

Chapter 6: The Metropolis Algorithm

You are now able to confirm that the Connor/Simberloff random swap algorithm for data like the finches does not give uniform limiting probabilities. If you create a random data table by starting with the finch data and making a large number of random swaps, then the table you end up with is more likely to be a table with many neighbors, less likely to be a table with few neighbors.¹ In short, then, the algorithm doesn't do what we need it to do in order to carry out a permutation test. Does that mean we have to scrap the algorithm, or can we find a way to fix it? Answering that question (by finding a way to fix the algorithm) is next on the agenda, and the topic of this chapter.

Introduction: looking back, looking ahead.

Our goal is to find ways to estimate p -values for complicated data sets like the finch data. Our method is this:

Algorithm for estimating a p -value

1. *Generate* a random sample of data sets X_1, X_2, \dots in a way that gives each possible data set the same probability of getting into the sample.
2. *Compare*: For each random data set, compute the value of the test statistic, and record a Yes or No: Yes if the value is greater than or equal to the value for the actual data, No otherwise.
3. *Estimate* p using the observed fraction of Yes answers in the sample:

$$\hat{p} = \frac{\#Yes}{\#data\ sets\ in\ the\ sample}$$

For a simple situation, as with the Martin data, there are straightforward ways to generate random data sets with uniform probabilities. For other situations, no simple methods exist. For many such data sets, and for the finch data in particular, we can try to generate random data sets using a random walk on a graph that has all the possible data sets as its vertices:

Generating random data sets: Step 1, expanded

- a. IN: *Start* with the observed data set.
- b. *Move* to one of the neighboring data sets (vertices) at random, choosing from among the neighbors with equal probabilities.

¹ If you go from one table to another using the swap algorithm, then the number of neighbors is the number of swappable 2x2 sub-matrices – what some ecologists call the number of checkerboard units (CUs).

- c. *Repeat*: Continue with random moves (repeat Step b for each move) until you have completed a very large number of steps.
- d. **OUT**: Output the data set (vertex) where you are at the end of Step c.

Example. Mini-finches.

There are only five 0,1 matrices with row sums 2 1 2 and column sums 2 1 2:

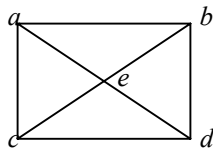
0 1 1	1 1 0	1 0 1	1 0 1	1 0 1
1 0 0	0 0 1	0 0 1	1 0 0	0 1 0
1 0 1	1 0 1	1 1 0	0 1 1	1 0 1
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>

Suppose, purely for the sake of keeping the example simple, that we decide to use the value of the center cell (the (2,2) element of the matrix) as our test statistic, and that our observed data set is *e*. Thus the observed value of the test statistic is 1, and the *p*-value is

$$p = P(\text{(2,2) element of a randomly chosen matrix is } \geq 1).$$

In this particular example, there is only one possible data set, *e*, with a value of 1 or greater, so $p = P(e) = 1/5$.

Now consider what happens if we try to use the random swap algorithm to estimate *p*. The random swaps define a walk on the five-point graph



If we use the graph walk to generate random data sets, then over the long run, the number of steps at each vertex is proportional to how many neighbors it has. This means that for every 3 times we end up at data set *a*, we can expect to end up 4 times at data set *e*. Consequently, in our random sample we should expect four *e*s for every three *a*s. Using such a sample to estimate *p* gives

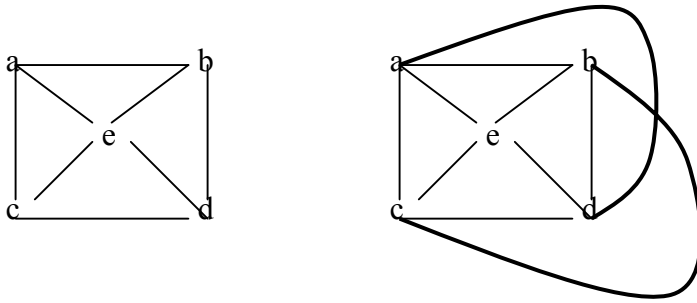
$$\hat{p} = \frac{\#e}{\#a + \#b + \#c + \#d + \#e} \xrightarrow{\text{as \#steps} \rightarrow \infty} \frac{4}{3 + 3 + 3 + 3 + 4} = \frac{1}{4}$$

As things are now, if we use the swap algorithm to generate random data sets, our estimate of the *p*-value *will converge to the wrong answer*, .25 instead of .20.

This chapter is about ways to fix the graph walk so that all vertices are equally likely in the long run. Before looking for strategies, however, we should think about what is required for a strategy to be one we could use in practice.

6.1 Updating the applied problem: toward a rat's eye view of graph walks

One approach to fixing the swap algorithm would be to identify the co-occurrence matrices that have few neighbors, and give them additional neighbors by adding new edges to the graph, doing this in such a way that leaves every vertex with the same number of edges. In theory, this would do the job, because if all vertices have equal numbers of neighbors, their limiting probabilities will also be equal. Display 6.1 shows an example of this strategy.

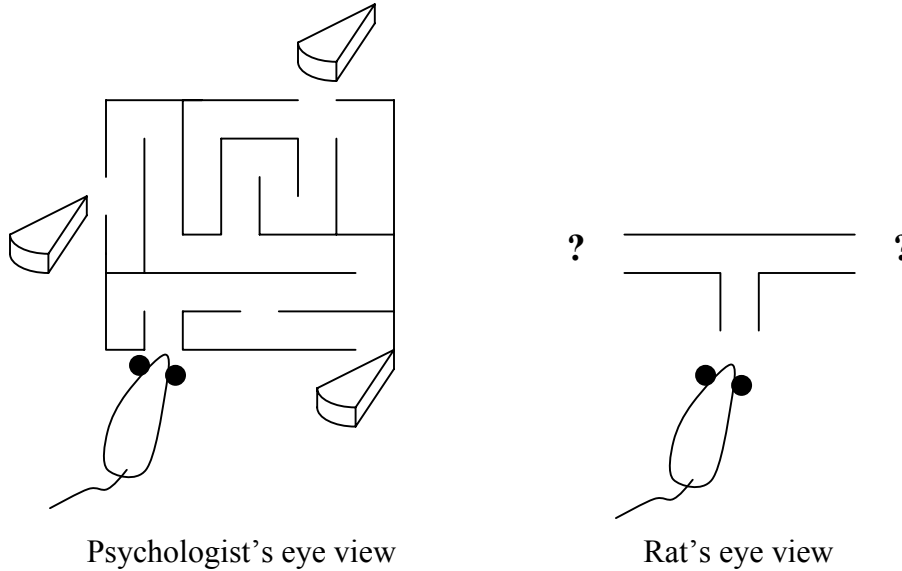


Display 6.1 Equalizing limiting probabilities by adding edges

A walk on the left-hand graph has unequal limiting probabilities. Vertex e , with 4 neighbors, is more likely than the others, which have only 3 neighbors each. The graph on the right has been “fixed” by adding two new edges, so that now each vertex has four neighbors, and the limiting distribution is uniform.

Unfortunately, the strategy of adding edges, though simple and elegant, is of absolutely no use in practice.² To see why, compare the following two views of a novice rat running a maze in search of three cheeses. (Display 6.2) Which of the two views makes it easier to change the maze so that all three cheeses are equally likely?

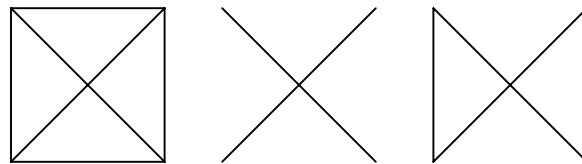
²This is a wonderful instance of how practical exigencies can push curious thinkers to create new mathematics. If it weren't for needing a solution you could actually implement, work on the mathematical ideas might well have stopped with the “simple and elegant ‘solution.’” In the same way, if all the late seventeenth century astronomical observations that theory said should be equal to each other had in fact been equal, Legendre, Gauss and Laplace would not have felt the need that spurred them to develop what is perhaps the prettiest area in all of statistical theory, least squares. In turn, least squares theory both anticipated and served as a paradigm for one of the most elegant and influential branches of twentieth century mathematics, functional analysis. Moreover, it cuts both ways: Just as applied problems inspire the creation of new theory, existing theory often guides the search for solutions to practical problems. Elegance entirely aside, functional analysis is what guided engineers, well before the days of desktop computing, to figure out a way to send hundreds of phone conversations over a single wire at the same time, and then unscramble them at the other end.



Display 6.2. Two views, global and local, of a rat in a maze

To the omniscient psychologist looking down from above, the possible paths are clear, because from above you can see past one turn to the next, and on from there. The rat, however, can only see the current set of choices, and not beyond.

Now transfer the metaphor to walks on a graph, as in Display 6.3.



Display 6.3 Three graphs that look the same from the center vertex

To a psychologist who puts a rat down at the center vertex, the three graphs are clearly different. To the rat at the center vertex, they all look the same. In conventional mathematical language, the psychologist has a **global** view, the rat has a **local** view.

If you are a smart rat who is lucky enough to live on a small graph, you can walk around long enough to map the whole graph, take a global view, and figure out which new edges would make all your vertices equally likely. But if you are a rat who lives on a graph with 10^{17} vertices, then, no matter how smart you are, making a map is hopelessly ambitious. You can never manage to soar above the labyrinth, your view will always be a rat's-eye view, and while you may aspire to think globally, you can only act locally.

The challenge, then, *is to find a local, rat's-eye strategy for fixing the random swap algorithm*. Adding new edges is not an option, because you can't figure out where they should go. The only vertices you can see are the ones that already have edges leading to the one where you are.

The current version of the mathematical challenge.

From any vertex v , the only thing we know is the list of edges incident to v , or, equivalently, the list of vertices that are neighbors of v . Until now, this information has been adequate. It is exactly what we have needed for a graph walk: We simply pick one of the known neighboring vertices, with uniform probabilities, and go there. Once there, we can see the new list of neighbors, and we repeat the process.

Now suppose that instead of choosing from among the neighboring vertices with *equal* probabilities, we are allowed to assign whatever probabilities we choose. For example, if we are currently at vertex a , and there are two neighbors b and c , the rules we have been using say we move next to b or c , choosing with equal probabilities. In the notation of transition probabilities,

$$p_{aa} = 0, \quad p_{ab} = p_{ac} = \frac{1}{2}, \quad \text{and} \quad p_{av} = 0 \quad \text{for all other vertices } v.$$

The last part, $p_{av} = 0$ for all other vertices, can't be changed, because to our rat's eye, these vertices are invisible. We can't even tell if there are 1000 of them, none at all, or 10^{17} .

According to our rat's-eye rules, we *are* allowed to change p_{aa} , p_{ab} , and p_{ac} , so long as we keep their sum equal to 1. Here, then is the mathematical challenge:

Given a graph G , find a rule for assigning transition probabilities to the current and neighboring vertices, based on knowing only the list of neighbors to the current vertex, so that the resulting Markov chain will have a uniform limiting distribution.

The Metropolis algorithm, which is the topic of this chapter, provides a solution. I find it useful to think of the Metropolis algorithm as a natural extension of similar but simpler algorithms in the same spirit. Accordingly, in what follows, I show you how you can construct the algorithm from a simple beginning. As a vehicle for thinking in concrete and intuitive terms both about why uniform probabilities are important, and about strategies for equalizing a non-uniform distribution, try the following activity.

Activity: Adventures in String Theory?³

Step 0. Prepare a bag of strings of various lengths. About 25 strings with lengths between 4 and 15 inches generally works well.

³ Based on Scheaffer, Richard L, Mrudulla Gnanadesikan, Ann Watkins, and Jeffrey A. Witmer (1996). *Activity-Based Statistics: Student Guide*, New York: Springer, p. 105-106.

Step 1. Without looking, reach into the bag, mix up the strings, draw one out, measure it, record the length, and replace the string.

Step 2. Repeat Step 1 for a total of about 10 times. Then compute the average of the lengths in your sample.

Step 3. Now either take all the strings out of the bag, measure their lengths, and compute the actual average length for your population of strings, or at least imagine doing that.

Discussion questions.

1. Apart from chance variation, would you expect the average for the sample to be pretty much equal to the average for the whole population, larger than the population average, or smaller? Explain.
2. Describe the bias in the sampling method. What is it that makes some strings more likely than others to be chosen for the sample?
3. (a) Suppose a bag has only two strings, one 12" long and one 6" long. What is the (approximate) probability you choose the 12" string? (b) Suppose now that the bag has three strings, of lengths 12", 8", and 4". What are the probabilities that go with the three lengths? (c) What is the relationship between the length of a piece of string and the probability that it is chosen for the sample?
4. Describe any parallels you see between sampling from the bag of strings and walking at random on a graph.
5. Consider using a random sample of hospital patients to estimate the average duration of a hospital stay. Explain why you would get a biased estimate if you simply took a random sample of people in the hospital on any given day, followed up later to find out how long each person stayed, and took the average of those values.
6. What is the relation between (5) and the strings? Can you think of other real-world scenarios where bias of this sort could be a problem?
7. If your goal is to estimate the average length of the strings in the bag, how can you use the biased sample and yet manage to remove the bias so as to get an unbiased estimate of the average length? (There are a number of ways to do this.)

Russian roulette: "killing" over-represented strings

Taking strings at random from the bag gave a biased sample because some strings were more likely to be chosen than others. For an unbiased sampling method, all strings should be equally likely. In other words, the probabilities should be uniform.

Consider the following strategy, sometimes called Russian Roulette:⁴ Each time you choose a string, you measure it, and then toss a coin. If it lands heads, you “kill” the measurement, that is, just return the string to the bag without recording its length. If the coin lands tails, you record the length. For this method to work, the chance of heads must be allowed to depend on the length of the string you have chosen, so that, by the right choices for $P(\text{Heads})$, you can adjust the sampling method to give all strings the same chance to be recorded.

Discussion questions.

8. Suppose the bag contains two strings, one of length 12” and one of length 6”. Assume that the chance of choosing a string is proportional to its length. (a) Tell how to use Russian roulette to make the two strings equally likely to have their lengths recorded. (b) Over the long run, what percentage of the sampled strings get recorded?
9. Now suppose there are three strings, with lengths 12”, 8”, and 4”. (a) Tell how to use Russian roulette to get unbiased samples. (b) What percentage of samples get recorded?
10. Is the strategy in (9) local or global? (Explain.)

Investigations

11. Suppose you don’t know the lengths of the strings in the bag, although you do know that all strings are at least 2” long. (a) Tell how to use Russian roulette to ensure that all strings have the same chance of being recorded. (b) What percentage of samples get recorded?
12. Russian roulette for graph walks. Consider a graph walk, whose limiting probabilities are proportional to the vertex degrees. For graphs with differing numbers of vertices, Russian roulette offers a strategy for equalizing the limiting probabilities. Each time you move to a new vertex, you toss a coin to decide whether to record that new vertex. If the coin lands heads, you don’t record; if tails, you do. (Either way, the “new” vertex becomes the “current” vertex, and its neighbors are the choices for the next new vertex.) Find a formula for $P(\text{Heads})$, in terms of the vertex degree, such that the limiting probabilities for the recorded vertices will all be equal. (Suggestion: Start with simple examples.)
13. Find a formula for the overall fraction of steps that get recorded.

6.2 Holding in place (Waiting for Metropolis?)

A shortcoming of the version of Russian roulette you’ve been working with is that you end up throwing away a lot of information, which means that it can take a very long time

⁴ Kahn, Herman (1956). “Use of different Monte Carlo techniques,” in Herbert H. Meyer, ed., *Symposium on Monte Carlo Methods*, New York: Wiley, pp. 146-190.

for your Markov chain to do its job, for example, to give you enough data to for a good estimate of a p -value. Instead of using a coin toss to decide whether to count a vertex or throw its data away, you can use a coin toss to decide whether to move to a new vertex, or just stay where you are and count the current vertex again..

Here's the same basic idea, without the coin tosses:

Weighting. Each time you reach a vertex v , print n_v tickets that say “ v ”. Choose the numbers n_v (the **weights**) in such a way that the distribution of vertex labels on the tickets has a limiting uniform distribution.

The rationale: The algorithm for estimating p -values assumes that all data sets (vertices) are equally likely, and takes an average over all data sets in the sample, giving them equal weight. If the vertices are not equally likely, we use ticket-printing to compensate. Vertices that don't get visited very often get to print lots of tickets when their turn comes; vertices that get lots of visits aren't allowed to print many tickets per visit.

Instead of printing extra tickets at a vertex, you can think of staying there for additional steps, or “waiting.”

Waiting. Each time you reach a vertex v , toss a coin, and stay there – that is, “move” from v to v and record v again – with probability p_{vv} , and move to a neighboring vertex with probability $1 - p_{vv}$, choosing from among the neighbors with equal probabilities. Set the values of the **self-loop probabilities** p_{vv} so that the limiting distribution for the resulting walk is uniform.

In effect, this strategy installs a “stochastic gatekeeper” at each vertex. To “get out of jail” from vertex v , you toss a biased coin with $P(\text{Heads}) = p_{vv}$. If it lands heads, you record v again and stay there for another step; tails, you move to a neighboring vertex and record it. You set the values of the probabilities so that vertices that get fewer visits are harder to escape from, vertices that get lots of visits are easy to leave. You can think of staying in v for one step as equivalent to printing one ticket that says “ v ”. Thus a long stay corresponds to printing many tickets, and a short stay, to few. In this sense, *waiting is just randomized weighting*. It is possible to prove that the self-loop probabilities p_{vv} must be chosen so that the expected number of steps until you leave is equal to n_v .

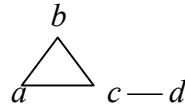
Discussion questions

14. Consider the three point linear graph: a — b — c . (a) Find its transition matrix. (b) What are the limiting probabilities? Which vertex or vertices are over-represented? Which vertex or vertices are under-represented?
15. At each under-represented vertex, install a stochastic gatekeeper, using a fair coin: with probability $\frac{1}{2}$, you stay at the vertex for another step; with probability $\frac{1}{2}$, you move to a new vertex, choosing from among the adjacent vertices with equal probabilities.

Find the transition matrix for the new walk, and verify that the limiting probabilities are uniform.

16. Use the same strategy to modify the walk on a 4-point linear graph ($a - b - c - d$) so that its limiting probabilities are equal.

17. Consider a walk on the “triangle with tail”:



(a) Write the transition matrix and the limiting probabilities. Verify that the vector of these limiting probabilities satisfies $\mathbf{xP} = \mathbf{x}$. (b) Which vertex or vertices are over-represented? Which vertex or vertices are under-represented?

18. Consider installing a stochastic gatekeeper at vertex d : toss a coin with $P(\text{Heads}) = p$; if it lands heads, stay at vertex d for another step of the walk; if it lands tails, move to c . (a) Write the transition matrix of the new walk. (b) Use the matrix to verify that no value of p will give limiting uniform probabilities.

19. Now consider installing gatekeepers at all vertices except c . (a) Explain why no gatekeeper is needed at c . (b) Explain why $P(\text{Heads})$ should be the same at vertices a and b . Let this probability be p_1 . (c) Let p_2 be the value of $P(\text{Heads})$ at vertex d . Which should be larger, p_1 or p_2 ? Explain why.

20. (a) Write the transition matrix for the walk in (19), in terms of p_1 and p_2 . (b) What values of p_1 and p_2 will give uniform limiting probabilities?

Investigations: limiting probabilities and rates of convergence

21. Complete the following tables. Then use them as a starting point for finding a rule for choosing probabilities p_{ij} that give uniform limiting probabilities.

(13) - (14)	a	b	c
Degree d_i			
$P(\text{stay}) p_{ii}$			

(15)	a	b	c	d
Degree d_i				
$P(\text{stay}) p_{ii}$				

(16) - (19)	a	b	c	d
Degree d_i				
$P(\text{stay}) p_{ii}$				

22. Give an algebraic proof that your rule in (21) works.

23. Consider the 3-point graph of Discussion Questions 14-16. At vertices a and c , install gatekeepers with $P(\text{stay}) = p_{11}$. At vertex b install a gatekeeper with $P(\text{stay}) = p_{22}$. (a) Write the transition matrix. (b) Write the system of two linear equations that p_{11} and p_{22} must satisfy in order to get uniform limiting probabilities. (c) Find the solution set for your system. (d) Use a computer to investigate the relationship between the value of the p_{ii} and the rate of convergence of $\mathbf{p}^{(n)}$ to its limiting values.
24. Extend your analysis using other examples.

Exercises: The Finch connection.

25. Consider holding in place applied to the 2x2 swap walk on co-occurrence matrices. For the 3x3 example with row sums 2, 1, 2 and column sums 2, 1, 2, consider the algorithm that is like the 2x2 swap, but with the following modification: When you choose a 2x2 sub-matrix, pick two rows at random, then pick two columns at random. If the resulting 2x2 is swappable, make the swap and record. If the 2x2 is not swappable, hold in place and record.

Write the transition matrix for this walk, and verify that the limiting probabilities are uniform.

26. Write the transition matrix that you get using the formula from (21). Which chain, this one or the one in (25), mixes faster?
27. Generalize your result in (25) to 2x2 swap walks on any set of co-occurrence matrices with given margins: Describe the rule for holding, and show that the rule gives uniform limiting probabilities.

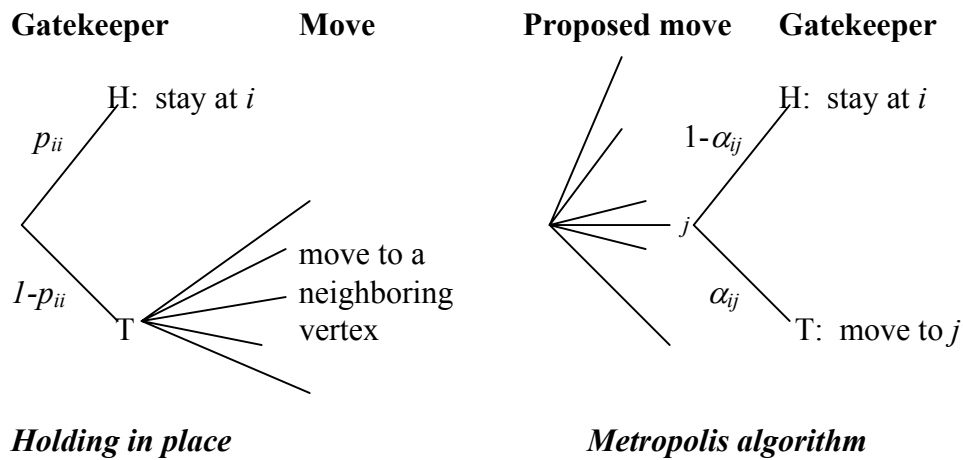
6.3 Smart holding in place: the Metropolis algorithm for graph walks.

So far, you've seen two strategies for equalizing limiting probabilities. **Russian roulette** installs "stochastic janitors" that throw away data whenever a coin toss lands heads. Although the strategy works – it throws away enough data from over-represented vertices to give uniform limiting probabilities -- it is inefficient because so many steps don't get counted. **Holding in place** installs stochastic gatekeepers – they won't let the walk move to a new vertex until a coin lands tails. This strategy also works – it makes you stay long enough at under-represented vertices -- but while it doesn't throw away data, it does tend to create a walk of the sort that MCMC practitioners call "sticky." (Just think of walking on second-hand bubble gum.)

Holding in place uses "dumb" gatekeepers – they decide whether to let you pass or not regardless of where you may be headed. The Metropolis algorithm substitutes "smart" gatekeepers that take into account where you propose to go before they decide whether to let you pass or make you stay in place. Using smart gatekeepers lets you move more

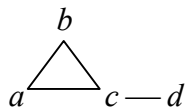
often – whenever you want to go to an under-represented vertex – and this makes the walk less sticky.

Put differently, with holding in place, you first decide whether to stay or move. Then, if allowed to move, you choose a destination. The decision whether to stay or move depends only on d_i , the degree of the vertex where you are, and when you do move, all neighboring vertices are equally likely, regardless of whether they are under-represented or over-represented. With the Metropolis algorithm, you first decide where you will go if allowed to move; then you decide whether to go there or stay in place. For this algorithm, the “smart” decision – stay or move – is allowed to depend not only on d_i , the degree of the vertex where you are, but also on d_j , the degree of the vertex where you propose to go. (See Display 6.4.) This allows different destinations to have different probabilities.



Display 6.4 Holding in place (left) has a “dumb” gatekeeper. The Metropolis algorithm has a “smart” gatekeeper that looks at the proposed destination.

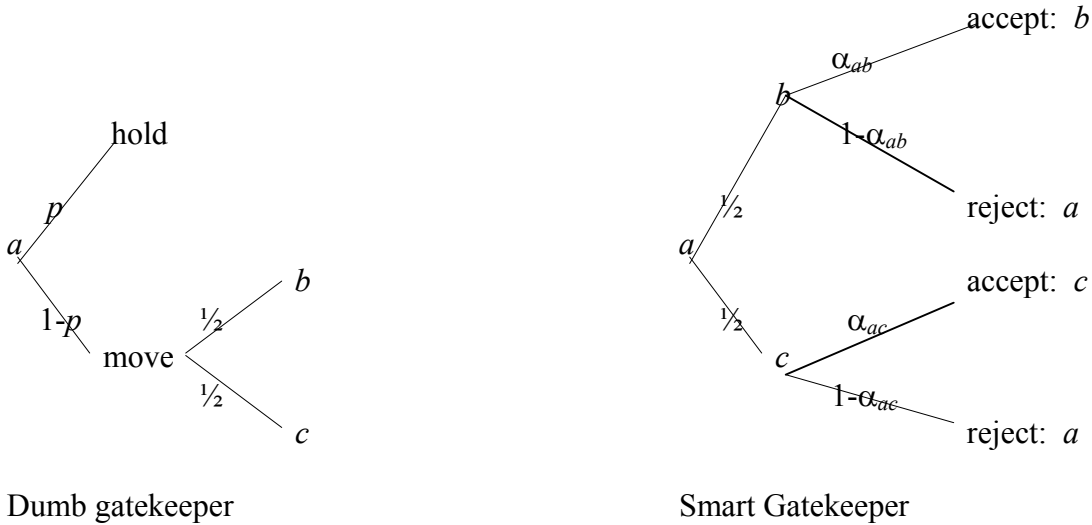
To illustrate the difference between smart and dumb gatekeepers, consider the walk of Exercises 17-20, on the “triangle with tail”:



Discussion questions

- 28. Compare the limiting probabilities for this graph walk with the target set of uniform probabilities. Which vertices are over represented? Underrepresented?
- 29. Suppose the walk is at vertex c . Explain why there is no need for a gatekeeper.
- 23. Now suppose the walk is at vertex d . Explain why there is no difference between a smart gatekeeper and a dumb one.

31. Next, suppose the walk is at vertex a . There are two possible destinations, b and c , with different vertex degrees. The dumb gatekeeper restricts moves from a to c by restricting *all* moves from a . The smart gatekeeper makes separate decisions based on your proposed destination. (See Display 6.5.) Explain why proposed moves from a to b should always be accepted.



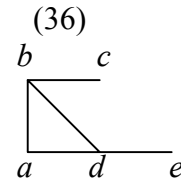
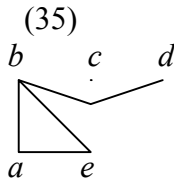
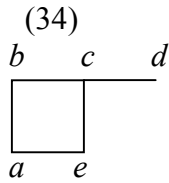
Display 6.5 Dumb and smart gatekeepers for the walk on a “triangle with tail”

32. Finally, consider moves from b . Explain why $\alpha_{ba} = 1$ and $\alpha_{bc} = \alpha_{ac}$.

33. Let α_{dc} be the acceptance probability for a proposed move from d to c . (a) Write the transition matrix \mathbf{P} for the walk with the smart gatekeepers, in terms of the unknowns $\alpha_{ac}, \alpha_{ca}, \alpha_{dc}$. (b) Write the system of linear equations that the α s must satisfy to give uniform limiting probabilities. (c) Solve to find the values that give uniform limiting probabilities. (d) Rewrite the transition matrix \mathbf{P} , this time with numerical values for all entries.

Exercises

(34)- (36) For each of the three graphs below (a) identify the vertices for which no gatekeeper is needed. (b) Identify vertices i for which dumb and smart gatekeepers are the same. For each such vertex find the acceptance probability α_{ij} that a move away from vertex i is allowed. (c) For each remaining vertex i , identify the destination vertices j for which no gatekeeper is needed. (d) For each remaining vertex pair (i, j) that needs a gatekeeper to restrict transitions $i \rightarrow j$, let α_{ij} be the probability of accepting a proposed move from i to j . Write the transition matrix. (e) Write the system of equations the α_{ij} must satisfy to give a uniform limiting distribution. (f) Solve the system, and write the resulting transition matrix.



Investigations

37. Summarize your work in (34) – (36) by filling in the tables below. (You’ll need to add more rows to your tables.)

A Vertices for which no gatekeeper is needed

Graph #	Vertex	Relevant features

B Vertices for which dumb and smart are equivalent

Graph #	Vertex i	Vertex degrees		α_{ij}
		From: d_i	To: d_j	

C Acceptance probabilities for other vertices

Graph #	Vertex pair (i, j)	Vertex degrees		α_{ij}
		From: d_i	To: d_j	

38. By design, the acceptance probabilities for a proposed move depend only on the degrees of the starting and proposed ending vertices, not on the rest of the graph. Fill in the table below of acceptance probabilities:

Acceptance probabilities $\alpha_{ij} = \alpha(d_i, d_j)$

		degree d_j for proposed ending vertex j			
		1	2	3	4
degree	1				
d_i of	2				
starting	3				
vertex i	4				

39. Summarize the table by giving a formula for α_{ij} in terms of d_i and d_j .
40. For each walk above, find the transition matrix for the Markov chain you get by using dumb gatekeepers to produce uniform limiting probabilities. (b) Which mixes faster, the Metropolized walk or the walk with dumb gatekeepers? (c) Characterize the graph walks for which the two convergence rates will be the same.

Extensions of the Metropolis algorithm to Markov chains

You now have seen how to use Metropolis coin flips to give a random walk on a graph a uniform stationary distribution. What if you have a Markov chain that is not a graph walk? Is it still possible to use Metropolis coin flips to get a uniform stationary distribution? If so, how? (What are the acceptance probabilities?) Is this possible for all transition matrices, or just some of them? (If only some, which ones?)

Suppose a Markov chain with transition matrix $\mathbf{P} = \{p_{ij}\}$ has limiting distribution $\mathbf{p} = (p_1, p_2, \dots, p_k)$. The rest of this section shows how to extend the formula for acceptance probabilities α_{ij} , writing them in terms of p_i and p_j , so that the Metropolized chain with transition matrix $\tilde{\mathbf{P}} = \{\tilde{p}_{ij}\}$, $\tilde{p}_{ij} = \alpha_{ij} p_{ij}$, will have a uniform stationary distribution. The extension relies on the fact that *if a transition matrix is symmetric, then it has a uniform stationary distribution*. To implement the general version of the Metropolis algorithm, you compare all pairs p_{ij} and p_{ji} , and, whenever the two probabilities are not equal, you force them to be equal by inserting a coin flip to reduce whichever probability is larger.

Metropolizing a Markov chain:

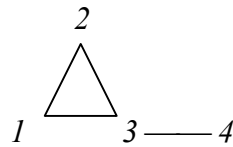
Step 1 (Theory). Find the new transition matrix $\tilde{\mathbf{P}}$: Identify all transitions for which $p_{ij} > p_{ji}$. Shift the excess probability $(p_{ij} - p_{ji})$ from p_{ij} to the diagonal p_{ii} .

Step 2 (Implementaion). Set the acceptance probabilities α_{ij} for the Metropolis coin flips. To reduce a transition probability from p_{ij} to p_{ji} , insert a coin toss with acceptance probability α_{ij} . Set the value of α_{ij} by solving the “gatekeeper equation:” (original)(acceptance) = (target)

$$p_{ij} \times \alpha_{ij} = p_{ji}.$$

To illustrate this more general approach to the algorithm, go back once again to a walk on a “triangle with tail.”

Example. Metropolizing a walk on a 4-point graph.
 Review the random walk on the graph in Display 6.6. Because the vertex degrees are unequal, the stationary distribution is not uniform.



vertex degrees: $d_1 = d_2 = 2, d_3 = 4, d_4 = 1$.

$$\text{stationary distribution} = (2 \ 2 \ 3 \ 1)/8 = (d_1 \ d_2 \ d_3 \ d_4)/\sum d_i$$

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{d_1} & \frac{1}{d_1} & 0 \\ \frac{1}{d_2} & 0 & \frac{1}{d_2} & 0 \\ \frac{1}{d_3} & \frac{1}{d_3} & 0 & \frac{1}{d_3} \\ 0 & 0 & \frac{1}{d_4} & 0 \end{bmatrix}$$

Display 6.6 Random walk on the walk on a “triangle with tail”

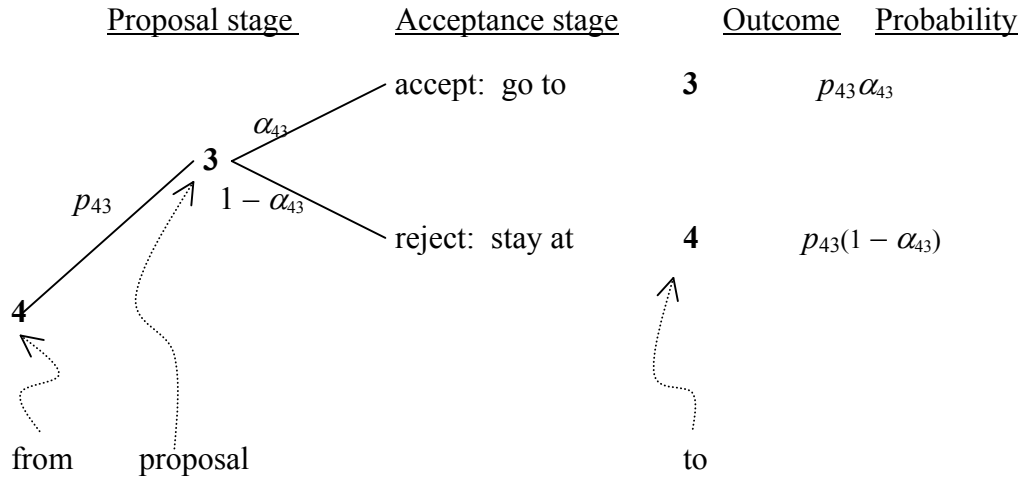
In order to achieve a uniform equilibrium distribution, you have to modify the transition matrix to make it symmetric. According to Step 1 of the boxed summary, to do this you identify entries in the transition matrix that are “too big”, and “shift” some of the excess probability into self-loops. (Check that in this particular example, all the entries of \mathbf{P} that are too big belong to column 3.)

- $p_{13} = 1/2$ is too big. For symmetry, it should be $1/3$.
 Keep $1/3$, and shift the excess $1/6$ to p_{11} .
- $p_{23} = 1/2$ is too big. For symmetry, it should be $1/3$.
 Keep $1/3$, and shift the excess $1/6$ to p_{33} .
- $p_{43} = 1$ is too big. For symmetry, it should be ____.
 Keep ____, and shift the excess ____ to p_{44} .

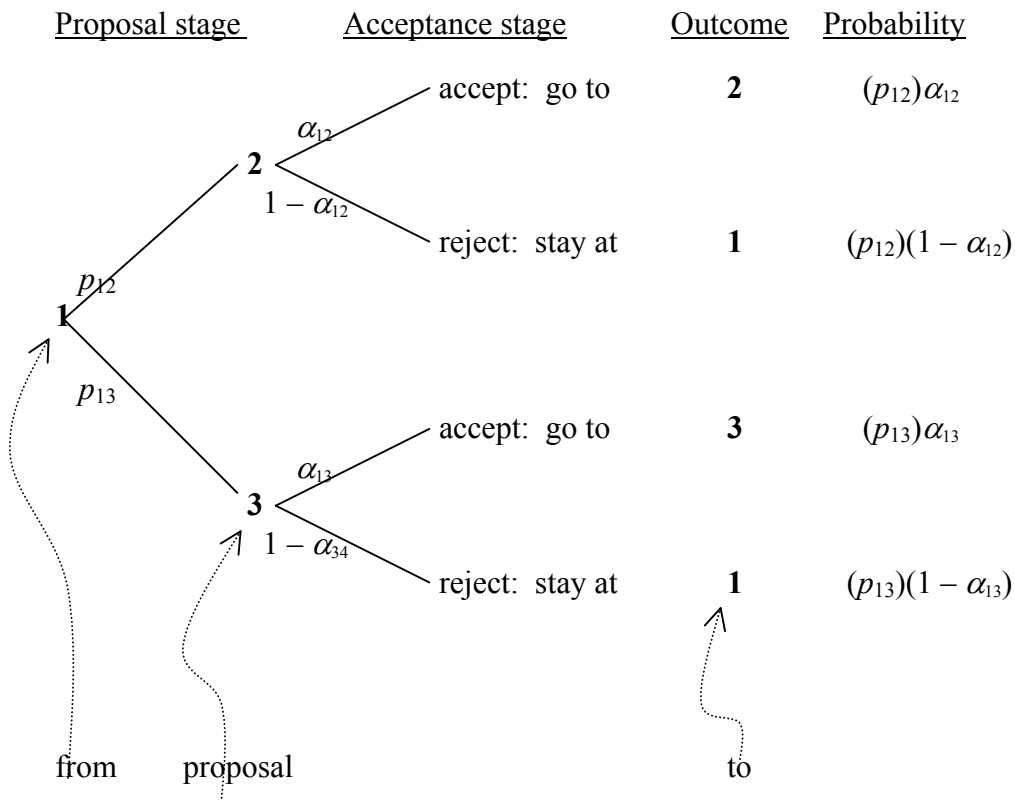
Drill:

41. Fill in the blanks for p_{43} above.
42. Write the (symmetric) transition matrix that results from these shifts.
 Shifting probability in this way gives a symmetric transition matrix $\tilde{\mathbf{P}}$ with uniform stationary distribution. That takes care of the theory, but implementation is another

matter. That's what Step 2 is about. In order to use what we have done in an actual simulation, it must be possible to accomplish the probability-shifting by means of Metropolis coin flips. Display 6.7 shows tree diagrams for parts of the Metropolized walk, with proposal probabilities determined by the original transition matrix \mathbf{P} , and the unknown acceptance probabilities α_{ij} on the branches for the acceptance stage. The top panel (6.7a) is for the simpler case, a move from vertex 4. The bottom panel is for a mover from vertex 1.



Display 6.7a Metropolized moves starting from vertex 4.



Display 6.7b Metropolized moves starting from vertex 1.

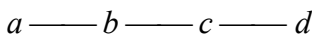
Drill exercises:

43. Consider a walk on the three-point linear graph: $a \text{ --- } b \text{ --- } c$

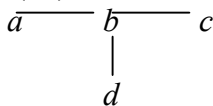
- a. Write the transition matrix.
- b. List the transitions that are too frequent, i.e., the pairs (i,j) for which $p_{ij} > p_{ji}$. For each such pair, compute the excess probability $(p_{ij} - p_{ji})$. Rewrite the transition matrix, shifting all the excess probability to the diagonal (adding the excess to p_{ii}), and thereby reducing p_{ij} so that it has the same value as p_{ji} .
- c. Verify that for the new transition matrix $\tilde{\mathbf{P}}$, the vector $\mathbf{u} = (1/3, 1/3, 1/3)$ satisfies $\mathbf{u}\tilde{\mathbf{P}} = \mathbf{u}$, i.e., there is a uniform stationary distribution.
- d. For each \tilde{p}_{ij} that is less than the value p_{ij} in the original chain, find the value of the acceptance probability α_{ij} that satisfies the gatekeeper equation (original)(acceptance) = (target).

44 - 47. Metropolize walks on the following graphs, that is, determine acceptance probabilities (for all the transitions) for which the resulting walk has a uniform stationary distribution.

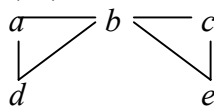
(44)



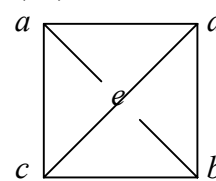
(45)



(46)



(47)



48. Show that your formula in (39) satisfies the gatekeeper equation.

Discussion

49. Give an example of a Markov chain for which the Metropolis algorithm “won’t work”, that is, for which no set of acceptance probabilities will give uniform limiting probabilities.

Investigations

50. Generalize from your example in (49): Find a rule for deciding whether a transition matrix can be Metropolized to give a uniform limiting distribution.

51. Restate your rule from (50) in the form of a theorem, and prove it.

6.4 Metropolis Algorithm for Independent Samples: The Independence Sampler

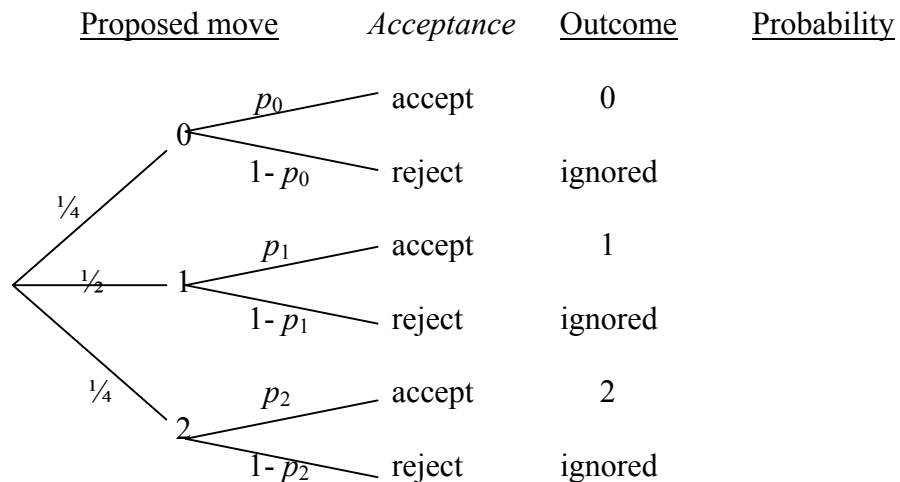
In Section 6.2 you saw how to “Metropolize” a graph walk so that it gives limiting uniform probabilities. Then, in Section 6.3 you saw that the Metropolis algorithm can be applied not just to graph walks, but to a much more general class of Markov chains. One particularly important sub-class of particularly simple Markov chains consists of the ones whose transition probabilities don’t depend on the current state of the chain. You’ve already seen an example: the string activity. For that chain, at each step (each time you draw out a string) the probabilities for the various lengths are the same as they were for the previous draw. Each draw is independent of all previous draws. An exercise (60) will ask you to Metropolize this chain. First, by way of preparation, here is a simpler example based on coin tossing.

Example. Metropolizing the number of heads in two tosses of a coin. Imagine that you have a fair coin, and that you want to use it to generate random numbers 0, 1, and 2 with uniform probabilities. If you toss the coin twice and count the number of heads, you get the random numbers 0, 1, 2, but their probabilities⁵ are $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{1}{4}$ rather than $\frac{1}{3}$, $\frac{1}{3}$, $\frac{1}{3}$. In what follows, you will implement three ways to adjust so that you get uniform probabilities.

Discussion question.

52. The scheme above defines a Markov chain with states 0, 1, and 2. Explain why the rows of the transition matrix for this chain are equal. Then write the matrix, and verify that $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ is a stationary vector.

Russian roulette. Out of every four pairs of tosses, you can expect one 0, two 1s, and one 2. If your goal is a uniform distribution, the 1s occur twice as often as they should, so you simply reject half of them: Whenever you get a 1, toss a coin; if it lands heads, ignore the 1; if the coin lands tails, record the 1.



Display 6.8 Rejecting to get uniform probabilities from the toss of two coins

⁵ $P(0) = P(TT) = (1/2)(1/2)$; $P(1) = P(HT) + P(TH) = 2(1/2)(1/2)$; $P(2) = P(HH) = (1/2)(1/2)$.

Discussion question:

53. What are the probabilities that go on the branches labeled “accept” and “reject” in Display 6.8? Use these to fill in the probabilities in the right-most column, and verify that 0, 1, and 2 are recorded with equal probabilities.

Waiting. On average, over the long run, out of every four pairs of tosses you can expect one 0, two 1s, and one 2. This makes 0s and 2s under-represented, and we can use holding in place to give them additional presence. Whenever you get a 0 (or a 2), toss a gatekeeper coin with $P(\text{Heads}) = \frac{1}{2}$. If it lands heads, “hold in place” by repeating the 0 (or the 2); then toss the gatekeeper coin again. As soon as the gatekeeper coin lands tails, go back to tossing a pair of coins and counting the number of heads.

Discussion questions:

54. The scheme just described defines a Markov chain with states 0, 1, and 2. What are the starting probabilities?

55. Consider transitions from 1 to each of 0, 1, and 2. These transitions do not involve tossing a gatekeeper coin. What are p_{10} , p_{11} and p_{12} ?

56. Now consider the transitions from 0, which do involve tossing a gatekeeper coin. (a) Explain why $p_{01} = \frac{1}{4}$. (b) Find p_{02} . (c) If the current state is 0, there are two ways the next state could also be 0. Describe them, and add their probabilities to get p_{00} .

57. Write the transition matrix, and verify that it has a uniform stationary distribution.

Metropolizing. Review the situation so far. We have a way to generate 0s, 1s, and 2s, but the 1s occur twice as often as the 0s and 2s. Russian roulette deals with this by randomly killing half the 1s. Holding in place deals with it by randomly doubling the number of steps you spend at each 0 or 2.

The Metropolis algorithm only alters particular transitions – those for which $p_{ij} > p_{ji}$. For our example, there are only two of these transitions: from 0 to 1 and from 2 to 1. For each of these transitions, install a gatekeeper, the Metropolis coin flip: Toss a fair coin; if it lands heads, hold in place; if it lands tails, make the move to 1.

Discussion questions.

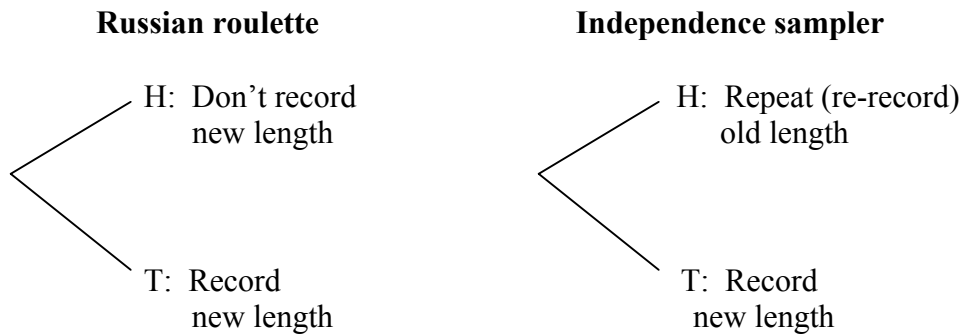
58. Use the transition matrix from (52) to verify that the transitions from 0 to 1 and from 2 to 1 are the only ones for which $p_{ij} > p_{ji}$.

59. Consider the transition matrix for the Metropolized coin tossing scheme. (a) Explain why five of its nine transition probabilities are the same as the ones in (52). (b) Next consider the two transitions (0 to 1 and 2 to 1) that involve the Metropolis coin flip. Find their probabilities. (c) Finally, consider the other transitions affected by the Metropolis

coin flip, from 0 to 0 and from 2 to 2. Find their probabilities. (d) Verify that the transition matrix has a uniform stationary distribution.

Exercise: String Theory revisited

60. Reread the activity at the beginning of the chapter, and note that here, too, the Russian roulette strategy of Exercise 10 is inefficient, in that you end up throwing away data. The Metropolis algorithm provides an alternative strategy: Each time you choose a string and toss a coin, instead of using the outcome of the toss to decide whether or not to record the length, use it instead to decide *which* length to record, the current one, or a duplicate of the previous one. (See Display 6.9.)



Display 6.9 Russian Roulette and the Independence Sampler compared

Suppose the current length is l_{new} , and the previous length is l_{old} . Find a formula for $P(Tails)$ in terms of the two lengths, with the property that it gives all strings the same chance of being recorded. (Here $P(Tails)$ is the acceptance probability.)

Investigation: Limiting probabilities (continued)

61. So far, you've used Metropolis coin flips to convert a non-uniform stationary distribution to uniform. Suppose now you have a Markov chain with a uniform stationary distribution, and you want some other distribution $\mathbf{q} = (q_1, q_2, \dots, q_k)$ as the stationary distribution. Find a formula for acceptance probabilities α_{ij} in terms of q_i and q_j so that the Metropolized chain with transition matrix $\tilde{\mathbf{P}} = \{\tilde{p}_{ij}\}$, $\tilde{p}_{ij} = \alpha_{ij} p_{ij}$, has stationary distribution \mathbf{q} .

62. Next, suppose you have a Markov chain with known but non-uniform stationary distribution $\mathbf{p} = (p_1, p_2, \dots, p_k)$, and you want $\mathbf{q} = (q_1, q_2, \dots, q_k)$ as the stationary distribution. Find a formula for acceptance probabilities α_{ij} in terms of p_i, p_j, q_i and q_j , so that the Metropolized chain with transition matrix $\tilde{\mathbf{P}} = \{\tilde{p}_{ij}\}$, with $\tilde{p}_{ij} = \alpha_{ij} p_{ij}$, has stationary distribution \mathbf{q} .

Investigation: Convergence rates

63. You now have formulas for a variety of situations:

Type of chain	Limiting probabilities		Acceptance probability α_{ij}
	Original	Metropolized	
Graph walk	$d_i / \sum d_i$	uniform	
General chain	p_i	uniform	
General chain	uniform	q_i	
General chain	p_i	q_i	

- a. Fill in the column of acceptance probabilities.
- b. Show that the formulas for the first three situations are special cases of the formula for the fourth.

64. In (57) and (59) you found transition matrices for two versions of the coin-tossing scheme for generating random 0s, 1s, and 2s. The first of these (holding) used dumb gatekeepers; the second (Metropolis) used smart gatekeepers. Compare the convergence rates for the two Markov chains: Is smart faster than dumb?

65. Use various small graphs, such as those in (44) – (47), to compare the best dumb gatekeepers with Metropolized walks: For each graph you look at, find the range of holding probabilities that result in a uniform stationary distribution. Then find the set of these probabilities that gives fastest convergence. Finally, compare the convergence rate for the fastest set of holding probabilities with the convergence rate of the Metropolized graph walk. Does Metropolis always beat the fastest set of dumb holding probabilities?

66. Smart versus dumb. Complete the following summary table for graph walks by filling in the bottom row of formulas for transition probabilities p_{ij} in terms of adjacencies a_{ij} and vertex degrees d_i and d_j :

Probability	No gatekeeper	Dumb gatekeeper	Smart gatekeeper
p_{ii}	0	$1 - d_i / \max_i \{d_i\}$	$1 - \sum_{j \neq i} \tilde{p}_{ij}$
$p_{ji}, j \neq i$			

The Finch connection: Metropolizing swap walks on co-occurrence matrices

Exercises 67 – 70: For each of the co-occurrence matrices shown below, Metropolize the graph walk on the set of 0,1 matrices whose row and column sums are the same. (The graph for (67) has five vertices, that for (68) seven, (69) six, and (70) twelve.

(67)

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

(68)

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

(69)

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

(70)

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