Introduction to Discrete MCMC for Redistricting

Daryl DeFord

May 30, 2019

1 Introduction

This is intended as a friendly introduction to the ideas and background of Markov Chain Monte Carlo (MCMC) sampling methods, particularly for discrete state spaces. Applications to political districting have created a renewed interest in these methods among political scientists and legal scholars and this introduction aims to present the underlying mathematical material in an interactive and intuitive fashion.

We will start with a review of some basic terminology from probability, before discussing Monte Carlo methods and Markov chains separately. Next, we will see how these methodologies combine to form one of the most useful algorithms ever invented. Finally, we will conclude by exploring the application of these methods to generating large ensembles of political districting plans, an approach that has been successful both in court challenges to gerrymandered maps, as well as reform advocacy.

1.1 Sage Examples

Throughout this document there are many links to interactive tools for experimenting with the concepts that are introduced. A web page organizing all of the interactive elements can be found here and all of the source code is available on GitHub. Each tool consists of an interactive terminal that allows you to vary the input values and provides many different types of plots and visualizations. This is a great opportunity to experiment with these concepts and build some extra intuition around the ideas.

2 Probability Background

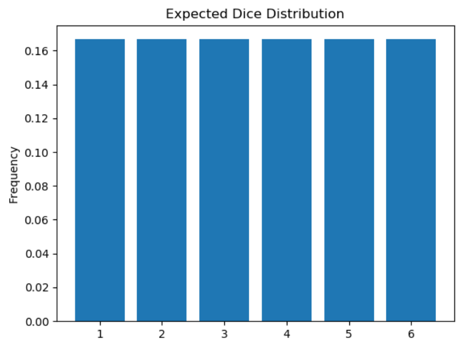
This section provides a brief introduction to some terminology and ideas from probability that we will use throughout the rest of the piece. If this feels like familiar material, you should feel free to skip ahead to Section 3 below. More detailed background information can be found in Grinstead and Snell's Introduction to Probability (.pdf link) if this inspires you to read more.

2.1 Distributions and Random Variables

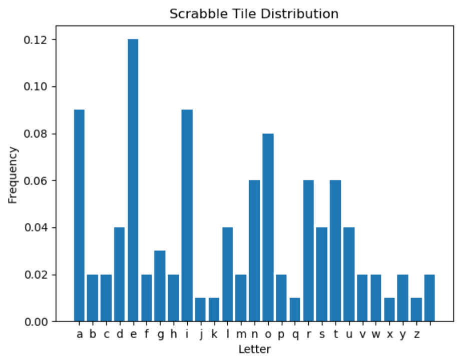
A *probability distribution* is a function that assigns a probability or likelihood to each of a collection of possible outcomes or states. An example is the result of rolling a die, where each face has an equal chance of being on top when the die stops moving. This is an example of a uniform distribution, where each outcome has exactly the same probability of occurring. An example of a non-uniform distribution is drawing a scrabble tile from the bag - there are 12 'E' tiles and only 3 'G' tile so we are 4 times as likely to draw an 'E' as a 'G'.

We will use the terminology *state* to refer to a possible outcome of our random variable and *state space* to refer to the full collection of states. Thus, for the example of choosing a scrabble tile[1], the state space is the set of alphabet tiles ('A' – 'Z' plus the blank tile ' '). A *random variable* is a function that maps elements of the state space to their probabilities. So the . We will summarize these visually using *histograms*, where the x–axis represents the individual elements of the state space and the height of the bars represent their probability of occurring. Figure 1 shows the histograms corresponding to the rolling a die and drawing a scrabble tile.

[1]If you are unfamiliar with Scrabble, see Section 2 below



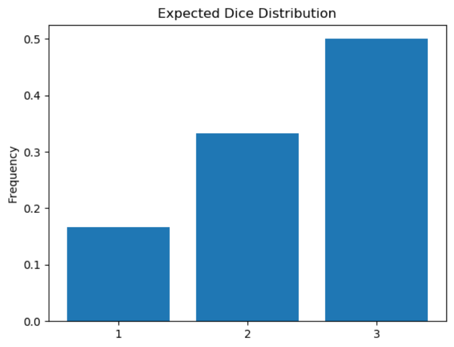
(a) Rolling a die



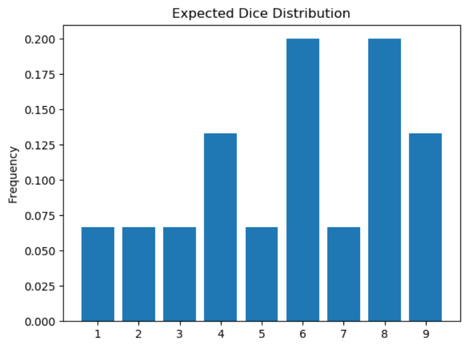
(b) Drawing a scrabble tile

Figure 1: Example Histograms. All the faces of the die are equally likely, so the random variable has a uniform distribution over all the states and all the bars are the same height. On the other hand, there are more 'A' tiles than 'Z' tiles in Scrabble, so the probability of drawing an 'A' is higher than drawing 'Z'.

Example 1 (Non–uniform Dice)**.** *The die we were considering above is a little boring since each face just appears once. We might instead consider what would happen if we made dice with more faces or with repeated faces. The interact module here: here will let you experiment with the distributions that you can generate by designing your own face values. Figure 2 shows the histograms for some non–standard dice.*



(a) 1,2,2,3,3,3



(b) 1,2,3,4,4,5,6,6,6,7,8,8,8,9,9

Figure 2: Histograms for Non–standard Dice. If you vary the number of faces or the values on the faces of a die, this changes the probability distribution of the values that you will see. Compared to the standard die in Figure 1(a) the first die (a) still has six faces but only three values with different frequencies while the die represented by plot (b) has 15 faces, again with varying frequencies.

2.2 Expected Value

The *expected value* of a random variable is a weighted average of the values of the state space, where the weights are given by the probabilities of the individual states. This means that for each state, we multiply its value by its probability of occurring and then add them all up. For rolling a die, the expected value is 3.5 since each state occurs with equal probability:

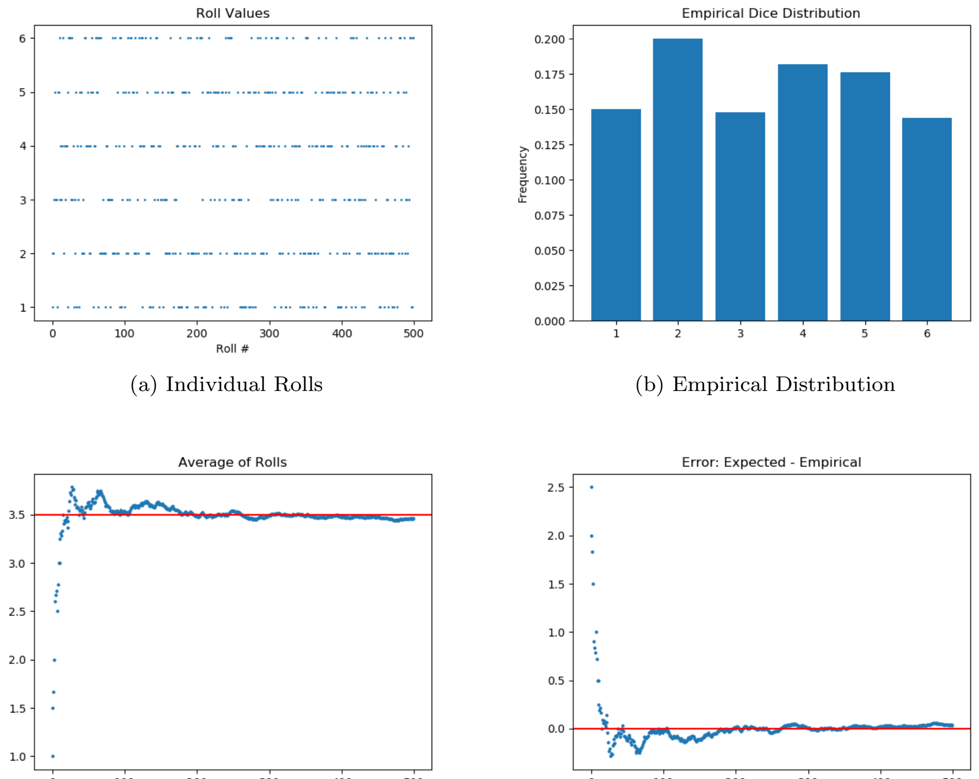
$$\frac{1}{6}\cdot 1+\frac{1}{6}\cdot 2+\frac{1}{6}\cdot 3+\frac{1}{6}\cdot 4+\frac{1}{6}\cdot 5+\frac{1}{6}\cdot 6=\frac{21}{6}=3.5$$

Notice although we will never actually roll a 3.5 on a 6 sided die, it does represent a type of average value if we rolled the die many times and kept track of the results. To formalize this notion, Figure 3 below shows the result of exactly this process by simulating rolling a die times and then measuring both the actual probabilities of each value that occur and the average value across all the rolls over time. Notice that as we continue to roll, the average gets closer and closer to the expected value. Another way to express this is that the *error*, or difference between the empirical average and theoretical expected value, is heading towards zero the more times we roll the dice.

The same calculation approach applies when the probabilities are not equal, it just changes the weights on the values. For example, if we die whose faces are 1,2,2,3,3,3 like in Example 1 the expected value of a roll is:

$$\frac{1}{6}\cdot 1+\frac{2}{6}\cdot 2+\frac{3}{6}\cdot 3=\frac{14}{6}$$

To try this out by computing simulations for the expected values your own dice examples you can use the code here. Try to compute these values for the non–standard dice introduced in Example 1. The other experiment to try is to see how the results change as the number of rolls increases. We will discuss this idea of using more attempts to get better accuracy more fully in the next section but it is good to think about how many steps it takes to get the empirical distribution to look like the theoretical distribution.



(a) Individual Rolls

(b) Empirical Distribution

(c) Average Roll Value

(d) Error

Figure 3: Expected value of 500 rolls of a regular die. Part (a) shows the individual roll values and (b) shows the histogram of those rolls. Notice that even though the theoretical expectation is uniform there are still differences between the number of times the values occurred in our experiment. Plot (c) shows how the average value of the rolls changes as the experiment progressed, with a red line showing the theoretical expected value of .5 while (d) shows that the difference between the experimental value and the theoretical value goes to zero.

Example 2 (Scrabble Tiles)**.** *The game of Scrabble uses 100 square tiles that are drawn from a bag. Each tile is labelled with a letter (or a space) and a number, which represents the score of the tile. The number of tiles and thee score of each letter are displayed in the table below:*

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Frequency	9	2	2	4	12	2	3	2	9	1	1	4	2	6
Score	1	3	3	2	1	4	2	4	1	8	5	1	3	1
Letter	O	P	Q	R	S	T	U	V	W	X	Y	Z	' '	
Frequency	8	2	1	6	4	6	4	2	2	1	2	1	2	
Score	1	3	10	1	1	1	1	4	4	8	4	10	0	

Table 1: Frequencies and point values of Scrabble tiles.

The score of a word in Scrabble is just the sum of the scores of the corresponding letters. For example, the word "RANDOM" has score:

$$1 + 1 + 1 + 2 + 1 + 3 = 12.$$

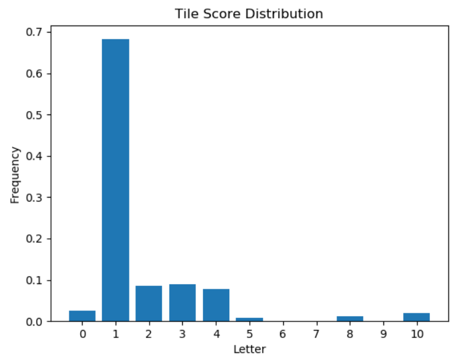
Since we are mathematicians, we will usually use the word "word" in a more general sense to mean any string of letters (and spaces). So for us, the string 'AHC PF VD' will be a 9 letter word with score:

$$1 + 4 + 3 + 0 + 3 + 4 + 0 + 4 + 2 = 21.$$

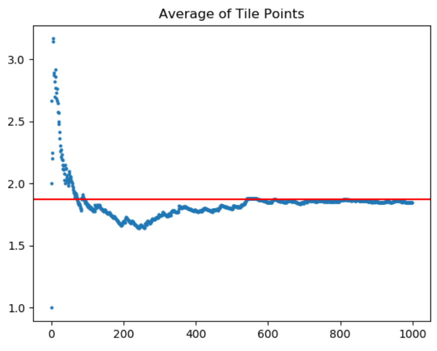
This fact that the tiles have scores allows us to compute the expected value (score) of a randomly drawn tile:

$$\frac{9}{100}\cdot 1+\frac{2}{100}\cdot 3+\frac{2}{100}\cdot 3+\frac{4}{100}\cdot 2+\frac{12}{100}\cdot 1+\frac{2}{100}\cdot 4+\frac{3}{100}\cdot 2+\frac{2}{100}\cdot 4+\frac{9}{100}\cdot 1+\frac{1}{100}\cdot 8+\frac{1}{100}\cdot 5+\frac{4}{100}\cdot 1+\frac{2}{100}\cdot 3+\frac{6}{100}\cdot 1+$$
$$\frac{8}{100}\cdot 1+\frac{2}{100}\cdot 3+\frac{1}{100}\cdot 10+\frac{6}{100}\cdot 1+\frac{4}{100}\cdot 1+\frac{6}{100}\cdot 1+\frac{4}{100}\cdot 1+\frac{2}{100}\cdot 4+\frac{2}{100}\cdot 4+\frac{1}{100}\cdot 8+\frac{2}{100}\cdot 4+\frac{1}{100}\cdot 10+\frac{2}{100}\cdot 0 =$$
$$\frac{187}{100} = 1.87$$

A Scrabble version of the expected value experiment, using the tile values for scores, is presented below in Figure 4. Comparing the table to the histogram, we can check that there are many more tiles with score 1 in the scrabble bag than any other value, which is reflected in our experiment. Notice that again after about 500 steps the empirical average has converged to very near the expected value of 1.87. You can design your own experiments using Scrabble tiles with the code here.



(a) Empirical Distribution



(b) Expected Value

Figure 4: Results of simulating drawing 1,000 Scrabble tiles (with replacement). The distributions observed in this experiment are quite close to those calculated theoretically.

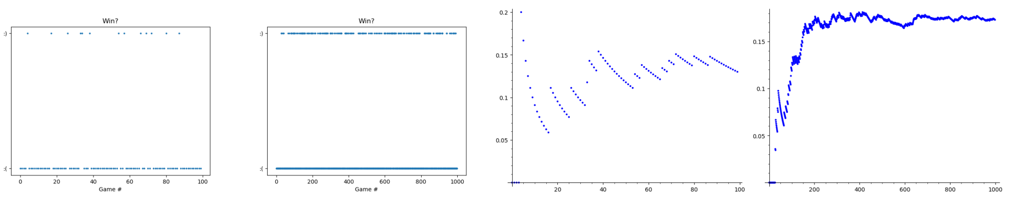
3 Monte Carlo Sampling

Monte Carlo methods are built to take advantage of the fact that some things are hard to compute exactly but easy to evaluate individual examples. This idea was formalized as a result of Stan Ulam's work in the Manhattan project but examples of these approaches for estimating computationally intensive values go back hundreds of years. Ulam was originally wondering about the probability of winning a particular type of non–deterministic solitaire game[2] and realized that he could estimate the probability, using the computer to shuffle the deck and evaluate whether or not the game was winnable.

To imagine a simple version of this, consider a "game" where you are presented with a shuffled deck of cards and win if the top three cards are in increasing order and lose otherwise. What is the probability that you win with a randomly shuffled deck? In principle, we could try to compute all the ways to form a 3-card increasing sequence and divide by all the ways to shuffle the deck:

$$52! = 80658175170943878571660636856403766975289505440883277824000000000000$$

While this seems hard in general, for any given shuffle it is easy to check whether or not you won – just look at the top three cards. This suggests a possible method for estimating the actual win rate, shuffling the deck many times and just computing whether or not the top three card increase. Figure 5 shows an example of this experiment and you can try it yourself here here. Notice that this is similar to what we saw in the previous section, where rolling more dice (or drawing more tiles) made our empirical estimate of the expected value better.



(a) 100 games (b) 1000 games (c) 100 games (d) 1000 games

Figure 5: Estimating the win rate of a simple, non–deterministic solitaire game. Plots (a) and (b) shows whether or not each individual game was won, while (c) and (d) shows the average win rate across all of the games. As the number of steps increases, the win rate appears to converge to a final value of approximately .165.

The same general outline that we applied to solitaire is common to most examples of Monte Carlo methods. Extracting the important steps, we get an algorithm that looks something like:

1. Draw an (independent) sample from the set of all possibilities
2. Compute some measure for each sample
3. Repeat many times
4. Average/aggregate the derived data

Notice that this is exactly the approach that we used in the previous section to compute expected values for dice rolls and scrabble draws in Section 2.2. In both cases, it is easy to repeatedly draw the samples and measure the values and the final estimate converged rapidly to the correct theoretical value that we calculated. The fact that it is usually simple to apply this approach to get good estimates of empirical values has led to it becoming one of the most common techniques in the numerical analysis tool kit. The examples below explore a couple of other specific applications but every field that has a numerical component makes use of this methodology at some point.

[2] A non–deterministic game is one in which the player cannot make any choices that impact the outcome of the game. Whether or not these should actually be called games is a philosophical question, not a mathematical one.

Example 3 (Distances in a cube)**.** *Although these steps seem abstract we can apply them to a seemingly simple example: What is the expected distance between two points randomly drawn in a unit cube? Although this problem has a mathematical formulation*

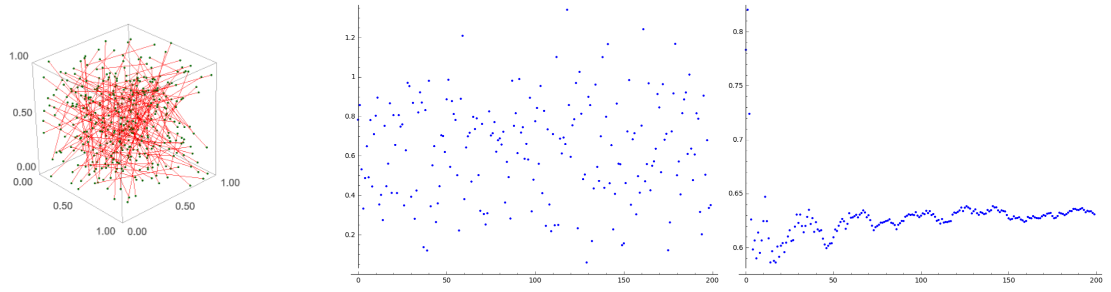
$$\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1 \sqrt{((x_1-y_1)^2+(x_2-y_2)^2+(x_3-y_3)^2}dx_1dx_2dx_3dy_1dy_2dy_3$$

and a mysterious looking exact solution

$$\frac{4+17\sqrt{2}-6\sqrt{3}+21\log(1+\sqrt{2})+42\log(2+\sqrt{3})-7\pi}{105}$$

this is a perfect problem to try out the Monte Carlo method[3]*. Trying to solve the problem directly would require us to consider how to compare arbitrary pairs of points but it is simple for us to instead select pairs points of uniformly in the cube – this is our method for sampling from the possible inputs (1). Then, it is easy to measure the distance between each pair of points that we select – this is our measurement of the value of our sample (2). Finally, we repeat this 1,000 times (3) compute the average across all of our trials (4).*

These steps are summarized in Figure 6. In (A) we see 1000 pairs of points connected by lines. Part (B) shows the lengths of each line individually, these look randomly scattered in space but part (C) shows the underlying structure of the average of the points. Although the distances between individual pairs can be quite far from the expected value, the average converges quite rapidly to $\sim 0.66233348...$ *which is just a hair smaller than the actual value of* $\sim 0.662959....$ *If we continued to select pairs of points this average would continue to get closer and closer to the true value.*



(a) n pairs of points (b) Distances between pairs (c) Average distance between pairs

Figure 6: This figure shows the steps of the Monte Carlo process for estimating the average distance between arbitrary pairs of points in a cube. Plot (A) shows 1000 randomly chosen pairs of points, plot (B) shows the distances between those pairs and plot (C) shows the cumulative average of the distances. Notice that even though the pairwise distances seem to be random, the average converges rapidly to the correct value.

Example 4 (Estimating π)**.** *Another example*[4] *is estimating the value of* π*, using the area of a circle. In this case, we can select individual points uniformly in the unit square (1) and for each point it is easy to tell whether it is inside or outside the circle (2). Again, we repeat this 1,000 times (3) and divide the number of points that landed inside the circle by the total (4). This example is shown in Figure 7. Part (A) shows 1000 random points in the unit square while (B) shows the running proportion of those points that land inside the circle converging to* π*.*

[3]An interactive widget for exploring this problem is here.
[4]An interactive widget for exploring this problem is here.

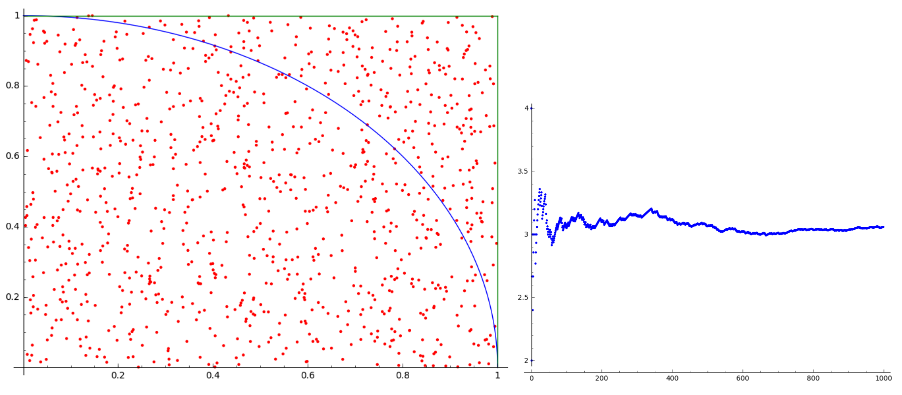

(a) 1000 uniform points in the unit square (b) Proportion of points inside circle

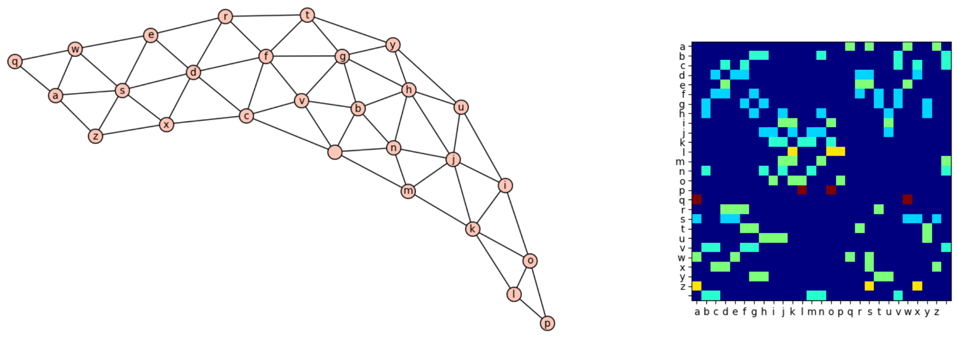
Figure 7: This figure shows a Monte Carlo experiment for estimating π. We start by drawing 1,000 random points in the plane (A) and then calculate the proportion of those that lie within the circle (B).

4 Markov Chains

A Markov chain is a special sequence of random variables, where the distribution of each variable only depends on the outcome of the previous variable not any of the earlier outcomes. Examples include children's games such as Chutes and Ladders and Monopoly, where the probability of landing on a particular square on your turn is completely determined by your current square, as well as more complex processes such as Google's original PageRank algorithm, which models the behavior of a random web–surfer moving from page to page by clicking hyperlinks.

The simplest case is when each variable is drawn from the same distribution, i.e. there is no dependence on the previous values at all. This is the case for our dice and scrabble examples - as long as we roll the same die or replace the tile that we drew at the previous step, the distribution is the same for every experiment. In this case, every random variable is the same, so it definitely doesn't depend on the output of the previous process and hence satisfies the Markov condition.

For an example where the variables do depend on the previous outcomes, consider an ant walking on a keyboard. Our state space will be the individual letters and the space bar.At each step, the ant can move from its current key to an adjacent one, chosen uniformly from those neighbors. For example, if the ant is on the 'E' key it can move to any of 'W', 'S', 'D', or 'R' with probability $\frac{1}{4}$ for each while if it is on 'H' it could move to any of 'Y', 'G', 'B', 'N', 'J', or 'U' with probability $\frac{1}{6}$. Figure 8 shows the letters and their connections and an animation of the ant walking can be viewed here.



(a) Keyboard Adjacency (b) Transition Probabilities

Figure 8: The adjacency structure of a standard keyboard. We can define a Markov chain by moving uniformly between adjacent keys with probabilities shown in (B).

To interpret this as a Markov chain, note that each key corresponds to a particular distribution over its neighbors that only depends on where the any is currently standing, not how it got to that square. Thus, the ant walk satisfies the Markov property, since the possible next steps only depend on the key it is currently standing on. Additionally, for any pair of keys α and β, we can compute the probability that the ant steps from α to β by:

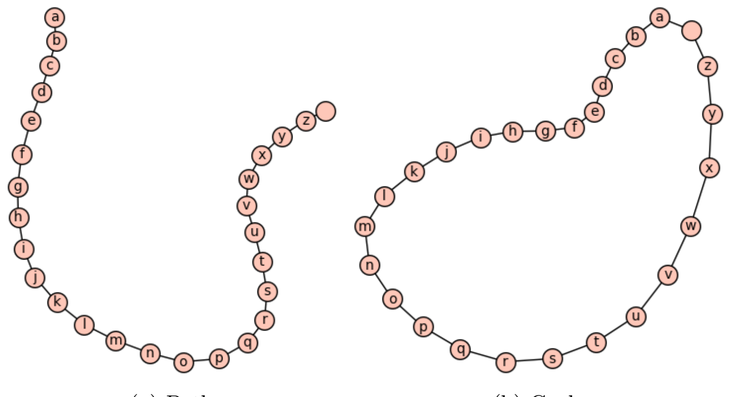
$$\mathbb{P}(\alpha \mapsto \beta) = \begin{cases} \dfrac{1}{\#\text{neighbors of } \alpha} & \text{if } \alpha \text{ is next to } \beta \\ 0 & \text{otherwise.} \end{cases}$$

These *transition probabilities* are enough to determine the full structure of the Markov chain, since for each state, we know exactly what the likelihood is that we will end up in any other state.

All Markov chains (on a discrete state space) can be formulated in a similar fashion. We begin by specifying the set of possible objects, like the squares on the chutes–and-ladders board or the symbols appearing in a text, which are referred to as “states”. Then for each state we describe the probability of transitioning to each other state. This is all the information that is needed to describe a Markov chain. Frequently, these are represented by a directed graph, where each node represents a single state and the directed edges represent the probability of transitioning between the states.

It is common to relate the Markov chain to a “random walk” on this type of digraph. We imagine a walker beginning on an arbitrary vertex and walking along the edges of the graph, visiting one node at each time step. To choose the next state to visit, the walker chooses from among the directed edges leaving the current node, proportional to their weights. To see that this process satisfies our definition of a Markov chain, note that the walker only uses information about the edges leaving the current node to determine his possible transitions, so the random variable describing the walker’s path only depends on the current state. We will return to this terminology several times throughout the remainder of this article.

Example 5 (Alphabet Paths and Cycles)**.** *Again using the alphabet as our state space, we will consider building a couple of particularly simple Markov chains. The first, which we will call the alphabet path, will form a graph where each letter is connected to its neighbors in the alphabet with the space ‘ ’ occurring after the ‘z’. For example, ‘a’ will only be connected to ‘b’, but ‘g’ will be connected to ‘f’ and ‘h’. The second, which we will call the alphabet cycle, will have a similar structure except we will connect the ‘a’ and ‘ ’ states so that the graph forms a ring. Figure 9 shows the two underlying graphs for our Markov chains.*



(a) Path (b) Cycle

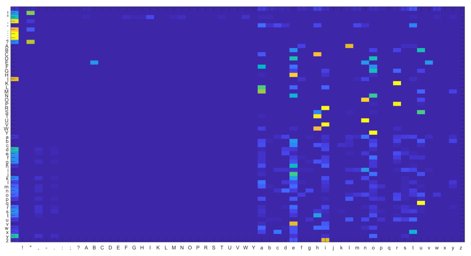
Figure 9: The alphabet path and cycle graphs.

The transition probabilities for these Markov chains are easy to express. For the cycle, the probability of transitioning to any neighbor in the alphabet is exactly $\frac{1}{2}$ *and* 0 *to transition to any non–neighbor. For the path, this is true for every state except for ‘a’ and ‘ ’ for which the transition probabilities are 1 to ‘b’ and ‘z’ respectively and* 0 *for anything else. We will make use of these examples again in the following section when we consider how Markov chains can be used to study distributions over state spaces.*

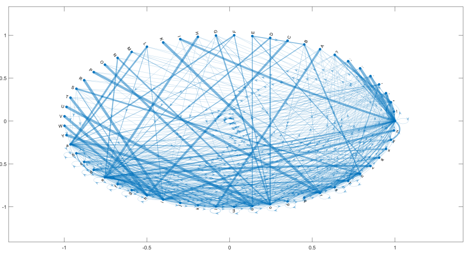
Example 6 (Text Generation)**.** *One of the first applications of Markov chains was the analysis of text passages, trying to predict the next letter that would appear in a book written by a given author. Similar methods are used for auto–complete functions in today's cellphones. Symbols in text are not distributed uniformly, as q is almost always followed by u, periods are followed by spaces, and the letter "e" is most commonly found at the end of a word. Given a long passage of text, we can compute how often each symbol follows each other symbol and use these proportions to generate new text. In this case, a state is the current letter and the probability for choosing the next letter only depends on the current letter.*

While using single characters does not return recognizable words, we can extend our states to include several sequential characters to get more interpretable results. Below are some examples generated using the story "Aladdin and the Magic Lamp" from the Arabian Nights using single characters, pairs of characters, triples of characters, and quadruples of characters. The first line is 50 characters chosen uniformly, for comparison. The single line transition probabilities are shown in Figure 10(a) below while Figure 10 (b) shows the digraph for the symbol transitions in Aladdin.

```
(Uniform) ,Kni;;.RgkY:f;;.?ACKKDFtjaBD-vjaIAezAFO-hOzOe?NAm

(1) y  mpo fewathe he m, main, wime touliance handddd

(2) If ho rembeautil wind was nearsell ith sins. He don the whimsels hed his
 the my mign for atim, but

(3) but powerful not half-circle he great the say woman, and carriage, she sup
window." He said feast father; "I am riding that him the laden, while

(4) as he cried the palace displeasant stone came to him that would not said:
  "Where which was very day carry him a rocs egg, and horseback."
  Then the might fetched a napkin, which were hunting in the
```



(a) Transition Probabilities



(b) Digraph

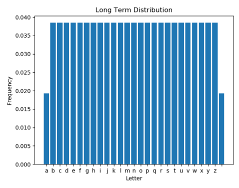
Figure 10: Plot (A) shows the heatmap of the letter to letter transition probabilities in Aladdin and the Magic Lamp. Notice that there is almost no probability of transitioning from a capital letter to another capital and that the most common symbol after "A" is "l" from the title character's name. Plot (B) is the associated digraph with edge widths proportional to importance.

There turns out to be a significant amount of interesting structure in this type of analysis, as simply looking at the transition matrices can frequently be enough to distinguish authors from each other or poetry from prose. Some cleaned collections of copyright free texts, along with some MatLab code for extracting the transition probabilities and analyzing the corresponding Markov chains, can be downloaded here. A guide to using the code is here and an essay prompt making use of this data that I have assigned in classes on mathematical modeling can be seen here.

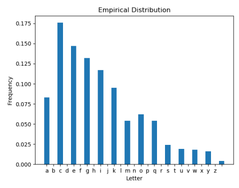
4.1 Markov Distributions

In our introduction to Markov chains above, we mostly focused on the trajectories of individual samples from the chain, like the actual keys stepped on by an ant or the specific letters drawn from a text. To connect these chains to our discussion of probability distributions in Section 2 we first observe that we can use the transition probabilities to compute the distribution over the state space at each step of the Markov chain. That is, while previously we imagined the ant on the 'q' key flipping a coin and stepping to 'w' if the coin was heads and 'a' if the coin was tails, instead we simply note that at the next step, the ant is at state 'a' with probability $\frac{1}{2}$ and at 'w' with probability $\frac{1}{2}$.

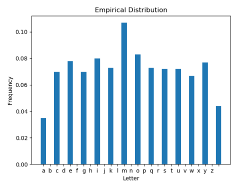
Thinking about Markov chains in this probabilistic way allows us to start to answer questions about where we might expect the ant to be after a hundred or a thousand steps, by assigning a probability to each possible outcome in the state space. Thinking back to Section 3, we might simply suggest running the experiment thousands of times, placing the ant on the 'q' and letting it take a hundred steps and recording the final position. Figure 11 shows an example of this experiment on the keyboard, path, and cycle graphs and you can try it out yourself here.



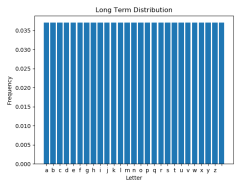
(a) Expected Path



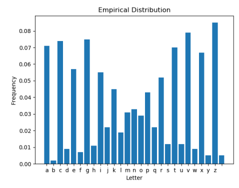
(b) 100 Path Steps



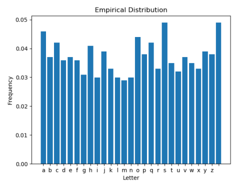
(c) 1000 Path Steps



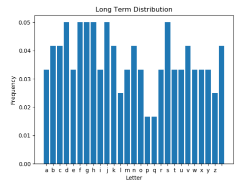
(d) Expected Cycle



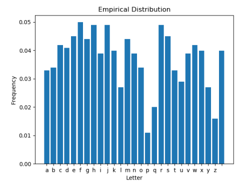
(e) 100 Cycle Steps



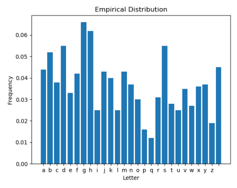
(f) 1000 Cycle Steps



(g) Expected Keyboard



(h) 100 Keyboard Steps



(i) 1000 Keyboard Steps

Figure 11: Example walks on the keyboard, path, and cycle graphs. For each experiment we started at the letter 'a' and took 100 or 1000 steps of the Markov chain, recording the final state reached. For each walk, we repeated the experiment 1000 times and plotted the resulting frequencies. The left column shows the long term expected behavior of the steady state distribution.

A potential issue with this experimental method is that instead of trying to estimate a single number like the expected value, we are trying to estimate a probability for each element of the state space. This gets us right back in to the problem that we were trying to avoid by sampling. Luckily, for our small examples, the alphabet only has 26 letters, so we might feel that we are getting reasonable results after a few thousand steps. However, if we are instead considering all possible $26^7 = 8,031,810,176$ 7 letter words, we should feel less confident. Additionally, the fact that we only observe "even–numbered" letters in the path walk suggests that chain has a special property we should investigate further. We will turn to this in Section 4.1.1 but next we will discuss a technique to determine these long-term probabilities without sampling.

In order to probe this question more generally, imagine that our state space is some set of n items

$S = \{s_1, s_2, \ldots, s_n\}$ and we have computed the transition probabilities between each pair of states $T_{i,J}$. Then, given a probability vector P whose entries P_i tell us the current probability that we are in any particular state i, we can compute the probability that we are at state j in the next step of the Markov chain as:

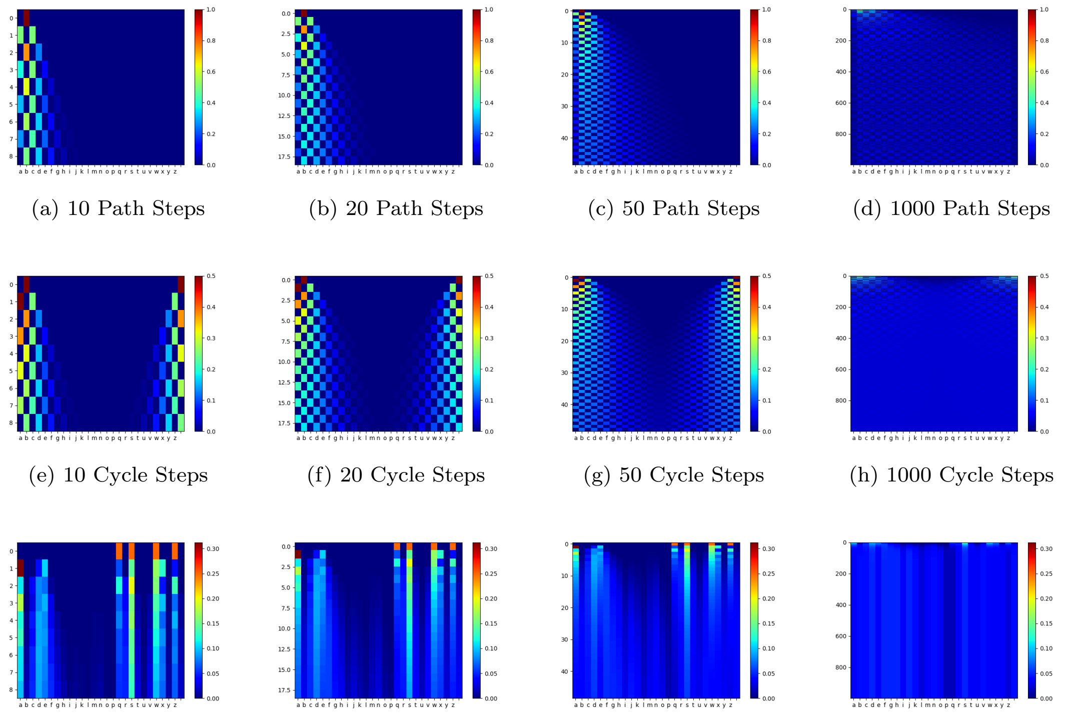
$$Q_j = \sum_{i=1}^{n} P_i T_{i,j}.$$

That is, summing up over all states i, the probability that we were at i and then transitioned to j. If we record the initial state/distribution of our Markov chain as a vector P^0, then we can compute the probability distributions at each successive step recursively with the same operation:

$$P_j^{k+1} = \sum_{i=1}^{n} P_i^k T_{i,j}.$$

This mathematical formalism will help us understand the general properties of Markov chains and provide us with some useful notation. In particular, these probabilities are exactly what we were trying to estimate in Figure 11. For those of you with some exposure to linear algebra, note that this is just the matrix–vector product of the transition matrix with the probability vector.

We can explore this a little more closely using the example chains we introduced in the previous section. For the path, cycle, and keyboard walks, we again begin at 'a' but instead of sampling individual states, we instead use the equation above to compute the exact probabilities of arriving at each other state. These experiments are summarized in Figure 12 below and show the slow diffusion of probabilities across the letters. You can experiment with these examples yourself here. It is particularly enlightening to compare different starting states across the various Markov chains.



(a) 10 Path Steps (b) 20 Path Steps (c) 50 Path Steps (d) 1000 Path Steps

(e) 10 Cycle Steps (f) 20 Cycle Steps (g) 50 Cycle Steps (h) 1000 Cycle Steps

(i) 10 Keyboard Steps (j) 20 Keyboard Steps (k) 50 Keyboard Steps (l) 1000 Keyboard Steps

Figure 12: Probabilistic versions of Figure 11. We compute the probabilities of arriving at each letter in a walk of length k (y–axis) starting at 'a'. Unlike the previous approach, these plots represent the exact probabilities.

These figures highlight a fascinating property of many Markov chains: No matter what initial distribution is used to start the chain, after a sufficiently large number of steps, the distribution will converge to a fixed, *stationary distribution* and continue to remain at that distribution no matter how many further steps are taken. In order to describe the types of chains that have this property, we need to discuss a discuss properties of some Markov chains that make them amenable to analysis.

4.1.1 Periodicity

The first adjective we will consider is *periodicity*. The period of a Markov chain is the greatest common divisor of all possible lengths of cycles that occur in the graph associated to the chain. A chain is said to be *irreducible* if the period is one. Looking at our example chains, we can see that the keyboard is irreducible since 'q'-'w'-'q' is a length 2 cycle and 'q' - 'w' - 'a' - 'q' is a length 3 cycle. Similarly, the cycle chain is aperiodic since 'a'-'b'-'a' is a length 2 cycle and following the entire cycle around is a length 27 cycle. However, the path chain has period 2, since if we number every letter with a=1, b=2, etc. then every step of the chain takes us from an even number to an odd number or vice versa and hence every cycle has even length.

This property explains the "oddness" of the plots in Figure 11 (b) and (c), since we sampled 100 and 1000 step paths, we could only get outputs that were even numbered letters. In order to convert a periodic chain to an aperiodic one we can add a small probability of remaining in place to each step of the walk, since adding length one cycles forces the gcd to be one. A walk with such a probability of not moving at each step is frequently called a "lazy" walk and adding these "waiting probabilities" is a common trick in this setting.

4.1.2 Reducibility

In addition to periodicity, an important feature of Markov chains is *irreducibility*. A Markov chain is irreducible if each state can be reached from any other state in a finite number of steps. All of the examples that we have encountered so far have this property. For example, on the keyboard, an ant starting at any key can reach any other key in at most 9 steps, while on the path, it takes 26 steps to get from 'a' to ' '.

4.2 Ergodicity

Markov chains that are both aperiodic and irreducible are called *ergodic*. These chains have the property that there is a unique *stationary distribution* that is the limit of the probability distributions formed by iterating the Markov chain. A stationary distribution P, satisfies the property that:

$$P_j = \sum_{i=1}^{n} P_i T_{i,j}$$

for each state j. In words, this says that applying the Markov chain (equivalently, the transition probabilities) to the steady state probabilities leaves those probabilities unchanged.

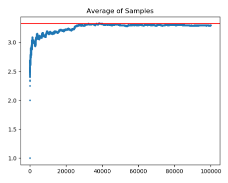
For chains that are formed by simple random walks on graphs, like our path, cycle, and keyboard examples, the probabilities in this distribution are proportional to the number of neighbors of each state in the graph. This explains the left column of Figure 11, the states of the keyboard with the most neighbors and corresponding largest probabilities are s, d, f, g, h, j in the center row, while the keys with the fewest neighbors and smallest probabilities are 'p' and 'q' in the corners. On the other hand, in the cycle, every letter has two neighbors, so the distribution is uniform.

Even better than the existence and uniqueness of stationary distributions, given a function defined on our state space that we are interested in evaluating, if we draw samples according to the Markov chain, the average of the function values evaluated on the samples converges to the expected value over the stationary distribution. We saw some trivial examples of this already with our experiments in Section 2.2, recalling that repeated draws from a fixed distribution form a simple Markov chain.

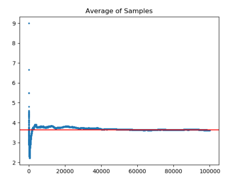
We now define score functions to evaluate on our simple Markov chains:

- **Uniform:** We assign each letter a score of 1.
- **Scrabble Points:** We assign each letter the score that is on the tile, as in Section 2.
- **Scrabble Count:** We assign each letter the number of tiles with that letter that appear in the Scrabble bag, as in Section 2.
- **Alphabetical:** We assign each letter a score based on its position in the alphabet 'a'=1, 'b'=2, ..., 'z'=26, and ' '=27.
- **Vowels:** We assign 1 to each consonant, 100 to each value, and 50 to 'y'.

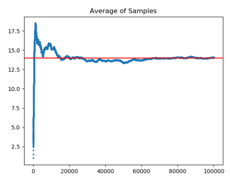
Using these score functions, we can use our simple Markov chains to compute the expected values of the scores with respect to the stationary distribution of the chain. Figure 13 shows these results for the three chains and four non–uniform score functions. The theoretical expected values are shown in red and the empirical values do appear to converge in each of our experiments, verifying the theorem statement referenced above. You can try this out on your own here, it is particularly instructive to adjust the length of the different chains and see how that impacts the accuracy of the final estimate.



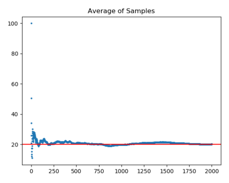
(a) Path –Points



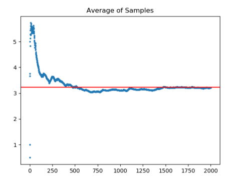
(b) Path – Count



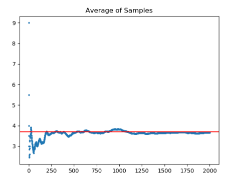
(c) Path – Alphabet



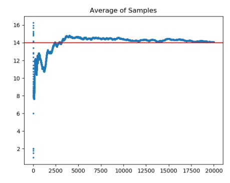
(d) Path – Vowel



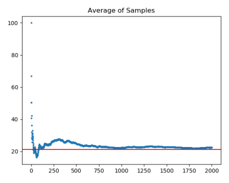
(e) Cycle – Points



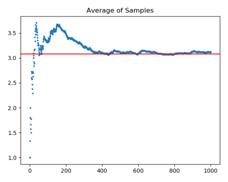
(f) Cycle – Count



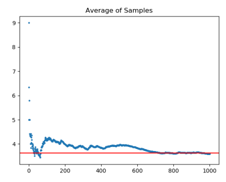
(g) Cycle – Alphabet



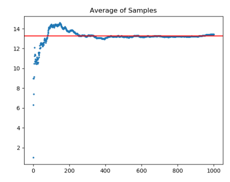
(h) Cycle – Vowel



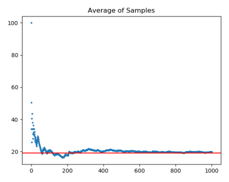
(i) Keyboard – Points



(j) Keyboard – Count



(k) Keyboard – Alphabet



(l) Keyboard – Vowel

Figure 13: Estimating expected values with three walks and four scores. For each of our three simple Markov chains on letters and the Scrabble Points, Scrabble Count, Alphabetical, and Vowel score functions, we use the Markov chain to estimate the expected value of the score across the steady state distribution.

While it is true that the values in each experiment eventually become a good estimate for the actual value, it is not true that each provides the same accuracy given a fixed number of steps. Table 2 below compares the theoretical expected values to the estimates obtained from each Markov chain by increasing the number of samples. These experiments can be reproduced here including the actual values at each step in the chain, as in Figure 3. Note that we should also not expect perfect accuracy over these lengths due to correlation between steps and the effects of the initial samples drawn before reaching the steady state.

Walk	Score	Actual Value	2k steps	10k steps	50k steps	100k steps
Keyboard	Points	3.075	3.129 (1.7%)	3.067 (0.3%)	3.089 (0.4%)	3.072 (0.1%)
Keyboard	Count	3.633	3.733 (2.7%)	3.587 (1.3%)	3.644 (0.3%)	3.638 (0.1%)
Keyboard	Alphabet	13.292	13.12 (1.2%)	13.34 (0.4%)	13.32 (0.2%)	13.30 (0.05%)
Keyboard	Vowels	19.13	21.02 (9.9%)	19.36 (1.2%)	19.53 (2.1%)	18.91 (1.2%)
Cycle	Points	3.22	3.17 (1.7%)	3.18 (1.2%)	3.19 (0.9%)	3.20 (0.7%)
Cycle	Count	3.70	4.00 (8%)	3.88 (4.9%)	3.71 (0.3%)	3.71 (0.2%)
Cycle	Alphabet	14	14.5 (3.5%)	14.32 (2.3%)	14.1 (0.7%)	13.88 (0.8%)
Cycle	Vowels	21.15	21.36 (1.0%)	21.76 (2.9%)	20.97 (0.8%)	21.22 (0.4%)
Path	Points	3.327	3.67 (10.3%)	3.18 (4.5%)	3.33 (0.02%)	3.31 (0.4%)
Path	Count	3.635	3.85 (5.9%)	3.68 (1.2%)	3.69 (1.6%)	3.59 (1.0%)
Path	Alphabet	14	15.75 (12.5%)	13.99 (0.07%)	14.04 (0.3%)	14.07 (0.5%)
Path	Vowels	20.02	18.69 (6.6%)	19.70 (1.6%)	19.29 (3.6%)	20.03 (0.07%)

Table 2: Convergence time comparison for expected value experiments. Percent error in parentheses.

Looking over this data suggests that it takes many more samples from the path chain to get a good estimate for the expected value than for the other walks. The speed at which the Markov chain converges to its stationary distribution from an arbitrary starting point is known as the *mixing time* of the chain. If you experiment with the code provided here you will see that not only does the path require more steps to achieve a particular level of accuracy on average but also displays a higher variance. Mixing times are of great practical importance because they determine the how many samples must be drawn to guarantee good converge of our estimators.

4.2.1 Reversibility

A Markov chain is *reversible* if its steady state distribution satisfies a symmetry condition known as detailed balance. This condition states that in the steady state, the probability of being at state i and transitioning to state j is equivalent to the probability of being at state j and transitioning to state i. In mathematical notation, if P represents the steady state vector and T the transition matrix, then

$$P_i T_{i,j} = P_j T_{j,i}.$$

Reversible chains have many nice properties and this symmetry condition means that the steady state distributions are particularly easy to analyze. Additionally, simple random walks on graphs are automatically reversible. Thus, instead of having to check that a Markov chain satisfies the detailed balance condition above, we can instead construct a graph that has that chain as its simple random walk. The Aladdin text chain in Example 6 is an example of a non–reversible chain.

5 Markov Chain Monte Carlo

It is finally time to put everything together. The key step of MCMC is to create an irreducible, aperiodic Markov chain whose steady state distribution P is a specific distribution we are trying to sample from. The same property that made problems tractable for Monte Carlo analysis (ease of evaluating the properties of a sample) also turns out to be useful for constructing Markov chains to draw from a specific distribution. In our Monte Carlo methods we required that the samples from our space were drawn uniformly but this is not always easy or desirable. Using a Markov chain to select our samples gives us a way to sample from a desired distribution (the steady state of the Markov chain) without having to know the specific probabilities associated to that distribution, just the transition probabilities.

This was the key idea that was exploited by Metropolis and coauthors[5] in 1953 to combine these methods to form what we now call MCMC. As with the original Monte Carlo method, this was yet another collaborative success of the Manhattan project, applied to the statistical mechanics of atomic particles. These

[5]This was another success story from the Manhattan Project

ideas were further developed by Hastings and others and have come to be one of the most fundamental computational tools in all of computer science and statistics. In 2000, the IEEE described MCMC sampling as one of the top 10 most important algorithms of the 20th century.

In most cases, the transition probabilities are determined by a multiple of the distribution that we are trying to sample from. Although this seems like an odd condition: "How could we ever know something proportional to a distribution without knowing the distribution itself?" this turns out to be a common situation in many examples in physics as well as Bayesian Statistics. For our purposes, this arises when we have a score function or a ranking on our state space and want to draw proportionally to these scores i.e. to prioritize states that score "better" under our metrics of interest. This is particularly useful in settings, such as all words of length n where it would be difficult or impossible to compute the probabilities directly.

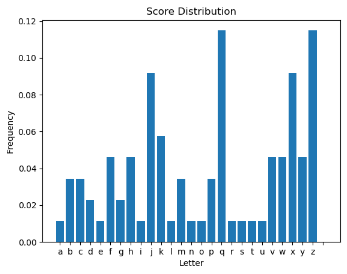
To place this in more mathematical language, imagine that we have a score function s, on our state space X, like the Scrabble values on tiles discussed in the previous section, so that $s : X \mapsto \mathbb{R}$. We then want to sample from the distribution where the states appear proportional to s. That is, element $y \in X$ should appear with probability

$$\mathbb{P}(y) = \frac{s(y)}{\sum_{x\in X} s(x)}.$$

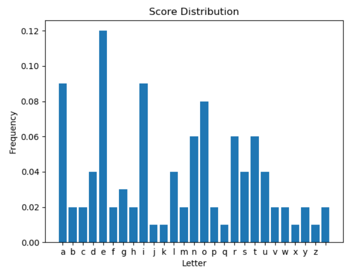
Figure 14 shows these distributions for our four score functions. When X is very large, there is no way for us to compute this denominator directly. However, notice that we can compute ratios of probabilities, since the denominators cancel:

$$\frac{\mathbb{P}(z)}{\mathbb{P}(y)} = \frac{\dfrac{s(z)}{\sum_{x\in X} s(x)}}{\dfrac{s(y)}{\sum_{x\in X} s(x)}} = \frac{s(z)}{s(y)}.$$

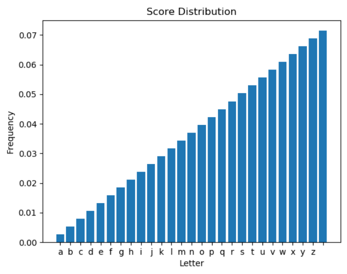
This is the trick that turns out to allow us to draw samples according to s without having to compute the denominator directly. Even more interestingly, Metropolis–Hastings MCMC works by transforming samples drawn using a known Markov chain into samples from a chain whose stationary distribution is proportional to our score function.



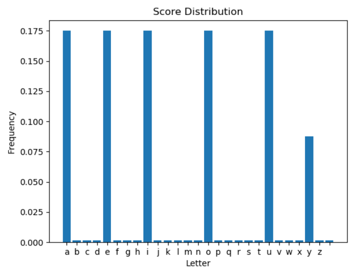
(a) Points



(b) Counts



(c) Alphabet



(d) Vowels

Figure 14: Distributions over letters that are proportional to the Scrabble Points, Scrabble Counts, Alphabet, and Vowels score functions.

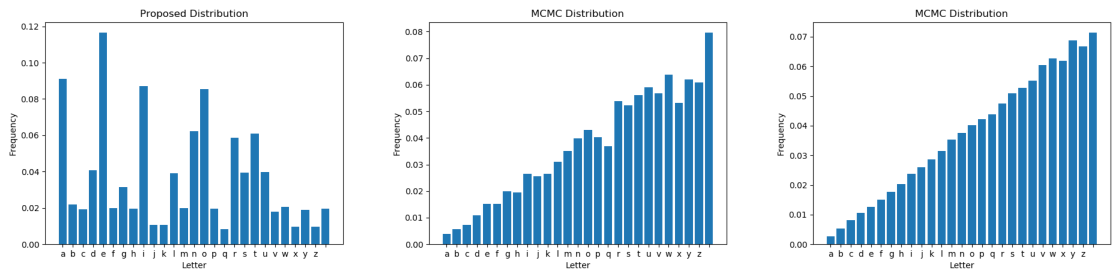
In order to perform the Metropolis–Hastings procedure we need a random proposal function Q which defines a distribution over the states given the current state and a score function s. We will usually take the proposal function to be a pre–defined Markov chain on the states or equivalently the transition probabilities at each state. That is, $Q_{x,y} = T_{x,y}$. At each step of the MCMC process we will use Q to generate a new proposed state and compare the score of the proposal to the current state. If the score of the proposal is higher, we proceed to the new state, otherwise we remain at the previous state with probability proportional to the ratio of the scores. It is this possibility of remaining in place that transforms the stationary distribution to our desired values.

More formally, at each step of the Metropolis–Hastings chain $X_1, X_2, \ldots$ we follow this sequence of steps, assuming that we are currently at state $X_k = y$.

1. Generating a proposed state $\hat{y}$ according to $Q_{y,\hat{y}}$.
2. Compute the acceptance probability:
$$\alpha = \min\left(1, \frac{s(\hat{y})}{s(y)} \frac{Q_{\hat{y},y}}{Q_{y,\hat{y}}}\right)$$
3. Pick a number β uniformly on $[0, 1]$
4. Set
$$X_{k+1} = \begin{cases} \hat{y} & \text{if } \beta < \alpha \\ y & \text{otherwise.} \end{cases}$$

This new Markov chain is ergodic and reversible with steady state distribution proportional to s. This means that we can use it to analyze properties of this new distribution, even though we might not be able to compute any of the probabilities directly. The software here allows you to vary the proposal distribution, score distribution, starting state, and length of the chain to gain some intuition for the MCMC transformation.

Figure 15 shows a first example of MCMC. Here the proposal distribution is drawing a tile from the Scrabble bag and we want to sample proportionally to the Alphabetical Score. Notice that although there are only 2 ‘ ’ tiles and 1 ‘z’ tile, the MCMC distribution sampled those states many more times than the ‘a’ tile, even though the ‘a’ tile was drawn much more frequently. This is an example of how the Metropolis–Hasting procedure of rejecting transitions and remaining in place changes the final distribution of samples. Notice that we will never be forced to remain at the ‘a’ tile, since its score is the worst of all of the states. As in our expected value experiments above, increasing the length of the chain increases the accuracy of the distribution.



(a) Proposed (b) 10k Accepted (c) 100k Accepted

Figure 15: The results of a length 10,000 MCMC run on letters. The proposals were generated according to the Scrabble tile distribution and the score function is alphabetical. In order to achieve this transformation, many transitions away from the high scoring tiles had to be rejected while we rarely remain at lower scoring tiles. Taking increasingly long chains (c) makes the output distribution closer to our desired distribution.

Having seen an example in action, we will step through some of the computations to see the numerical impacts of weighting, using the keyboard walk and the Scrabble score function. We begin by imagining that our current state is ‘a’. Following our algorithmic outline above:

1. We uniformly pick a key on the keyboard next to ‘a’: ‘q’.
2. We next need to compute some numbers in order to compute the acceptance probability:
   - $s(a) = 1$
   - $s(q) = 10$
   - $Q_{q,a} = \frac{1}{2}$ since ‘q’ has two neighbors
   - $Q_{a,q} = \frac{1}{3}$ since ‘a’ has three neighbors

   These let us compute:
$$\alpha = \min(1, \frac{10}{1}\frac{\frac{1}{2}}{\frac{1}{3}}) = \min(1, 15) = 1$$
3. Uniformly pick $\beta = .188256$
4. Set the next state to be ‘q’ since $\beta < \alpha$.

This makes sense with our interpretation, since ‘q’ has a higher score than ‘a’ we should always accept this transition. Now we will try to take another step from ‘q’;

1. We uniformly pick a neighbor of ‘q’ and get ‘w’.
2. We again need to compute some numbers:
   - $s(q) = 10$
   - $s(w) = 4$
   - $Q_{q,w} = \frac{1}{2}$ since ‘q’ has two neighbors
   - $Q_{w,q} = \frac{1}{3}$ since ‘w’ has four neighbors

   These let us compute:
$$\alpha = \min(1, \frac{4}{10}\frac{\frac{1}{4}}{\frac{1}{2}}) = \min(1, .2) = .2$$
3. Uniformly pick $\beta = .7593544$
4. Set the next state to be ‘q’ since $\beta > \alpha$.

This time, we proposed a state with a lower score and the move was rejected so our current MCMC walk so far has visited states ‘a’, ‘q’, and ‘q’. However, there was a 20% chance that we would have accepted the proposal and moved to ‘w’ in order to more fully explore the space. Figure 16 displays the score values sampled from a similar walk which displays the characteristic behavior of MCMC samples, exploring high values but also returning to low ones.

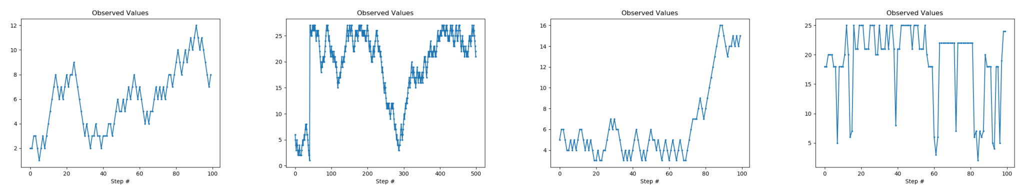
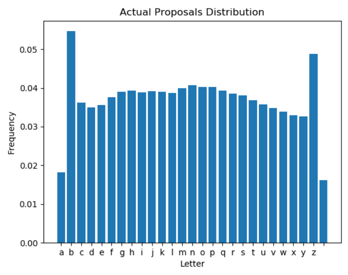
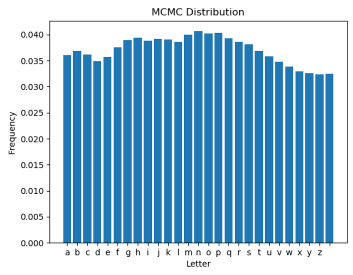


Figure 16: Traces of MCMC walks with the Alphabetical score function.

There are several natural cases in which the computation of the acceptance probability α can be simplified. The first is when we are trying to sample from a uniform distribution, in which case the score values are equal for all states and α is just equal to the ratio of the probabilities of transitioning between adjacent states. It is an instructive exercize to calculate the α values for all states in the path Markov chain when the score function is uniform. Figure 17 shows a helpful hint for this problem. The other simplification is when the two way transition probabilities are always equal, as is the case in the cycle walk, since each state has exactly two neighbors. In this case, the second term cancels and we are simply left with α as the ratio of the two score functions, which is usually simple to compute.



(a) Proposed



(b) Accepted

Figure 17: Converting the path Markov chain to uniform. Plot (a) shows the proposed steps and plot (b) shows the accepted samples after 100,000 transitions. How does MCMC change the weights on the original chain?

Recall that in this setting, our main goal is to draw samples from a predefined distribution. Thus, our metric of success is how closely the observed collection of samples matches the distribution proportional to s. One way to measure this is the *Total Variation* distance between the two measures. Given two distribution vectors P and Q, the total variation distance between them is

$$\sum_{i=1}^{n} |P_i - Q_i|.$$

When this value is zero the two distributions are exactly the same. To see that our MCMC method is converging our samples to the proper distribution, we can measure the total variation distance between our empirical draws with MCMC and the theoretical expected distribution that is exactly proportional to our score function. Figure 18 below shows some of these results. The green dots represent the distance between the MCMC distribution and the score distribution, while the blue dots represent the distance between the MCMC distribution and the steady state of the proposal distribution. As expected, the green dots go to zero as the number of steps increases, while the blue dots are bounded away from zero as the chain converges.

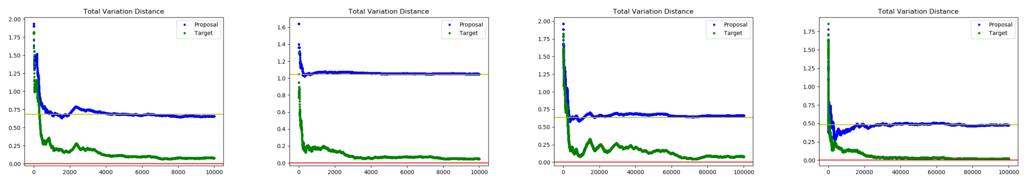


Figure 18: MCMC convergence in total variation. These plots show the total variation distance between the MCMC distribution and the score distriubtion going to zero as the number of steps increases.

6 Lifted Walks

The MCMC techniques discussed above show one way to take samples from Markov chain and transform them to reach a new stationary distribution. Another similar idea occurs in the setting of lifted Markov chains, where we will use an auxiliary graph to form a new chain on our original state space that converges more rapidly to the desired distribution. The key idea is that in well-behaved graphs (e.g. those with lots of symmetry) we can construct faster mixing chains by lifting to a larger graph where we already understand fast mixing walks. Although this is unintuitive at first glance, as we are moving to a setting with more nodes to try to move faster, this technique has been successfully applied in many combinatorial settings.

The purpose of this section is to explore **Example 1.1** presented in the paper Lifting Markov Chains to Speed up Mixing by Chen, Lovasz, and Pak. A GitHub repo with the source code for the corresponding Sage Widget and some colorful schematic diagrams is here. Before considering the specific example, we give a few words about the lifting procedure for walks on graphs. Given a graph, G our current interest is in drawing samples from the stationary distribution associated to a specified walk on the graph. However, the chain may be slow mixing (i.e. it takes many steps/samples to converge to stationary) and we desire a more rapid procedure for sampling. The plan is to find a different graph, H, along with a projection π from the nodes of the new graph to those of the old. We then form a new chain by lifting from G to H using π^{-1} and then taking a step on a faster mixing chain M on H before projecting back to G using π. This procedure is summarized in the following diagram:

$$\begin{tikzcd}
H \arrow[r, "M"] & H \arrow[d, "\pi"] \\
G \arrow[u, "\pi^{-1}"] \arrow[r, dotted, "\hat{M}"] & G
\end{tikzcd}$$

This procedure defines a Markov chain $\hat{(}M) = \pi M \pi^{-1}$ on G and if we choose H, π, and M appropriately we can guarantee both that the stationary distribution of $\hat{M}$ is equal to that of our desired walk on G and favorably bound the mixing time.

For the specific example considered here G will be the path graph on n nodes P_n and H is the cycle graph on $2n-2$ nodes C_{2n-2}. The simple random walk on the path has steady state proportional to the vector $(1, 2, 2, \ldots, 2, 1)$ while the stationary distribution on the cycle is uniform as long as each node has the same probability of stepping to the right[6]. We can choose any projection $\pi : C_{2n-2} \to P_n$ that maps exactly two nodes of the cycle to the interior nodes of the path and exactly one node each to the endpoints of P_n. Notice that given such a π a sample drawn from the uniform distribution projects to a sample from the stationary distribution on P_n. Figure 19 shows two possible cycle projections where the nodes from the cycle map to the correspondingly colored nodes in the path. Note that the purple and red colors only occur once on each cycle, while each other colors appear twice so that the stationary distribution is hit by the projection.

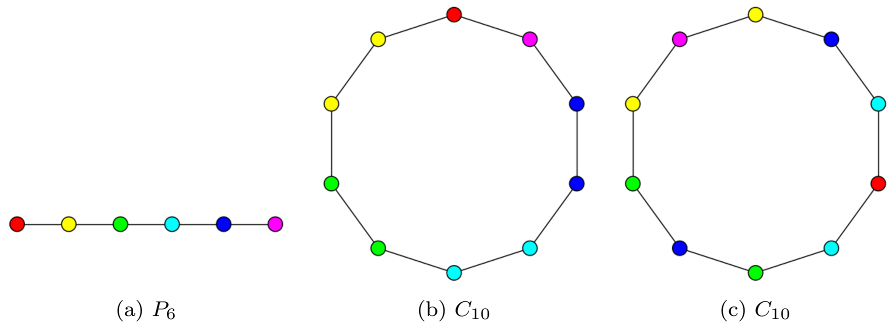

(a) P_6 (b) C_{10} (c) C_{10}

Figure 19: Permissible projections of the 10 cycle to the 6 path. The nodes on the cycles project to the correspondingly colored node on the graph.

[6]and hence to the left as well

More formally, we form the lifted Markov chain by repeating the following sequence of steps, assuming that we are currently at node i in the path at step n, so $X_n = i$.

1. Select a node p in the cycle uniformly from $\pi^{-1}(i)$
2. Draw a uniform number z on $[0, 1]$
3. Move to a neighbor q of p:

$$q = \begin{cases} p+1 \pmod{2n-2} & \text{if } z < \sigma \\ p-1 \pmod{2n-1} & \text{otherwise} \end{cases}$$

4. Project back to the path $X_{n+1} = \pi(q)$
5. Repeat with $X_{n+1} = \pi(q)$

The schematic in Figure 20 shows an example step of this procedure with $G = P_6$ and $H = C_{10}$ using the projection defined in Figure 19 (c).

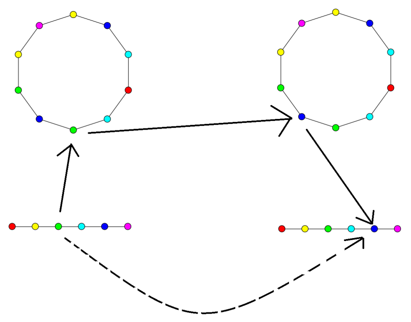
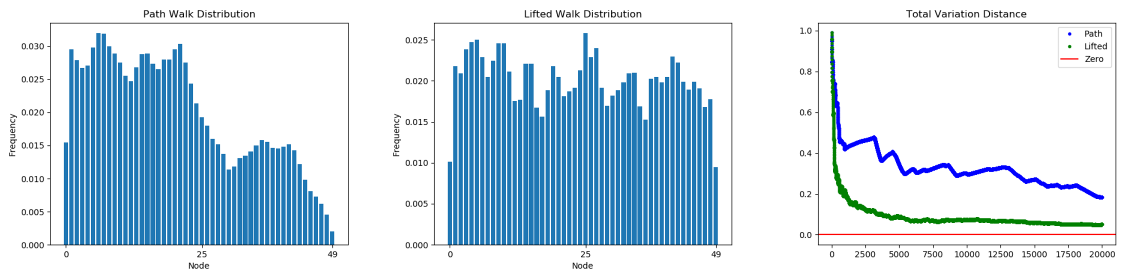


Figure 20: A sample step in the lifted Markov chain. We are at the green node of the path, so we begin by selecting one of the green nodes in the cycle at random. To complete the step, we move to an adjacent node on the cycle and then project back to the path.

The claim that we actually want to evaluate is that this new walk is faster, even though the cycle graph has approximately twice as many nodes as the path. To empirically demonstrate this, we can run both chains in parallel and compare their convergence times to the stationary distribution using the TV distance introduced above. Figure 21 shows an example of this, comparing the simple random walk on P_{50} to the lifted walk through C_{98}. As expected, the lifted walk converges to stationary much more rapidly than the original walk. You can try varying the parameters yourself here.



(a) Path Walk (b) Lifted Walk (c) Mixing Time

Figure 21: Improved mixing time on the path through lifting.

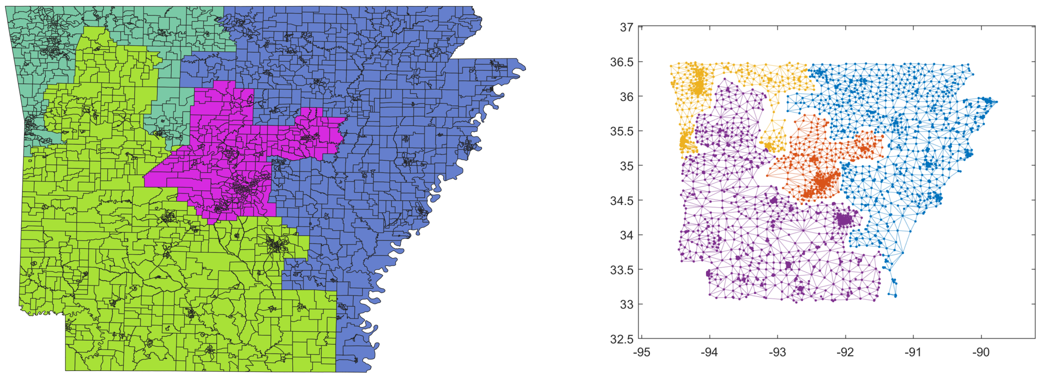
7 MCMC for Redistricting

We now turn our attention to the application of MCMC methods for political redistricting. This methodology has been applied in several significant court cases over the last decade to evaluate the outlier status of individual enacted or proposed plans. More recently, these techniques have been used to evaluate baselines for partisan metrics across different states and to predict potential impacts of reform legislation. For all of these tasks, what is needed is a large *ensemble* of plans, sampled from the set of feasible districtings that satisfy the relevant state and federal rules. Since the purpose of this piece is to describing the mathematics of these Markov chains we won't look too deeply at the specific applications here. However, many examples of this type of analysis can be seen in our work here.

The remainder of this section describes the mathematics that underlies these sampling methods. First, we write a describe a formal version of the problem in the language of *graph theory* in Section 7.1 and explain how to "operationalize" state laws as mathematical functions to define the set of permissible plans in Section 7.3. Much of the recent research in this area has been focused on the proper choice of proposal distribution and we address that problem in Section 7.4. Finally, we describe the types of score functions used to define our intended sampling distributions and techniques to reduce the mixing time in Sections 7.5 and 7.6 respectively.

7.1 Dual Graph Partitioning

Although most people think of gerrymandering in terms of lines or polygons drawn in the plane, political redistricting is really a discrete problem where individual units, like census blocks or voting precincts are divided into groups. Our main object of study will be the *dual graph* of a state, which represents each individual unit with a node and places an edge between two nodes if they share a common border. Figure 22 shows this process on the Arkansas precincts, comparing the underlying geography to the corresponding dual graph.



(a) Geography (b) Dual Graph

Figure 22: The state of Arkansas divided into precincts and the corresponding dual graph.

As discussed above, the first step towards defining a Markov chain is to identify the state space. For our study of political redistricting the basic set of states will be *districting plans* that assign each unit to a particular district. Figure 23 displays some example partitionings generated by computer compared to the currently enacted plan. Notice that there are many "poorly behaved" districting plans, that can satisfy many of the constraints as written in to redistricting law. One of our main goals with MCMC is to move around the space of plans without encountering these.

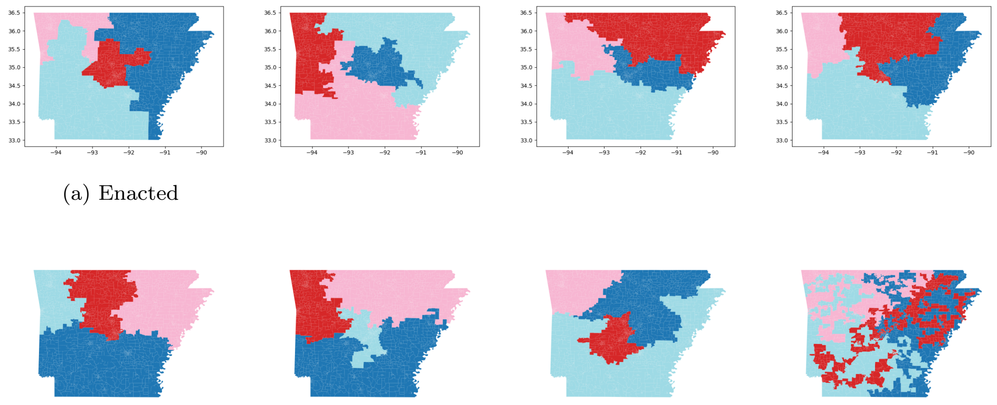

(a) Enacted

Figure 23: A collection of partitionings of the Arkansas geography. Plot (a) shows the currently enacted plan, while the remainig plots were generated by various computer methods.

7.2 Data Gathering

Although this document is mostly focused on the properties of Markov chains, it is worth pausing to discuss the difficulties inherent in actually assembling the data to perform this analysis. One of the main issues is that the placement of local precinct boundaries is usually left up to the municipalities or counties and there is no requirement that the local units report changes in the boundary lines to the state. This leads to the common situation that actually no one knows where the precinct lines are located because a the boundaries are only described on paper maps, tacked up to the wall in a county office building.

More detailed information about the data gathering and processing procedures necessary to prepare for a chain run is presented here. That piece also describes some of the data challenges, such as disconnected units, water boundaries, missing data, doughnut precincts, etc. that you might encounter while assembling an accurate shapefile. Throughout the remainder of this piece we will assume that the data has been cleaned properly and that the associated dual graph is well–defined.

7.3 Defining Permissible Partitions

As we saw in Figure 23 there are many possible graph partitions that do not correspond to good districting plans. In order to address this issue, we will put some constraints on our state space, so as to avoid those types of degenerate partitions. We begin by analyzing the commonly enforced "Traditional Districting Principles." The most natural one to enforce is the notion of contiguity, that the individual districts be connected as subgraphs of the full state graph. Figure 24 shows the subgraphs corresponding to the current AR districting plan. Within each district, each node can be reached from each other node, so the subgraphs are connected and the districts are contiguous.

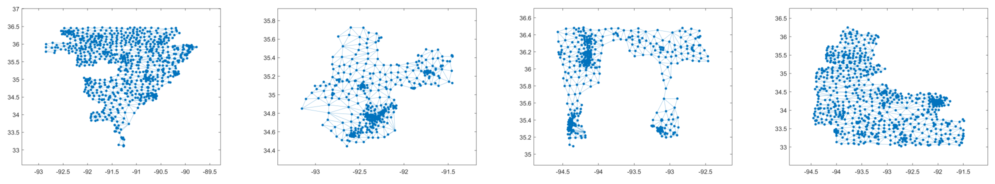


Figure 24: Subgraphs corresponding to the current AR districts. Notice that each forms a connected graph when considered on its own.

The next metric we consider is population balance, usually expressed by the slogan "One person, one vote." The national standard for congressional districts requires very well balanced districts – most states try to zero–balance, or to make sure that the difference in populations between any pair of districts is at most one person. For state legislative districts, imbalances of up to 10% between the largest and smallest district are permitted without triggering a malaportionment claim.

This definition of imbalance highlights one of the complicated issues that arises when trying to operationalize these laws. There are many ways to measure the overall population deviation of a given plan. We might compute some type of average of the deviations from ideal of each district or simply restrict the maximum deviation of any district from ideal to be below some percentage. Each of these formulations leads to a different state space and potentially a different distribution of permissible plans. Our recent work in Virginia suggests that depending on the particular Markov chain, the amount of population deviation permitted may not impact the overall distributions significantly.

In addition to these measures, we must also operationalize various compactness scores, locality preservation measures, compliance with the Voting Rights Act of 1965, and other rules enacted by state legislatures. For each of these, as in the case of population balance, we must make a choice of metric, either to evaluate on the individual district or the plan as a whole to rule out non–conforming plans. Expert reports in court cases show the wide variety of measures that have been implemented by various groups performing MCMC analyses.

This approach to specifying the state space in terms of function scores has a computational advantage as well. For most of our examples, it is much easier to evaluate whether or not a proposed partition satisfies each of a collection of constraints than to compute or generate all of the districting plans that are adjacent to the given state under an arbitrary proposal. For example, using the ReCombination proposal described below, it is possible to have an unimaginably large number of neighboring states, far more than could ever be stored or evaluated before the heat death of the universe. However, it is easy to generate samples from of these neighboring partitions and relatively quick to *rejection sample* or discard proposed neighbors that don't satisfy the constraints defining the state space. As long as we pretend that these proposals never occurred, we maintain the Markov property for our walk.

This means that our algorithm for generating samples from our Markov chain is a little different than that described above. Given our current state, we generate a sample according to our proposal distribution and evaluate it according to the constraints on the state space. If it is a permissible state, we accept it into the chain and repeat the process. However, if the proposed state is not permissible because it fails one of the constraints we generate a new state from the proposal distribution and try again. Note that we only move to a new state when it is in the state space, so the transition probabilities are exactly what they would be if we had access to the list of adjacent states, even though we cannot compute them directly.

7.4 Proposal Distributions

In the original setting of Markov chains on the letters of the alphabet, we specified the chain by defining the transition probabilities between each pair of letters. Unfortunately, there are far too many partitions of a state sized graph for us to attempt to compute or store all of the necessary probabilities. This leads us to specify our Markov chain by describing a set of local moves, or rules, that we can apply to a given state in order to generate proposed states that are "close" to our current state in some sense. We will allow these rules to have a random component so at each partition we get a distribution over nearby partitions for each step but we will not actually compute the transition probabilities directly.

As an example, consider the rule where at each step we choose a node in our dual graph uniformly at random and change its assignment to a uniformly random district. This certainly gives us a distribution over partitions but for an average districting plan is unlikely to generate a contiguous partition since most nodes lie on the interior of their districts. Although this is a poor proposal in practice, it is still instructive to think about how much additional computation it would be necessary to compute the transition probabilities for a specific partition. We would have to evaluate each possible adjacent districting. For the Congressional districts of Pennsylvania, built out of precincts this gives us about $9000^{17} \sim 16.7 \times 10^{67}$ neighboring partitions to evaluate. On the other hand, it is a very simply process to carry out computationally, since selecting and flipping a single node is a relatively efficient process.

This tension is similar to that discussed above for score functions, in that it would be remarkably inefficient to actually evaluate each partition, instead of just sampling a subset until we find one that works. This type of procedure is also a type of rejection sampling and can be quite efficient as long as the rejection rate, or number of attempts we have to discard before finding one that satisfies our constraints is not too high. The simple example considered in the previous paragraph is too inefficient (grids/PA/VA?) but the next set of proposals we will consider is actually too

Much of the original work on MCMC for redistricting focused on versions of the *boundary flip* proposal, where at each step a single node changes assignment to match the district of one of its neighbors. This type of proposal has some computational advantages, in that it is easy to iteratively update the computations of score functions and keep track of the set of nodes that can potentially be changed at each step while preserving contiguity.

Even this simple sounding proposal can be implemented in many subtly-different ways, each leading to different steady state distribution, similar to the problem we had above in trying to define how to measure the idea of population imbalance. One option would be to select an edge at every step and randomly change the assignment of one of its ends (also at random) to match the other. Another version might instead keep track of all of the (node, district) pairs that could be changed to remain contiguous and sample uniformly from that set. It is an instructive exercise to show that these proposals do not have the same steady state distribution.

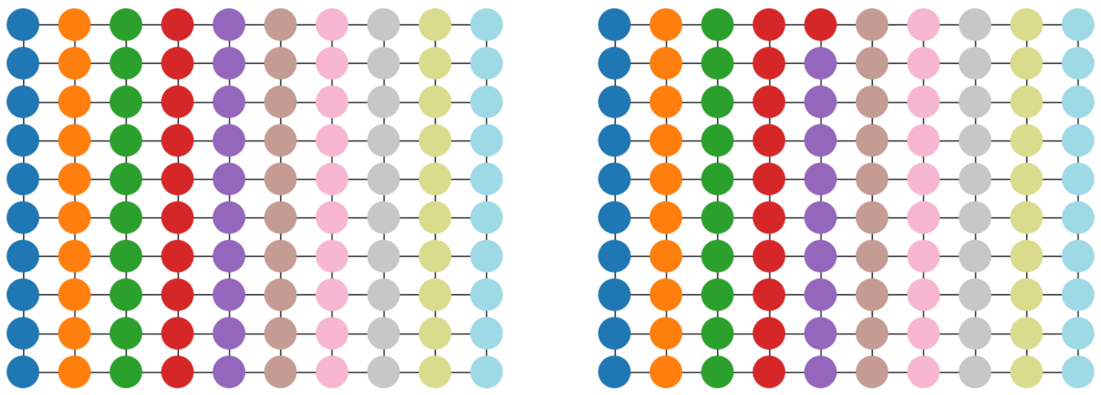


Figure 25: Schematic of the boundary flip proposal.

A more recent proposal, introduced by our group here makes larger adjustments to the partitioning at once. At each step of the chain, we begin by merging the units of two adjacent districts in the plan and then bipartitioning the induced subgraph using a spanning tree method.

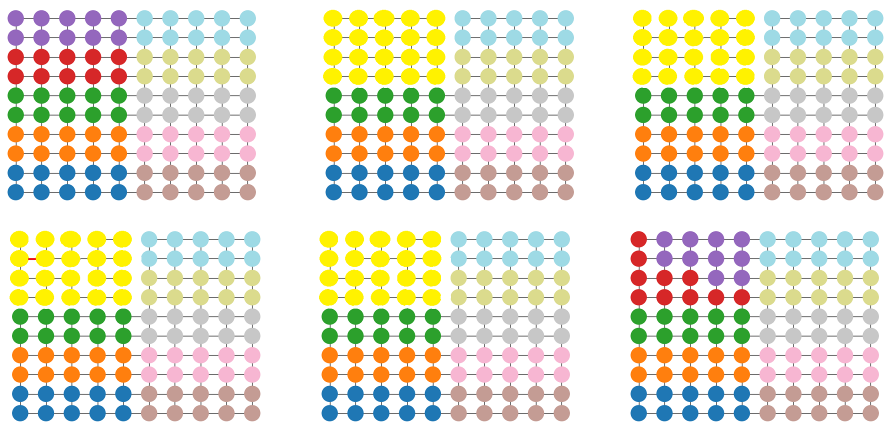


Figure 26: Schematic for the Tree ReCombination proposal.

The document here contains several more examples of bipartitioning functions one could use to generate new proposal distributions with this "ReCombination" approach. Analyzing the properties of these distributions is an active area of research as is generating new proposal distributions that mix rapidly on the space of graph partitions. Some tools for experimenting with partitions on grids like these can be found here.

Another useful exercise is determining weights for MCMC to transform any of these proposal distributions into the uniform distribution similar to the method referenced in Figure 17. As the proposal methods in this space are more complex than those considered above, being able to use Metropolis–Hastings to transform the distribution of the underlying Markov chain to one of our choosing becomes increasingly important.

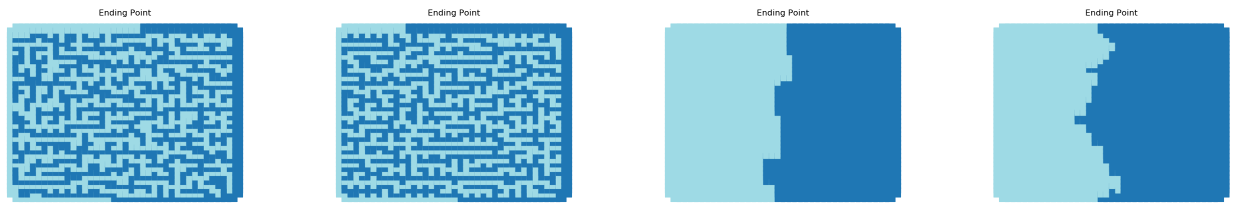
7.5 Acceptance Functions

An seemingly promising initial approach is to try to sample uniformly from the class of permissible districting plans. However, this is a computationally difficult task, as our recent work shows that no general choice of proposal distribution can converge to the uniform distribution across a broad class of graphs efficiently. Thus, if we want to sample effectively, we must choose another type of distribution. As in the original MCMC examples with Scrabble, a natural way to approach this problem is by constructing a score function that prioritizes the kinds of properties, like population balance, compactness, or locality preservation, that we would like our districting plans to satisfy.

Motivated by approaches used in statistical physics we can construct an "energy" function $s(P) = e^{-\beta \sum \alpha_i \sigma_i(P)}$ where P is a partition of our graph, β is a constant called the inverse temperature and the α_i are linear coefficients on the σ_i which are the individual score functions representing our metrics of interest. We then want to sample partitions proportional to their score $\mathbb{P}(P) = \dfrac{f(P)}{\sum f(Q)}$. The denominator here is a sum over all partitions of the graph that satisfy our binary constraints. Since we are unable to compute this set on real world examples this is a perfect setting for applying the MCMC method.

As an example of an energy function we will consider partitioning a grid graph into two parts prioritizing partitions that have the same number of nodes in each part. Thus, give a partition $P = (A, B)$ we set $\sigma_{pop}((A, B)) = ||A| - |B||$ to be the difference in the sizes of the partitions and our score function is $e^{-\sigma_{pop}(A,B)}$. Sampling proportional to this distribution means that a plan with exactly balanced populations should appear approximately $e^{10} \sim 22,026$ as frequently as a plan with 10 more nodes in A than B. Figure 27 (a-b) show generic examples of such a population balanced plan, generated with the boundary flip proposal.

As the previous plan would be unsatisfying as a legislative district, we decide to add a second term to our energy function to control the length of the boundary. Given a partition $P = (A, B)$ we set $\sigma_{cut}((A, B)) = |\{(u, v) \in E(G) : u \in A \ v \in B\}|$ and change our energy function to $e^{-\sigma_{pop}(A,B)+\sigma_{cut}(A,B)}$. Figure 27 (c-d) show examples drawn from this distribution that prioritizes both compactness and population. Although this looks like a nice partition, we notice that this energy function makes an implicit choice of relative importance of population balance against compactness - a single node imbalance is equal to a single extra unit of boundary length. Whether or not this is a reasonable choice is a complex question on grids to say nothing of real world examples. This example highlights some of the difficulties in making principled decisions about this type of sampling and studying the impacts and tradeoffs inherent in these procedures is an active area of research.



(a) (b) (c) (d)

Figure 27: Partitions of a grid into two parts. Plot (a) shows a generic population balanced plan while plot (b) shows a plan that is both population balanced and has low perimeter.

Having built the Markov chain itself, the next step is to construct Metropolis–Hastings acceptance functions for this setting. From a theoretical perspective, this functions exactly as in Section 5 computing the score of the current and proposed state and the transition probabilities between states. From a computational perspective, if we choose our σ_i properly evaluating the score function should be efficient. However, computing the exact transition probabilities can be complex even for a specified pair of states.

To see why this is, consider the simple boundary flip proposal where we choose an edge between two districts and change the assignment of one of its endpoints. At first, it seems easy to compute the probability of moving between adjacent partitionings since we can compute the number of edges that lie between districts and we choose uniformly over the edges and endpoints. This is insufficient for two reasons. First, recall that our state space is actually formed out of partitions of the graph and there may be multiple edges that connect a node to the same partition. Thus, the actual states are not uniformly represented by the edges. Secondly, not all of the possible choice of edges can be flipped while remaining in our state space. Some choice of nodes to change might disconnect the graph or cause the populations to be too unbalanced. Solving this problem can require significant extra care in the choice of proposal method or application of a different type of preferencing procedure to move through the state space.

7.6 Mixing Times and Temperature

Having selected a distribution that we would like to converge to, we are left with the problem of how to make sure that we actually get there. The graphs that we are studying frequently have hundreds of thousands of nodes, and hence requiring even a polynomially large number of steps may be infeasible. However, we would also like to be consistently generating plans that satisfy our binary constraints and not waste too many steps on low acceptance probabilities.

This is a particular problem for the boundary flip for three reasons. First, since it only changes the assignment of a single node at each step, it takes many steps to move around the space. Secondly, the flip procedure produces a distribution that is close to uniform. A consequence of this is that maps sampled from this procedure tend to look like the last plot in Figure 23 or even worse, since a generic connected partition can be shown to be space-filling in a certain theoretical sense. Figure 28 (a) also shows an example of this, plotting the length of the boundary against the number of steps in an unconstrained run. Finally, adding strong constraints around compactness or population balance disconnects the state space – since the walk is very local it is easy to construct examples of partitions that cannot be reached from each other using this procedure.

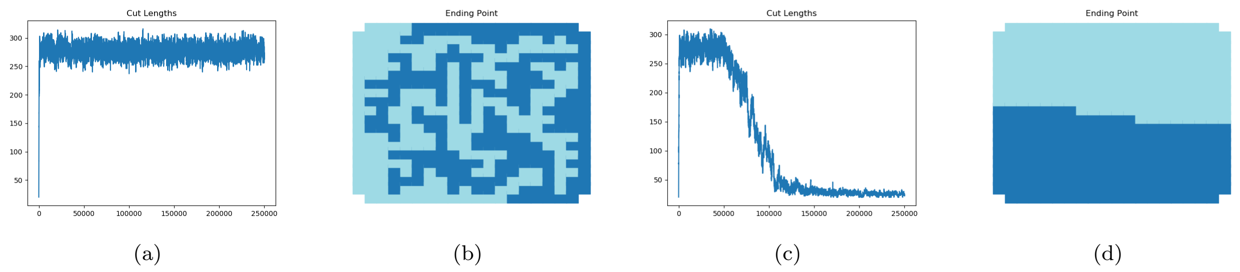
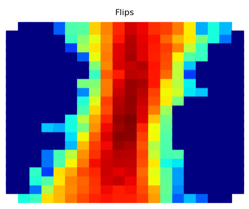
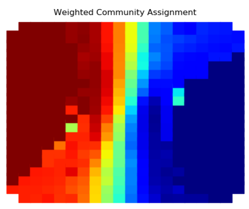


Figure 28: Length of the boundary of a grid partition using the boundary flip walk. Figure (a) shows an unconstrained walk while (b) shows the final state. Plots (c) and (d) show an example with annealing.

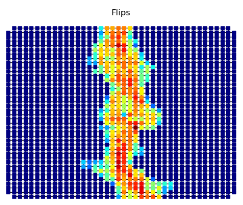
In particular, the second and third problems are in direct tension with each other, since in order to move throughout the space we need to permit flexibility of compactness and population balance but the chain itself will tend towards plans that are fractal, given any opportunity. Thus, if we weaken the constraints, we are likely to generate many partitions that from the less compact part of the distribution. This tension exacerbates the problem of slow mixing because trying to force the chain towards more compact partitions using MCMC means that we will reject many proposals. Figure 29 shows an example of this behavior, plotting the number of times each node is flipped in walks that constrains the boundary length and population balance individually.



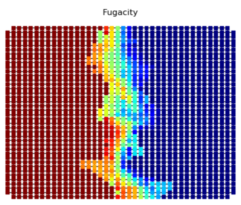
(a)



(b)



(c)

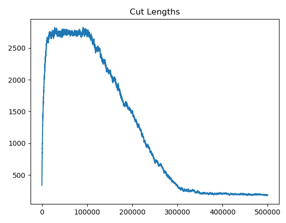


(d)

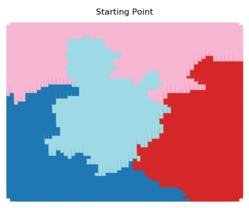
Figure 29: Constrained examples of the boundary flip walk. Plots (a-b) show the number of flips and average community assignment for a population constrained Markov chain, while (c-d) show similar experiments with the boundary length constrained. In both cases, the chain struggles to move away from the initial horizontal partition.

One technique from the physics literature that applies well to this setting is the idea of *simulated annealing*. The key idea is to use the inverse temperature parameter β in our energy function to control the importance of the properties that we are measuring. Notice that when β is very small, the score function is approximately 1 for all plans, so our sampling procedure is approximately unconstrained or uniform. Correspondingly when β is very large, the relative differences between plans get magnified, so almost every accepted state will have a higher score than the previous one. Sometimes this distinction in behavior is referred to as explore vs. exploit since low values allow us to move around the state space easily but large values force us towards the extremes of our distribution. Figure 28 (c-d) shows an example of this type of behavior where the initial steps have a high temperature and approximate the uniform distribution but as the temperature is lowered the boundary contracts to a more reasonable shape.

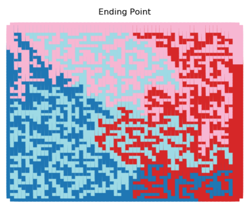
Figure 30 shows some more examples of this behavior. Notice that the final states are quite compact but that they are not substantially different than the initial states. A deeper look shows that while the centers of the distributions get quite complicated, the boundaries tend to remain fixed. Thus, it is still difficult to actually move to a new portion of the partition space, even with annealing.



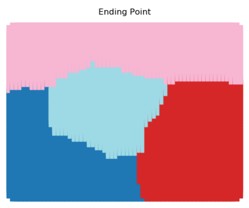
(a)



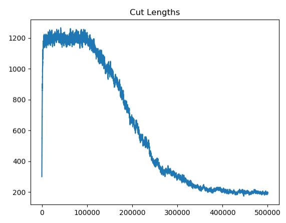
(b)



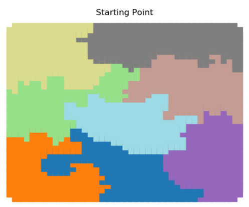
(c)



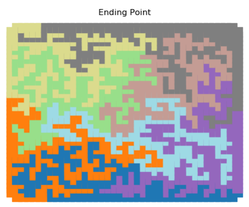
(d)



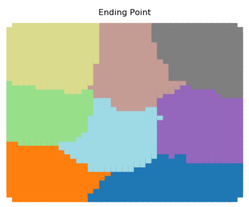
(e)



(f)



(g)



(h)

Figure 30: Annealing examples with multiple districts in the partition. The left column shows the length of the boundary of each partition in the Markov chain. The other columns show the initial partition, the state before cooling starts, and the final state after the annealing process.