art works can be split into three groups of equal value. How might you search for a solution? If you can't find a solution, can you find a way for the heirs to choose items as fairly as possible?

12 The Knapsack Problem

In the middle of the desert, your off-road vehicle breaks down, and you realize it's a long walk to safety. You have a knapsack which can hold 165 pounds of supplies. You brought 10 cans of food supplies, and looking at the labels, you see their weights in pounds listed as:

```
 23, 31, 29, 44, 53, 38, 63, 85, 89, 82
```

You want to pack your knapsack as close to full as possible with cans. Assuming all the food supplies are equally nutritious by weight, what's your best choice?

Thus, our problem can be thought of as follows. We are given:

- A knapsack of a known total weight capacity;
- Objects of different weights;

and we are asked to:

- Find a way to pack the knapsack with the most amount of weight.

It might be hard to see how to even begin this problem. But let's just take a few steps, at least! Here is how we might start to think:

1. First of all, any object that by itself weighs more than 165 pounds is useless, and can be discarded.
2. If any object weighs exactly 165 pounds, then we can pack that, and we're done. That's easy to check.
3. If any *two* objects weight exactly 165 pounds, we can pack them, and we're done. It's pretty easy to check all possible pairs.
4. I suppose we could check all sets of three, four, five objects, and so on, but isn't this starting to get complicated?

Indeed, there's no guarantee that we can ever completely fill the knapsack, and if we have an awful lot of little objects of different weights, it would take a great deal of time to check all the combinations. Perhaps we should start by taking a "common sense" stab at the solution. An approach like this, which is not guaranteed to get the best answer, but which seems to be based on some reasonable ideas, is sometimes called a *heuristic* solution.

Our approach is as follows:

1. Take the heaviest object that will fit in the knapsack.
2. Add the heaviest remaining object that will still keep the weight below 50.
3. Keep adding objects until you reach 50 (fat chance!) or can't add any more objects without going over the weight.

This method has the advantage that you always know what to do next on each step. This is an example of a "greedy" algorithm, because when picking a step, it always takes the biggest one possible.

There's no guarantee that this will produce the best solution. In fact, if, for example, our objects weigh 110, 100, and 65 pounds, then the greedy algorithm will grab the 110 pound weight and halt, whereas the very next two choices would give a perfect solution.

This suggests that we might start our solution using the greedy algorithm, but then try to improve it, perhaps by deciding to recompute the solution without one of the heaviest weights. Can you think of a way to carry this out?

If you look carefully, you can find versions of the knapsack problem in a surprising variety of situations. Let's suppose, for instance, that I am a thief, and have broken into a warehouse. I have to escape over a fence, so everything I take must fit into my knapsack, which can carry no more than 165 pounds. The warehouse is filled with objects, and I know their weights, but now I also happen to know their *values* in dollars.

For a thief, the goal is to maximize the value of the haul - and that means that we look first at the price tag, and only then at the weight! Thus, we might have 10 items, as before, of the given weights, but now we are very interested in their corresponding values:

Item	Weight	Value
1	23	$92
2	31	$57
3	29	$49
4	44	$68
5	53	$60
6	38	$43
7	63	$67
8	85	$84
9	89	$87
10	82	$72

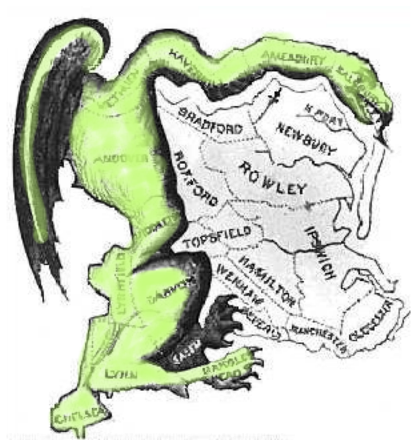


Figure 16: **The Gerrymander**
Governor Gerry Proposes a Strangely-Shaped Voting District

In other words, now we are given:

- A knapsack of a known total weight capacity;
- Objects of different weights and values

and we are asked to:

- Find a way to pack the knapsack with the most *value*, without exceeding the weight limit.

How could a thief with a 165-pound capacity knapsack maximize the value of the haul? Hint: when shopping, it pays to look at the price per pound. For each of the items, we can compute a corresponding "value per pound". This will be a strong indication of what to do next.

13 The Gerrymander

In any group of two people, the saying goes, there are at least *three* opinions. When large groups of people need to agree on a single choice, an election can be held to make the choice. If there are only two choices, and everyone votes as a group, it's easy to determine the choice that satisfies the majority of the voters. But in practical situations, there are many choices, and a complicated process to follow. It has been known for ages that an election process can often be arranged to prefer, or sometimes even to guarantee, a particular election result.

Consider the case of representative democracy, where a country or province is divided into smaller regions, each of which elects a representative to a common council. One might think it a desirable property of this system that the composition of the council be similar to that of the people who elected them. On the other hand, if you were a politician, this might *not* be how you think!

In that case, you might hope to pack more of your party onto the council. One way to do this would occur if you were allowed to decide the boundaries of the districts. We might suppose that each district has to be connected, and perhaps needs to have roughly the same number of inhabitants as all the others. But if we have any idea of the voting preferences of the inhabitants, we can surely find a way to cleverly draw the boundaries of the districts to help our party.

This can be done by trying to all your own party members live in districts where they form a respectable majority; the members of the opposing party live either as minorities in such a mixed district, or else in "pure" districts where there are none of your party. If things work out for you, every member of your party has a vote that is worth double that of the other party.

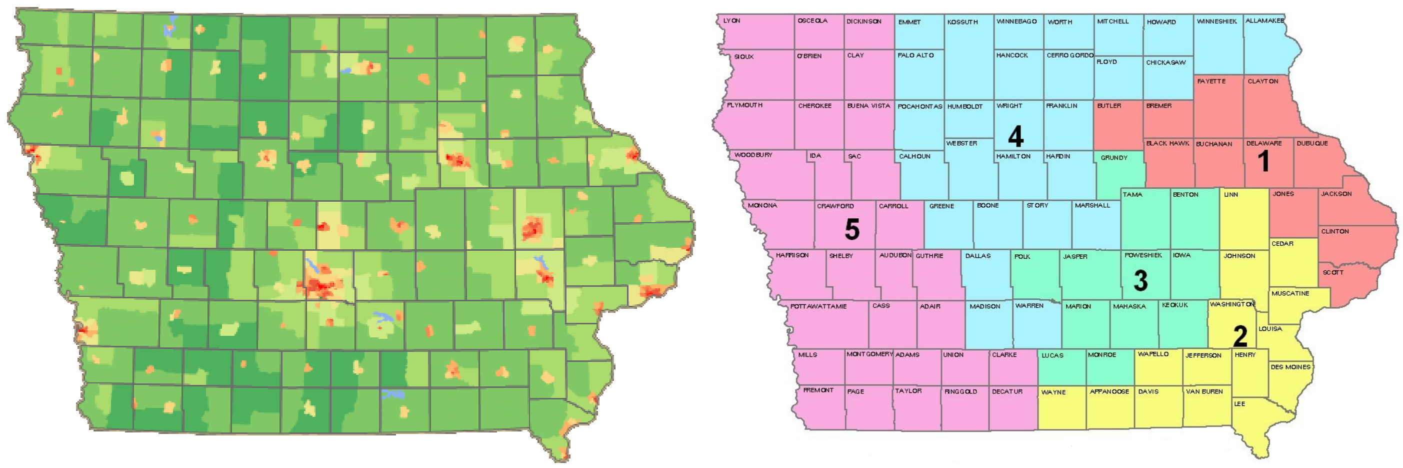


Figure 17: **The State of Iowa**
Iowa's 99 counties are divided into 5 congressional districts.

One of the early researchers in this field was Elbridge Gerry, signer of the Declaration of Independence, vice President under James Madison, but in 1811, the Governor of the state of Massachusetts. His party happened to be the Jeffersonian Republicans. He arranged for the voting district of Essex to be constructed in a way that encircled the city of Boston. A cartoonist made fun of its snakey shape by drawing on a pair of wings and calling the result a "gerrymander", a name which has been applied to this practice ever since.

Now to make this into a computational problem, suppose we have a graph of "wards", which are the "atomic units" of our voting system. We are required to create 5 voting districts, each of which is constructed by conglomerating adjacent wards. Each district should have roughly the same population. How would you go about solving such a problem? Here is some artificial data, suggested by the Iowa example, that you might try to work with. We assume there are 96 wards, arranged in an 8 by 12 checkerboard:

10	10	25	50	10	50	10	50	50	25	25	10
100	50	50	10	10	10	5	100	50	50	10	10
25	50	50	10	25	25	15	10	25	25	10	10
75	15	20	10	75	50	40	10	100	25	25	80
20	30	40	10	75	100	100	30	25	25	90	30
25	30	20	10	40	175	50	40	10	30	60	40
25	30	10	25	30	40	30	20	10	20	30	40
10	20	10	10	20	10	30	10	10	10	10	50

This problem is small enough that we could probably come up with some sort of solution by hand. But the point is, we want to use a computer to solve much bigger problems, so we start with the small problem and try to think of how we could "explain" to the computer a procedure that would produce a reasonable answer.

There probably isn't a formula that we can use to instantly assign every ward to a district. In that case, a typical procedure is to propose some solution, practically *any* solution, and then repeatedly try to make small improvements to it. So, how can we come up with an initial solution, no matter how bad it might be?

The only requirements are that every ward must be assigned to one of the districts, and the wards belonging to a district must be connected. One way to achieve this is to pick five wards randomly as the "seeds" for each district, and then to add neighboring wards one by one.

Specifically, here is a procedure that should work:

- 1: Pick 5 distinct wards at random, and assign them to districts 1 through 5, respectively.

- 2: If all wards have been assigned a district, we are done.
- 3: Select a random ward W that has not been assigned a district.
- 4: Select a random "side" S (east, north, west, south) of the ward.
- 5: If there is no neighboring ward on side S, return to step 3.
- 6: Let the neighboring ward on side S be called N. If N has not already been assigned a district, return to step 3.
- 7: Assign ward W to the same district as we previously assigned ward N.
- 8: Go to step 3.

This seems like a slow and indirect process, but it has the advantage that it will work properly no matter how many wards there are, and no matter how they are arranged geometrically, as long as we have at least as many wards as districts, and the wards form a connected set. In step 4, we assumed the ward was a rectangle with four sides, but it's clear that really we just need to be able to select one of the neighboring wards at random, no matter how many there actually are.

This procedure produces a division of the state into 5 voting districts. But we will almost surely not like the result - that's because we prefer each voting district to have roughly the same population, something our procedure didn't worry about at all. But now that we have a kind of solution, and an idea of what is wrong with it, it is obvious that we can try to make small adjustments to the solution to improve the population distribution.

Again, we want to suggest a procedure that is simple-minded enough for a computer to handle, and which does not rely on any special facts that might only occur in very special instances. An obvious idea is to consider "trading" a ward from one district to another. So we could pick a ward at random, ask if one of its neighbors belongs to a different district, and if so, ask whether, by transferring this ward to the other district, (or, the other ward to this district!) we improve the population distribution.

Now at this point, we will need to be more precise about this measurement of population distribution.

If we wished to make the problem more realistic, we could now suppose that for each ward we are given not just the populuation, but also the numbers D of Democrats and R of Republicans. This is going to change the problem in a way similar to how the "Knapsack Problem" problem changed when we added values to the objects. Although there may be many approximate solutions to the problem of grouping the wards into districts, we may prefer those arrangements that result in an advantage to a particular party.

If we are given the extra information of party affiliations, how might you try to create the voting districts so that:

- as many districts as possible have a Democratic majority;
- as nearly as possible, each district has the same ratio of Republicans and Democrats;
- the ratio of Republican and Democratic majority districts corresponds to the ratio of Republican and Democratic voters overall.

You might notice that the Gerrymander problem has some similarities to the problem of the Jealous Heirs. In both cases, we have many items, of fixed value, and we want to arrange them into groups of equal or roughly equal value. However, in the Gerrymander case, we have the additional severe restriction that the items in a group must form a connected geographical region.

Figure 18: **A numeric door lock**
How many 3 digit codes are possible?
How many don't use any digit twice?
How many are in ascending order?

14 Breaking the Code

Some homes and cars use a keyless system involving a pad with the ten digits 0 through 9. To open the lock, you need to push a certain number of digits (typically three or four) in the correct sequence.

How safe is such a system? Well, that depends in part on how hard it is to guess the code. A 1 digit code is worthless, since an intruder could guess it in no more than 10 tries. For a 2 digit code, the intruder has to guess the first digit, and then the second, but really, this just amounts to trying all the numbers from 00 up to 99, so the code can be guessed in no more than 100 tries. Not very secure!

By now, you should be able to guess that a 3 digit code allows for just 1,000 distinct choices, while a 4 digit code offers 10,000 choices. Even then, a very patient and careful intruder could eventually break in, simply by going through the codes in order.

An interesting twist occurs for certain kinds of locks. As you press the buttons, they stay depressed, until you have entered the complete code and tried the handle. This means that the code cannot use any digit more than once. Suppose we have a lock that uses a 3 digit code and does not allow repeats. Instead of 1,000 possible codes, how many do we have?

Let's think about how you might have gone about selecting your code. You have 10 choices for the first digit. No matter what you chose, that means that when you choose the second digit, there are only 9 choices remaining, and the third digit must be selected from 8. The number of choices for codes can be counted by multiplying the number of choices at each step: $10 \times 9 \times 8 = 720$.

Now suppose you want to generate every possible 3 digit code for which we don't allow digits to be reused. Isn't it time to call in a computer? A procedure to do this might look like this:

1. The first possible code is 012, so print it;
2. Add 1 to the current code;
3. If the code doesn't repeat any digits, print it;
4. If the code is 987, that's the highest possible code, so stop;
5. Otherwise, go back to step 2.

Note that we are treating the three digits of the code as though they represented a number. Instead of writing (0,1,2) we write 012. We do this because it makes it easy to move through all possible codes simply by adding 1. The rules of arithmetic take care of the fact that when the last digit of the code was 9, we reset it to zero and increase the next higher digit.

The other thing to note about this problem is that we can immediately tell how many possible codes we have to check. In our case, there are 1,000 codes to consider when we search for those that don't repeat any digits. There are two different ways to see that there are 1000 codes to check:

- our possible codes are every number from 000 to 999;
- there are 10 possibilities for each digit, and we have three digits to choose, making $10 \times 10 \times 10 = 1000$ codes

As far as computer calculations go, checking 1,000 cases like this does not represent a significant amount of work.

Suppose now that we want to choose a 3 digit code with the property that the digits are distinct, and increasing. It's probably not so easy to figure out that there are 120 such codes that are, but here's a hint as to why this happens:

$$\begin{array}{rll}
120 = & (1+2+3+4+5+6+7+8) & \text{0** codes} \\
 & +(1+2+3+4+5+6+7) & \text{1** codes} \\
 & +(1+2+3+4+5+6) & \text{2** codes} \\
 & +(1+2+3+4+5) & \text{3** codes} \\
 & +(1+2+3+4) & \text{4** codes} \\
 & +(1+2+3) & \text{5** codes} \\
 & +(1+2) & \text{6** codes} \\
 & +(1) & \text{7** codes} \\
 & +(0) & \text{8** codes} \\
 & +(0) & \text{9** codes}
\end{array}$$

You can see that there is only one possible three digit "7**" code beginning with a 7 and using ascending digits. Now look at the "6**" and "5**" cases, and look for the pattern.

Even if you can't see the pattern, you could still come up with a computer procedure that would generate all 100 possible codes, testing each one, and only printing out the ones with ascending digits.

It may seem like we have solved this problem without really understanding it - that is, we didn't really figure out a way to generate the ascending codes, but instead, figured out how to generate all the possibilities and then recognize the ones that are actually solutions to our problem. But sometimes, this kind of approach is the best we can do, or at least is easy to set up for small problems.

While there are other methods to deal with the numeric code problem, we will now look at the problem of the "Unseen Queens", where this ability to generate all configurations and then recognize the desirable ones is exactly what we will need.

15 The Unseen Queens

How many queens can you place on a chessboard so that no two queens can "see" (that is, attack) each other? Under the rules of chess, a queen can move any number of spaces horizontally, vertically, or diagonally. Thus, it's easy to see that, on a standard 8×8 chess board, if you try to place more than 8 queens, at least one column would contain two queens, who would hence see each other. Thus, the maximum number of queens that can coexist peacefully on a chessboard must be 8 or less. It turns out that you can reach that limit of 8, if you are careful. In fact, there are quite a number of ways that this can be done.

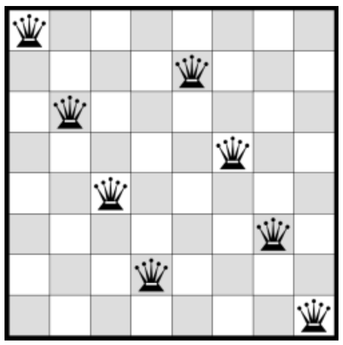


Figure 19: **Eight queens on a chessboard**
This is not quite a solution, since two queens see each other!

If we wish to study this problem, how can we represent the important features of the problem in a useful way? Given a representation of a chessboard with 8 queens, how can we determine whether any two queens can see each other? Can we find every possible configuration of non-attacking queens? Can we count the number of such configurations in advance? And if we don't have a formula for how to find solutions, is there a way to "stumble" towards a solution by starting with a random setup and then trying a series of reasonable adjustments?

Even though these questions are phrased in terms of a chessboard, the very same kind of issues come up all the time in mathematical and computational investigations. Considering this problem is like standing in a room with many doors - we will open several of these doors and suggest what is beyond them, in the hope that you may find yourself wanting to pursue one of these topics further.

15.1 Representing a Configuration

To begin with, we can consider the question of representing the configuration. An article about chess will describe a position by displaying all 64 squares; occupied squares are indicated by a small picture of the corresponding piece. This seems far too elaborate for our problem. Since the only chess piece we will use is a queen, we could simply mark an 8×8 array with 1's for those positions containing a queen, like so:

```
10000000
00001000
01000000
00000100
00100000
00000010
00010000
00000001
```

Since we happen to know that a legal configuration will always use exactly 8 queens, and that each queen must occupy a distinct column of the board, we can come up with an even more economical representating. Our scheme would be to start in column 1, and note the occupied row, then do the same for column 2, and so on. In that case, we can replace our matrix representation by a much simpler one, called the *column representation.* For example, we might have:

```
(1,3,4,6,3,5,7,2).
```

This representation certainly takes less space to write down. Another thing it does is make it easy to detect configurations that will not be acceptable. The example we have displayed can't be a solution, since we can easily see that there are two queens in row 3.

This suggests an even better representation, in which we still use eight digits, but now require that each digit be used exactly once. This will enforce the rule that each queen must occupy a distinct row of the board. An example would be

```
(1,3,5,7,2,4,6,8).
```

This is called a *permutation* of the digits 1 to 8, and so we have a *permutation representation* for configurations, which now automatically enforces two of the three requirements for non-attacking queens.

15.2 Checking for Attacking Queens

If we use the matrix representation for configurations, then a configuration is represented as a matrix A, which contains in row i and column j a value $A_{i,j}$ or $A(i,j)$, which is 1 if there is a queen at that position, and 0 otherwise. If we asked a computer to check that no two queens were in the same row or column, we might write something like this:

```
matrix check:
  for i = 1 to 8
    for j = 1 to 8
      if A(i,j) is 1 then
        A(k,j) must be zero if k is not i and
        A(i,k) must be zero if k is not j and
        A(i+j-k,k) must be zero if k is not j and
        A(i-j+k,k) must be zero if k is not j
```

Here, the number k is chosen to check all the possible rows, columns, diagonals (going down as we move right) and antidiagonals (going up as we move right) in the board.

On the other hand, if we use the permutation representation, we don't have to check rows and columns. Our representation is a permutation, p, which is a reordering of the digits from 1 to 8. To check that there are no attacking queens, we could write something like this:

```
permutation check:
  for i1 = 1 to 8
    for i2 = 1 to 8, skipping i1
      p(i2) - p(i1) must not equal i2 - i1
      p(i2) - p(i1) must not equal i1 - i2
```

We can see attacking queens immediately, but a computer can't. In order to give the computer the ability to take over the job of checking for attacks, we have to figure out a way to express the situation in terms of arithmetic. Whether we use the matrix or permutation representations, I expect you would not find it easy to come up with a quick way of checking for the diagonal attacks. Something that we can do at a glance can turn out to be surprisingly difficult to define as a formula.

15.3 Exhaustion: Generating and Checking All Possible Configurations

In the combination lock problem, we were seeking all the combinations that had some property (no repeating, or increasing digits). Since we didn't know a formula that would magically produce, one by one, just the desirable combinations, we took the simple approach of generating every possible combination, and rejecting those that didn't satisfy us.

The same approach can be adopted for the queens problem - generate every configuration, and keep the good ones. However, it may be a shock to discover that, unless we think ahead, there can be a prohibitively huge number of configurations to check.

For instance, we might choose the matrix representation for our problem, in which we have 64 squares, 8 of which contain a 1 representing a queen. If we can generate each of these configurations, then our computer procedure *matrix check* can alert us to the special cases where the queens are non-attacking. But the number of configurations to check is $\binom{64}{8}$ (because we have 64 choices for the first queen, 63 for the second, and so on, and we have to divide by 8! to account for the fact that the same result can be gotten in several ways). This number is about 4 billion (4,426,165,368 to be exact). Even for a computer, this starts to represent a serious amount of work!

If we choose the column representation, where we assume that there is exactly one queen in each column, then we automatically discard a huge number of configurations, without losing any of the ones of interest to us, and since, for each slot in the representation, we have 8 possibilities, the number of configurations is about 16 million: $8^8 = 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 = 16,777,216$.

But the 16 million configurations include many where a row is used twice. If we move to the permutation representation, in which each digit from 1 to 8 appears exactly once, we drop down to about 40 thousand possibilities, because $8! = 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 40,320$. So an efficient procedure might be to use the permutation representation, generating every possible permutation, and calling the procedure *permutation check* to determine whether the permutation represents a solution of the queens problem.

For the matrix and column representations, it is easy to see how to generate one representation after another, because they are really very similar to counting. For instance, the list of all possible column representations looks something like:

```
11111111
11111112
11111113
...
45327816
45327817
45327818
...
...
88888887
88888888
```

If we want to use the very efficient permutation representation, is there a way to generate the permutation list efficiently. (Since we want an efficient computation, we do *not* want to generate the permutations by generating the 16 million column representations first and then selecting those that are permutations!) The shorter list we are talking about looks like this:

```
12345678
12345687
12345768
...
87654231
87654312
87654321
```

The answer to this question is <u>yes</u>. We will not go into the procedure here, but you might think about how to do it. If you think about it, you should see that you can look at the digits of any permutation in the list, and work out what the very next one will be!

15.4 Improving an Imperfect Configuration

Suppose that we don't want every possible configuration - we just need one. Suppose we don't have the time or inclination to think deeply about the problem, and we don't have a computer. Wouldn't we simply use trial and error? More specifically, we would slap down the pieces more or less at random, and then try to make some simple changes that improve the situation a little. Eventually, we will improve ourselves right

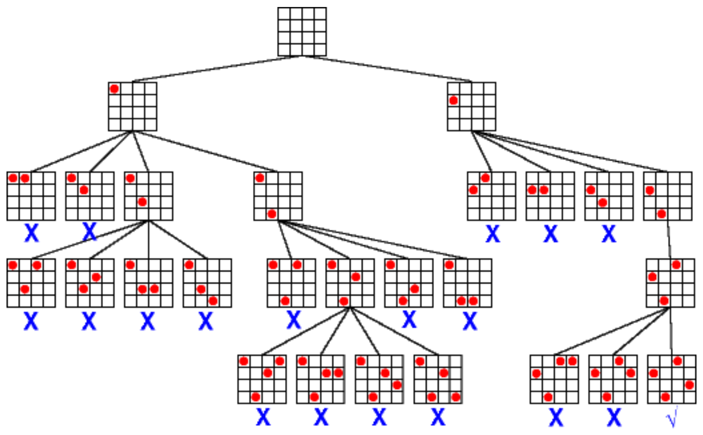


Figure 20: **Backtracking**
How backtracking works for queens on a 4x4 board.

into a solution or else reach a dead end, where we don't have a solution, but there is no obvious improvement to make. In that case, we have nothing to lose, since we can just start over with another more or less random configuration.

The key to this approach is to have some way of examining the board and choosing an improvement. For example, here is one approach that could be taken: *If your current configuration is not a solution, find a queen that is under attack. If under row attack, maintain the column but move to an unoccupied row. If under column attack, maintain the row, but move to an unoccupied column. If under diagonal attack, pick any third queen, and swap rows with her, while preserving columns.*

A human could certainly come up with more sophisticated improvements, but we can take these as a starting point. You might try to see whether you can quickly arrive at a solution to the problem by repeatedly "improving" an arbitrary starting setup.

15.5 The Backtrack Method

Another way to try to handle the queens problem is similar to how we might try to solve a maze, or a Sudoku, or other puzzles where the solution can be thought of as a series of choices. For a maze, we choose whether to go left, straight, or right; for a Sudoku, we choose what digit to put in each empty box; for the queens problem, we choose where to place the first queen, the second, and so on.

The *backtrack* method is useful for problems where we don't have an overall solution strategy, but we can often very quickly detected that a partial solution is a dead end. In situations where there are very many possible configurations to check, the backtracking method can be very powerful if we are able to spot dead ends quickly. This is because when we detect that a partial solution is actually a dead end, we are actually skipping over *all* configurations which begin with the partial solution.

Let's look at what this means for the queens problem. To simplify the discussion, we'll cut back to a problem where we are trying to place just 4 queens on a 4×4 board.

```
  1  2  3  4        (1,1)  (1,2)  (1,3)  (1,4)
  5  6  7  8        (2,1)  (2,2)  (2,3)  (2,4)
  9 10 11 12        (3,1)  (3,2)  (3,3)  (3,4)
 13 14 15 16        (4,1)  (4,2)  (4,3)  (4,4)
```

We begin with an empty board. Now we have 16 positions in which we could place the first queen. We place her in the first position, 1, the upper left corner with coordinates (1,1), and remember that we still have 15 untried possibilities for the first move, which we will have to come back to. Our partial solution is now (1,?,?,?).

Now we must place the second queen, and we have 15 places available. We can quickly reject positions 2, 3, 4, 5, and 6, while 7 is a possibility. Our partial solution is now (1,7,?,?). We have eliminated all configurations that start (1,2,?,?), (1,3,?,?), (1,4,?,?), (1,5,?,?), (1,6,?,?), and we must remember that we haven't yet checked using positions 8 through 16 for the second queen.

For the third queen, it seems we have 14 positions open, but we can quickly eliminate all of them except position 14! So the only partial solution that begins with (1,7,?,?) must be (1,7,14,?).

But now when we want to place the fourth queen, we see that there are no unattacked squares left. So there are no full solutions that begin with (1,7,14,?). Since 14 was our only choice for the third queen, we know that there are no solutions that begin (1,7,?,?).

So, if the first queen is in position 1, the second queen can't go in position 7. But we haven't checked putting her in positions 8 through 16. So we consider the partial solution (1,8,?,?), and move ahead to see if we can place the third and fourth queens.

As it turns out, there are no full solutions if the first queen is placed in position 1. The backtracking method then suggests we try placing the queen in position 2. After a few steps, we discover that (2,8,9,15) is a solution.

```
  1  Q  3  4
  5  6  7  Q
  Q 10 11 12
 13 14  Q 16
```

If we just wanted one solution, then we can stop. But the backtracking method also allows us to continue our search methodically, so that we can, if we wish, be sure that we discover every possible solution to the problem.

16 Conclusion

When we say that we understand some phenomenon such as planetary motion, the formation of hurricanes, the spread of disease, or the melting of a stick of butter, we generally mean that, in our minds, we have a mental model, comprising a set of objects, and rules for how they interact and change.

When we encounter an unfamiliar situation that we wish to understand, such as the way an airplane wing provides lift, we try out various new models in our heads, and see if their behavior conforms to the facts that we see.

The simple problems we have discussed here are meant as modeling exercises. For instance, from the description of a situation, we might need to identify the interesting objects and the rules that govern them, form some typically numeric way of describing the various possible configurations of the objects, and then work out a configuration that satisfies some desirable criterion.

The problems presented here are small, simple, and artificial, but they offer a fair introduction to the kinds of thinking necessary for any kind of scientific work.