

Using MCMC to Measure Voter Unfairness in the Process of Redistricting

Natalie Tripp Foulds

May 9, 2021

Abstract

Markov Chain Monte Carlo (or MCMC) has become a commonly used statistical method over the past 50 years in a wide variety of areas, one of these being the process of redrawing voting district boundaries before elections. This process is known as redistricting and involves partitioning an area into a number of districts that accurately represent the populations vote whilst also abiding by the state and federal laws. Despite sounding like a relatively simple task redistricting can often lead to misrepresentation of the political demographic. This is more commonly known as gerrymandering. This paper goes into detail on the process of MCMC, the mathematics behind it and how it can be used to measure the unfairness of a political district map.

Contents

1	Introduction	3
2	Gerrymandering and how it occurs	3
3	Measuring unfairness	4
3.1	Partisan Symmetry	4
3.2	The Efficiency Gap	5
3.3	Sampling using MCMC methods	6
4	Sampling in practice	6
5	Markov chains	8
5.1	Periodicity	11
5.2	Reducibility	11
5.3	Recurrence and Transience	12
5.4	Ergodicity	13
5.5	Stationary Distributions	14
5.6	Reversibility	15
6	The Monte Carlo Method	15
7	Markov Chain Monte Carlo	18
7.1	The Metropolis-Hastings algorithm	18
8	Redistricting	20
8.1	Area representation	20
8.2	Implementing the rules	22
8.3	MCMC and random walks	23
8.4	Mixing times	23
8.5	Flip and ReCom	24
8.6	Outlier Analysis	25
9	Conclusion	26

1 Introduction

During the 2012 United States House of Representatives election an unusual event occurred [1] where despite the Democratic Party receiving the majority of votes nationwide, the Republican Party still won the majority of legislative seats. When described in these simple terms you would not expect such an occurrence to be possible however when looking at the specific district border lines such an anomaly can be understood.

There have also been incidents in the UK where the specific district border lines have been a matter in question. In 2010 the Labour Party in particular were criticised for their mapping proposals as they were thought to favour their party [2]. In fact, there have been several disputes between the Labour and Conservative parties over how the electoral boundary lines should be drawn [3]. This particularly occurs when there appears to be no clear winner, as this slight change in electoral boundaries can then be enough to push a certain party to success.

Both these occasions happened due to the partitions of voting districts that were made. When the political district lines are deliberately drawn in such a way as to favour a certain party it is known as gerrymandering. In general it appears to be a far more common phenomenon and to occur to a greater extent or severity in the United States. This is mainly because for the majority (43) of US states it is a legal requirement (by the Reapportionment Act of 1929) to redistrict every 10 years as a minimum to account for population shifts [4] and in some of these states the party in power has control over the redistricting process [5].

To reduce the risk of gerrymandering states will usually employ independent bodies to oversee the process [6] and increasingly mathematical solutions have been sought that would provide an objective measure of the fairness of any district border adjustments. One of the more recent techniques used by mathematicians to address this challenge is MCMC.

2 Gerrymandering and how it occurs

Now, to explain in detail how gerrymandering happens we need to look at the mathematics behind it. Kristopher Tapp in his article "Measuring Political Gerrymandering" [7] shows a very simple yet effective example of this. If you take an area containing 50 voters, which is partitioned into 5 districts (10 voters in each), and 30 vote red whilst the other 20 vote blue. There are several different ways of dividing the area that will result in different outcomes for each district, and hence a different overall outcome see figure 1. The district vote is decided based upon the majority of results for that district. As we can see in the figure plan 1 and 2 both favour red overall however in plan 3, despite red still winning

the total amount votes, blue would win the election as there are three districts with majority blue and only two with majority red.

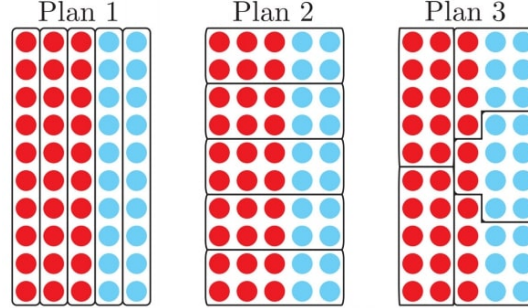


Figure 1: Different ways of dividing sample area [7]

This simple principle can be applied to countries like the U.S. with states and districts, however the picture becomes much more complex with far more advanced maths behind it as the population size increases. The highly populated areas become more varied and the pattern of votes over the area gets more intricate.

3 Measuring unfairness

We can now see how this occurs, but how can you tell if it is occurring and to what extent? As we can see from the previous example it appears very easy to sway the votes using simple techniques hence there are rules and concepts in place to spot this prevent it from happening.

3.1 Partisan Symmetry

One of these techniques [8] is known as *Partisan Symmetry* which attempts to plot the way the election map affects the votes and use symmetry to see if it is considered fair. It is measured by evaluating V and S . V (for votes) describes the proportion of overall votes won by a party and S (for seats) refers to the proportion of electoral seats won by that party. V and S are plotted on a graph and the symmetry around the central point is assessed and used to show how many of the seats a party can secure with the amount of votes they have on a given map.

We can see an example of this in figure 2 displaying the partisan symmetry graphs for the Republican Party in two states. We can see that there is a symmetry present in the graph for Minnesota however in Ohio it appears the Republicans are able to gain a lot of seats for a smaller proportion of the overall votes.

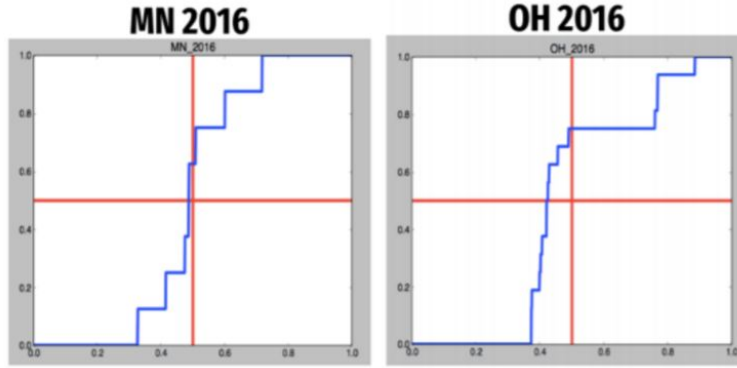


Figure 2: V (x-axis) and S (y-axis) in Minnesota and Ohio 2016 [8]

3.2 The Efficiency Gap

Another common concept [8] used to check the fairness of particular boundaries is known as the *efficiency gap*. The efficiency gap looks at the amount of votes wasted for each party and suggests that a plan is fair if that number of votes wasted is equal for both opponents. An ideal border plan, where both party wastes the same amount of votes, will have $EG = 0$. An EG threshold is proposed, for example $|EG| = 0.08$, and if the EG is higher, the border plan is classed as being unfair (see figure 3).

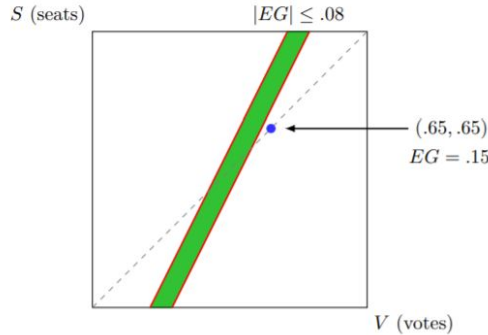


Figure 3: The green line shows the area in which the outcome must be to be under the EG threshold of 0.08 [8]. Similarly to the partisan symmetry a graph of V (x-axis) and S (y-axis) is taken and an area under the EG threshold is shown. As we can see point $(0.65, 0.65)$ is not within that particular area (as $0.15 > 0.08$) and hence this would be considered to be an example of gerrymandering.

3.3 Sampling using MCMC methods

Sampling is a recent approach to the issue of redistricting which measures gerrymandering by collecting all of the possible ways a specific area can be partitioned to see how accurately each map represents the overall demographic vote of that area. The maps are collected by using Markov chains (see section 5) and are plotted on an *outlier analysis graph*, an example of this is shown in figure 4 below. On these plots, the x -axis depicts the percentage of the population vote and the y -axis the amount of maps. By visualizing this data on a plot we are able to see which maps accurately represent the voting data. Any outliers on these maps are thus likely to be a case of gerrymandering.

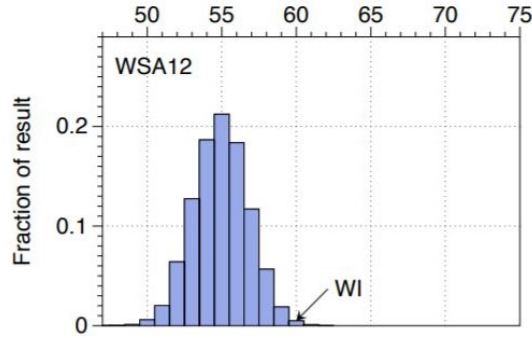


Figure 4: This plot shows the outlier analysis from the 2012 presidential elections in the U.S. state of Wisconsin [13]. The x -axis shows the republican seats (99 seats in total). In 2012 the majority of maps showed that approximately 55 seats would be won by the Republicans, however the map which was used (as shown by the arrow) resulted in 60 Republican seats, and was further from the average result of 55 seats than 99.5% of the other maps [8]. This redistricting plan could hence be considered to be a gerrymander.

Despite being used on several occasions in court MCMC sampling has not yet become a legal obligation for creating fair redistricting plans [10].

4 Sampling in practice

Unfortunately this technique is not straightforward to implement, the main problem being the sheer amount of maps that can be produced. If we picture a simple 2×2 grid and we partition it into 2 equally populated pieces, it is very easy to compute that we will only have 2 different maps, the first with a vertical partition down the centre and the second with a horizontal partition. If we increase this to a 4×4 grid with 2 equally populated districts, we get 13 different ways

of splitting the grid without considering their symmetries. If we take a look at figure 5 below the green grid only has 2 rotations without repeating the same grid and hence, we multiply this grid by 2. The blue grids have 4 rotations (and are multiplied by 4), the orange grids have 2 rotations and a reflection (hence each are multiplied by $2 * 2 = 4$) and the pink grids have a reflection and 4 rotations (so are multiplied by 8). Once the possible rotations and reflections have been taken into account we calculate the total number of possibilities as follows,

$$1 * 2 + 2 * 4 + 5 * 2 * 2 + 5 * 4 * 2 = 70.$$

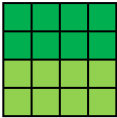
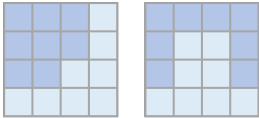
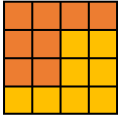


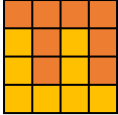
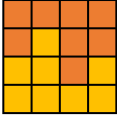
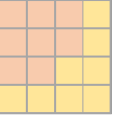
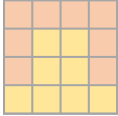
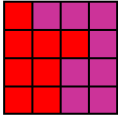
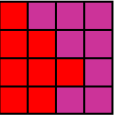
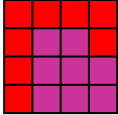
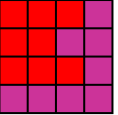
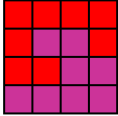
	2 Rotations	4 Rotations
No reflection		
Reflection	      	    

Figure 5: This table shows each of the possible ways of splitting a 4×4 grid (without including symmetrical variants). The grids that are shown in a faded colour are shown in two sections (but should only be counted once) as each has a reflection that is equivalent to a number of rotations.

From this we can see how the number of possibilities rapidly increases as the area and population increases. When looking at real (and oddly shaped) U.S. states we are dealing with populations between 500,000 and 40 million, these can not always be split into completely equal districts and will have more partitions than just two which suddenly increases the number of possibilities to an incalculable number that would take longer than our lifetime for any modern computer to compute [10].

Due to this issue a special technique using Markov chains is used. Markov chains use spanning trees to work out how many ways of partitioning a state there are rather than working out each possible map. There are different approaches and algorithms using Markov chains being used and tested to find the most efficient use of the technique for the problem [9] [11]. A paper particularly relevant to this topic is Moon Duchin, Daryl DeFord and Justin Solomon’s article ”Recombination : A family of Markov chains for redistricting” (2020) [12], in which a recent technique of MCMC using the ReCom markov chain is proposed to be a more efficient and successful tool in the process of redistricting compared to the more traditional and commonly used *Flip* algorithm. These two techniques will be discussed in further detail in section 8.5.

Another factor that limits the incredibly high outcome of possibilities is the current federal laws in places that each map proposal must follow. The main rules include population balance, all districts should have a similar population size, contiguity, each district must be entirely connected, and compactness, each district should have a compact and regular shape i.e. there should not be too many edges bordering two separate districts [12]. These constraints are what are used in the process of redistricting as constraints that each map must satisfy. This allows us to limit the number of possibilities to a valid and broad range of district maps.

5 Markov chains

Markov chains are a sequence of random variables that revolve around the principle that each variable is dependant only on the variable before it.

Definition 5.1. *A stochastic process is a collection of random variables, $X = \{X_n : n \in N\}$ taking values from a finite set S . Further, a stochastic matrix (also known as a probability matrix), P , is a matrix with the following two properties*

- $\sum_{j \in S} P_{i,j}$ for all $i \in S$ (Stochasticity)
- $\forall i, j \in S$ where $P_{i,j} \geq 0$ (Non-negativity).

We define Markov chains mathematically as follows [16]:

Definition 5.2. *Let $X = \{X_n : n \geq 0\}$ be a stochastic process on a finite set S . X is a Markov chain if*

- $P(X_{n+1} = i_{n+1} | X_0 = i_0, X_1 = i_1, \dots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n)$
- $P(X_{n+1} = j | X_n = i) = p_{i,j}$

for any $i, j \in S$, where $n \geq 0$.

Note. S is commonly referred to as the state space. $p_{i,j}$ is the probability that the chain transitions from state i to state j in a single step and $P_{i,j}$ is the transition matrix containing these probabilities.

This is a transition matrix, P , with state space, $S = \{1, 2, \dots, S\}$, where $i, j \in S$ and $2 < i, j < s$ [15].

$$P_{i,j} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} & \cdots & p_{1,s} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j} & \cdots & p_{2,s} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots & p_{i,s} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{s,1} & p_{s,2} & \cdots & p_{s,j} & \cdots & p_{s,s} \end{pmatrix}$$

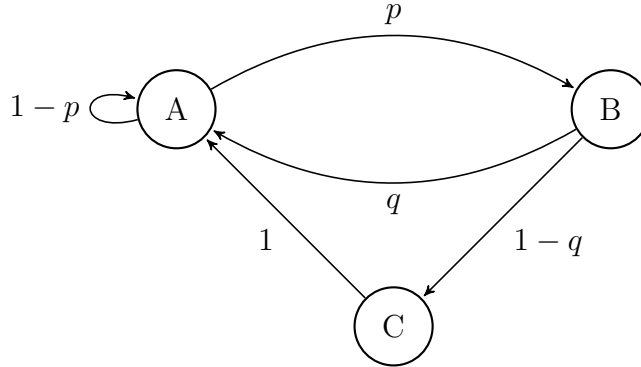
Theorem 5.1. Let $X = \{X_n : n \geq 0\}$ be a stochastic process with transition matrix, P .

$$P(X_0 = i_0, X_1 = i_1, \dots, X_n = i_n) = p_{i_0 i_1} \cdots p_{i_{n-1} i_n}$$

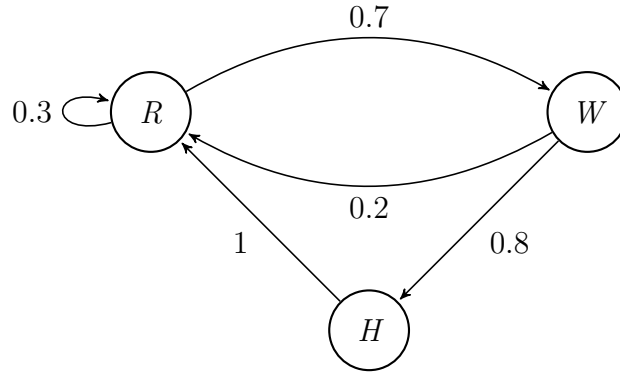
if and only if X is a Markov chain.

As mentioned previously in this section all Markov chains share an important attribute, the state arrived at in a step, denoted by X_n , is solely dependent on the state that it arrived from, X_{n-1} . This property is known as the Markov property.

Markov chains can be depicted in transition diagrams such as the one below where p and q represent probabilities.



Example 5.1. We can apply a real problem to the previous diagram. Say for example that an athlete is trying to predict their exercise routine for the following days. Each day there are three options, the athlete could go for a run, R , they could go for a walk, W , or they could stay home, H . Hence the state space S is made up of $\{R, W, H\}$. By looking at their past daily routine they are able to calculate the probabilities of each event occurring given the exercise from the day before. They produce a diagram of the state space showing the corresponding probabilities:



First the athlete builds the transition matrix (made up of probability vectors). Note that each row adds to 1.

$$P = \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.2 & 0 & 0.8 \\ 1 & 0 & 0 \end{bmatrix}$$

As the athlete decided to stay home today they then define the initial state vector, X_0 .

$$X_0 = [0 \quad 0 \quad 1]$$

The athlete can already tell by looking at the sequence diagram that in this case the following state they arrive at must be R. However, to do this mathematically they must multiply the current state vector, X_0 , by the probability matrix, P to get the probabilities for day 1, X_1 .

$$X_1 = X_0 P$$

$$X_1 = [0 \quad 0 \quad 1] \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.2 & 0 & 0.8 \\ 1 & 0 & 0 \end{bmatrix} = [1 \quad 0 \quad 0]$$

The resulting vector, X_1 , now confirms that the probability that the athlete will go for a run the day after they stay at home is 1. They continue this formula onto the next state to find out which exercise they are likely to be doing on day 2, X_2 ,

$$X_2 = X_1 P = [1 \quad 0 \quad 0] \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.2 & 0 & 0.8 \\ 1 & 0 & 0 \end{bmatrix} = [0.3 \quad 0.7 \quad 0]$$

From this the athlete knows there is 70% chance that he will go for a walk on day 2 and a 30% chance of going for another run. The athlete can now repeat this formula any number of times, n , to determine what exercise they will be doing on a given day, X_n .

5.1 Periodicity

A cycle in a Markov chain describes the transitions from a given state back to that same state, the cycle length is the number of transitions required. The greatest common divisor of the lengths of every possible cycle in a chain gives us the *period* of the Markov chain. Take the chain from example 5.1, the smallest cycle for state C would be 3, C to A to B back to C . The smallest cycle for A however is 1 as it has a self-transition, therefore no matter what other cycle lengths there are the gcd will remain 1 and the period for this chain is 1.

Definition 5.3. A Markov chain, P , is aperiodic if for all i, j where $P_{i,j}^n > 0$

$$\gcd(n) = 1.$$

A state is *aperiodic* if it has period 1 and is otherwise described as periodic. Hence state A in the previous example is an aperiodic state. If every state in a Markov chain is aperiodic the entire chain is aperiodic. If a Markov chain is aperiodic it cannot get caught in a single cycle which can be a very useful property when it comes to the applications of Markov chains.

Note. An easy way to make a chain aperiodic is by adding a self-transition at every state.

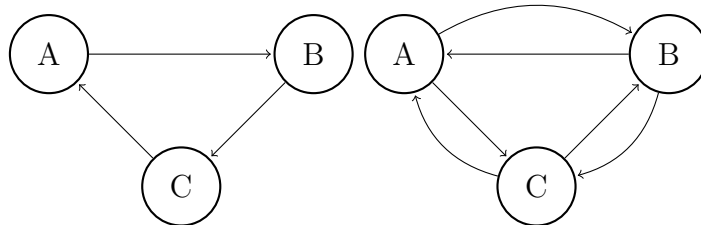


Figure 6: An example of a periodic (left) and aperiodic (right) Markov chain.

5.2 Reducibility

A Markov chain is *reducible* if there exists a state, known as the transient state, which cannot be reached from other states, known as the recurring states.

A Markov chain is irreducible if its period is 1. A chain's irreducibility means that each state can be reached by any state through a finite number of transitions.

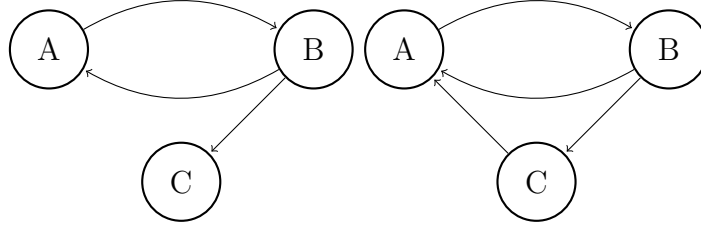


Figure 7: A reducible (left) and irreducible (right) example of the first Markov chain example. The reducible chain can be reduced into a chain consisting of state A and B , and another chain consisting of just state C .

Definition 5.4. A Markov chain, P , is irreducible if for all i, j , there exists n such that

$$P_{i,j}^n > 0.$$

Note. It is easy to get aperiodicity and irreducibility confused. A chain is aperiodic if each of its states has period 1, whereas a chain is irreducible if the period of the whole chain is 1 (an example is shown in figure 8).

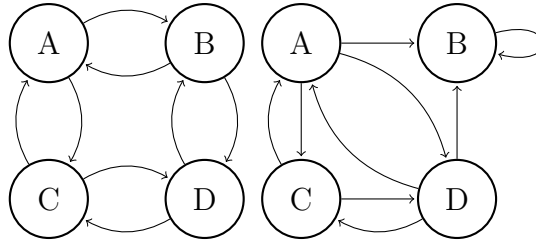


Figure 8: An example of an periodic irreducible Markov chain (left) and a aperiodic reducible Markov chain (right).

5.3 Recurrence and Transience

All states can either be transient or recurrent. As mentioned in the previous subsection if a Markov chain has both recurrent and transient states, it can be reduced to two separate chains.

A state is recurrent if it can be reached by all other states (through any number of steps).

Definition 5.5. Let X be a Markov chain with a transition matrix, P . i is a recurrent state if

$$P_i(X_n = i \text{ for infinitely many } n) = 1$$

A state is transient if it cannot be reached by all other states (through any number of steps).

Definition 5.6. Let X be a Markov chain with a transition matrix, P . i is a transient state if

$$P_i(X_n = i \text{ for infinitely many } n) = 0$$

.

5.4 Ergodicity

Markov chains are described as *ergodic* if they are both aperiodic and irreducible. If a Markov chain is ergodic it must have a unique *stationary distribution* (see Section 5.5). Ergodicity is a very useful property for a Markov chain when it comes to applications. We can see why if we take a look at ergodic theory.

To understand the ergodic theory around Markov chains we must first look at the following theorem:

Theorem 5.2 (Strong Laws of Large Numbers). Let Y_1, Y_2, \dots be a sequence of independent and identically distributed (i.i.d), nonnegative, random variables with a mean, $E(Y) = \mu$ that may be infinite

$$P\left(\frac{Y_1 + Y_2 + \dots + Y_n}{n} \rightarrow \mu \text{ as } n \rightarrow \infty\right) = 1$$

We can create a variable, V_i , to count the number of times state i is visited:

$$V_i = \sum_{n=0}^{\infty} 1_{(X_n=i)}$$

This allows us to look at the *Ergodic Theorem for Markov Chains*.

Theorem 5.3 (Ergodic Theorem for Markov Chains). If P is an irreducible Markov chain then

$$P\left(\frac{V_i(n)}{n} \rightarrow \frac{1}{m_i} \text{ as } n \rightarrow \infty\right) = 1$$

where $n \geq 0$, $V_i(n)$ is the number of visits to state i before step n . m_i is the expected return time (number of steps) to arrive back at state i .

$\frac{V_i(n)}{n}$ is the proportion of time spent on state i over n steps which converges to the expected proportion of time spent on state i . This only occurs in irreducible Markov chains as each of their states can be reached by any other state in a finite number of steps, hence each state in the chain can be regularly visited. This may seem to be an obvious observation, however it is a surprisingly useful feature, especially when we add the property of aperiodicity.

As stated previously (see Section 5) in Markov chain that is aperiodic the chain will not get caught in specific cycles around the same states. Thus, if we take a chain that is both irreducible *and* aperiodic, we know from these properties and the *Ergodic Theorem for Markov Chains* that each state will be visited regularly. This leads us to probably the most useful feature of ergodic chains. If you were to leave an ergodic chain running for enough steps it would eventually visit every state in the state space at least once.

5.5 Stationary Distributions

After a certain number of steps most (but not all) Markov chains will reach a *stationary distribution* (also known as steady state or equilibrium state) after which at each step the probabilities of landing on a specific state remain the same. Professor and mathematician Daryl DeFord states [14]: “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”.

Definition 5.7. *Let P be an ergodic Markov chain and π a probability vector. π is the stationary distribution of P if*

$$\pi = \pi P.$$

We can write this another way [20], if P is an ergodic Markov chain taking values from S and with stationary distribution π then

$$\pi_j = \sum_{i \in S} \pi_i P_{i,j}$$

for all $i, j \in S$.

Note. *We can see from these equations that π is the left eigenvector of P and must have eigenvalue 1.*

To know which Markov chains have this feature we take a look at the following theorem [17]:

Theorem 5.4 (Fundamental Theorem of Markov Chains). *Let $P_{i,j}$ be a Markov chain indexed by the finite set S . If $P_{i,j}$ is both aperiodic and irreducible it has a unique stationary distribution, π . Hence there exists n such that $P_{i,j}^n > 0$ for all $i, j \in S$, and*

$$P_{i,j}^n \rightarrow \pi_j \text{ as } n \rightarrow \infty.$$

5.6 Reversibility

A Markov chain is reversible if the probability of transitioning from i to j is equivalent to the probability of transitioning from j to i . For this to occur the chain must have an *invariant* measure, stationary distribution π , that satisfies a symmetry condition known as detailed balance [14].

Definition 5.8. *Let X be a Markov chain taking values from S . X is reversible if there exists an invariant measure, π , on S that satisfies the detailed balance equations*

$$\pi_i p_{i,j} = \pi_j p_{j,i}$$

where $i \in S$ and $i \neq j$.

6 The Monte Carlo Method

Monte Carlo sampling is a method of taking random data with an unknown or irregular distribution and creating a simulation to approximate the expected value. As stated by Daryl DeFord in his paper on the mathematics of MCMC [14], “Monte Carlo methods are built to take advantage of the fact that some things are hard to compute exactly but easy to evaluate as individual examples”. That is exactly what this process does, rather than trying to compute a difficult equation or problem to find an exact expected value, it takes individual samples and evaluates each of their expected values to find an overall average or prediction.

An easy way to grasp how this works is by splitting the process into steps [21]:

1. **Find input sample.** The first step to Monte Carlo sampling is to define the sample distribution from which data will be taken from. These are all the selection of random points from which we wish to find an expected value or distribution.
2. **Build the model.** Once the input data is gathered a measure needs to be found for each individual sample. A deterministic model is created around these measures that recreates the real scenario to be able to predict the expected outcome. This is where Monte Carlo methods become incredibly useful as rather than finding a system for all of the input data (which could be a very large amount) a model is generated that can be used on individual samples one at a time.
3. **Repeat many times.** This is the key step to Monte Carlo, where the simulation comes in. In this step you take your individual samples and simulate each them on the deterministic model to create an output value for each input value. This section is measured in steps, one step for each value that is simulated. As the overall result is drawn from these random

value simulations, the more steps completed, the more accurate and realistic the final result.

4. **Analyse output data.** Finally we are left with this new set of data which does not form a single outcome hence it is analysed and often averaged (this is often by finding the mean however occasionally the median or mode are used) to find a result. The type of statistical analysis used varies as it is dependent on what the scenario is and what type of outcome is required.

Example 6.1. *The idea of Monte Carlo sampling stemmed from a Polish-American scientist, Stanislaw Ulam, who was curious about the probability of winning a type of solitaire game [14]. In this game the player could not make any moves that would change the end result and hence the outcome, win or lose, was solely dependent on how the pack was shuffled. If we look at all the possible ways there are to shuffle a deck we get an incredibly large number, $52! \approx 8 \times 10^{67}$. A general solution to this problem is hard to find due to the complexity of the game and because of the sheer amount of ways there are to shuffle a deck, however Ulam was able to easily compute whether single games were winnable and hence calculated a probability from that.*

1. **Find input sample.** *The input sample group would be each way you can shuffle the deck of cards (it would be large but a smaller number could be selected from the whole group).*
2. **Build the model.** *The model consisted of a computation designed by Ulam that decided whether the game was winnable or not.*
3. **Repeat many times.** *The model game is repeated using a newly shuffled deck each time, and the outcome is computed.*
4. **Analyse output data.** *The outcome data points are averaged to find the percentage of games won over the course of a certain number of steps, this percentage will give us the approximate probability of winning the game. This was also attempted using a similar simple non-deterministic solitaire game (see figure 9).*

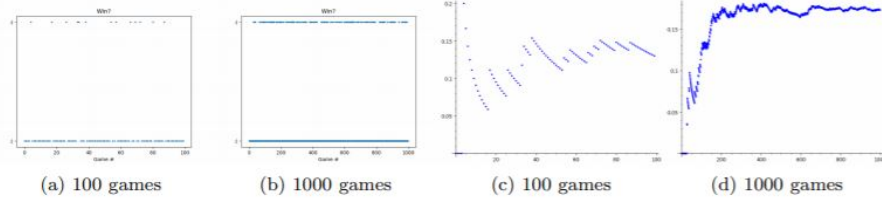


Figure 9: The first two graphs show the output data from 100 and then 1,000 steps and data points. The last two pictures show us the average times the game was won over all the steps, the line gradually tends to the final probability which in this case was 0.165 [22].

Example 6.2. The value of π has also been found to be possible to estimate through Monte Carlo methods [23]. This is done by taking a circle with radius 1 so that its area is equal to $\pi r^2 = \pi$ and checking if random points lie within or outside the circle.

$$\frac{\text{Points inside circle}}{\text{Points inside square (total points)}} = \frac{\text{Circle area}}{\text{Square area}} = \frac{\pi}{4}$$

1. **Find input sample.** The input sample for this problem is all the randomly allocated points across the area of the square as shown in figure 10.
2. **Build the model.** We know that a point, (x, y) lies within the circle if $\sqrt{x^2 + y^2} \leq 1$.
3. **Repeat many times.** Now we have a model scenario, for each point in the sample group we simply make this calculation and record whether or not it lies within the circle.
4. **Analyse output data.** We take our previous formula and divide the points inside the circle by the total number of points in the whole square giving us $\frac{\pi}{4}$. Once this result is multiplied by 4 we have our estimate for π .

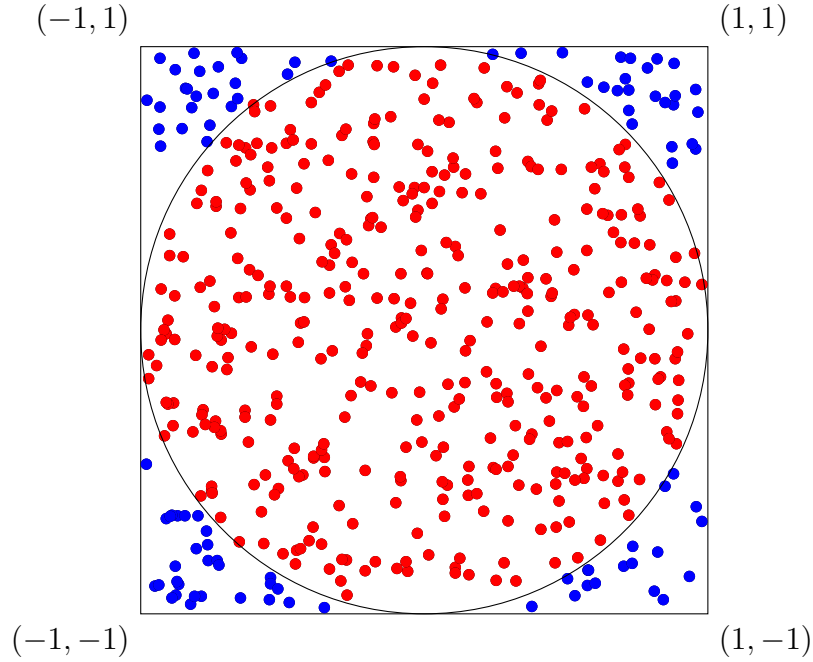


Figure 10: Circle ($radius = 1$) within a square with random points within the circle shown in red and points outside the circle shown in blue.

7 Markov Chain Monte Carlo

Markov Chain Monte Carlo methods are algorithms used to create the ideal distributions to sample from. The name is a combination of *Markov Chain* and *Monte Carlo*, as samples are drawn at each step and rejected or accepted similarly to classic Monte Carlo (the biggest difference being that in MCMC each data sample drawn is dependent on the last rather than using completely random data points). This is done by updating each point in the chain, either with no change or with a new point as replacement. The most common and famous algorithm in this collection of methods being the *Metropolis-Hastings algorithm* [26].

7.1 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm originated in 1953 as part of an article by Nicholas Metropolis, Marshall and Arianna Rosenbluth, and Edward and Augusta Teller [25] and has now become one of the most used statistical techniques amongst statisticians [26]. In this article they studied the energy of particles, and in particular how n particles would configure themselves in a square. This experiment has been recreated by using the analogy of hard-discs in a box [24] (see

figure 11), and is a simplified but helpful way to understanding the full algorithm (shown after example).

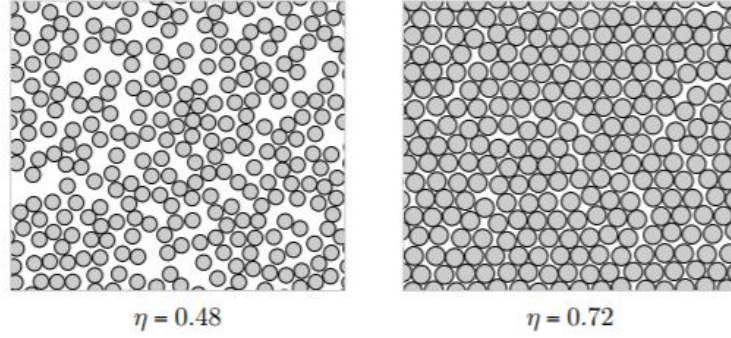


Figure 11: An example of the hard-disk problem, with a low density (left) and high density (right) [24]

Example 7.1. *An approximate process for this experiment is listed as follows [17].*

1. *Start on the centre of a disc.*
2. *Pick a proposed point around that disc.*
3. *Move the centre of the disc to the proposed point.*
4. *If the proposed point does not create an overlap accept, otherwise reject.*
5. *Repeat this for all discs in square (see figure 12).*

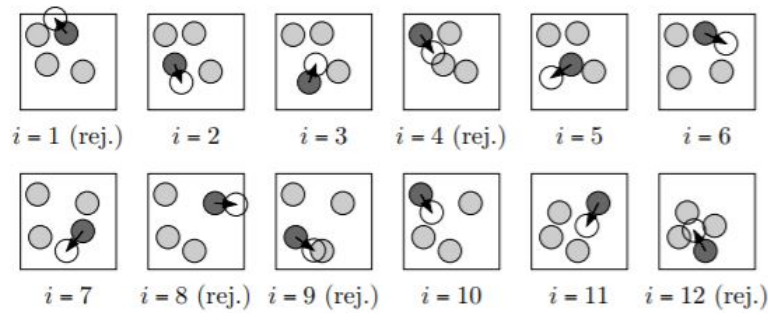


Figure 12: A simple depiction of how the algorithm would work, i represents each repetition or iteration of the algorithm for each disc in the square[24]. As we can see certain moves are rejected, such as $i = 1, 4, 8, 9, 12$, whereas others are accepted.

The full Metropolis-Hastings algorithm is shown below [27].

1. Initialize $X_n = x$.
2. Generate proposed state \hat{x} according to $P_{x,\hat{x}}$
3. Compute acceptance probability

$$\alpha = \min \left(1, \frac{\pi(\hat{x})}{\pi(x)} \frac{P_{\hat{x},x}}{P_{x,\hat{x}}} \right)$$

4. Pick a number u uniformly on $[0, 1]$
5. Set

$$X_{n+1} = \begin{cases} \hat{x} & \text{if } u < \alpha \\ x & \text{otherwise} \end{cases}$$

The variable x represents the state we are in and \hat{x} the proposed state to move to. Hence $\frac{\pi(\hat{x})}{\pi(x)}$ is looking at whether the probability of being at the proposed state \hat{x} is smaller or larger than the probability of being at the current state x . Similarly, but not quite the same, $\frac{P_{\hat{x},x}}{P_{x,\hat{x}}}$ looks at whether the probability of moving from state \hat{x} to x is smaller or larger than the probability of moving from x to \hat{x} .

Recall that if we have an ergodic Markov chain, it will reach a stationary distribution such that $\pi = \pi P$. The acceptance formula measures if $\pi(\hat{x})P_{\hat{x},x}$ is larger than $\pi(x)P_{x,\hat{x}}$, if it is then $\alpha = 1$ and $X_{n+1} = \hat{x}$ (apart from in the rare case that $u = 1$). If it is not, then $\alpha = \frac{\pi(\hat{x})P_{\hat{x},x}}{\pi(x)P_{x,\hat{x}}}$ and the lower that α is the more likely it is that $X_{n+1} = x$ (i.e. the state remains as it is).

Note. Due to the randomness of u there is always a chance that $X_{n+1} = \hat{x}$ even if α is very low, or that $X_{n+1} = x$ when α is very high. This is simply a way of increasing the element of randomness and to fully view the range of the whole state space.

8 Redistricting

There are too many possible maps to produce every possibility, instead large ensembles of plans are created containing samples (this allows us to still view the full range of possibilities without having to recreate every single plan).

8.1 Area representation

Dual graphs are used to represent U.S. states (as shown in figure 14), or any area required, and are defined as follows [28].

Definition 8.1. A dual graph $G = (V, E)$ where V represents the set of vertices and E the set of edges, is made up of a planar graph G and its geometric dual G^* , which is constructed by placing a vertex v in each region of G (including the exterior region) and, if two regions have an edge e in common, joining the corresponding vertices by an edge.

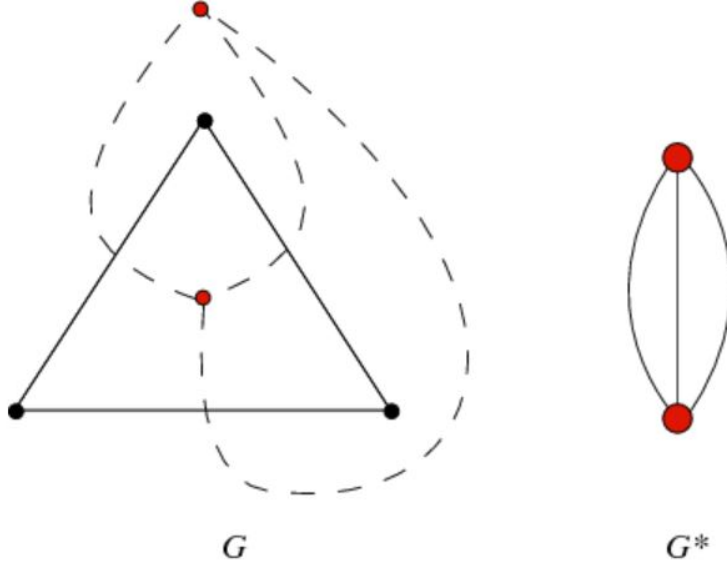


Figure 13: A simple example of a dual graph [28]. On the left the planar graph G and on the right it's geometric dual G^* . How G^* would configure itself on G is shown by the dotted line.

So, taking a look at the dual graph, each vertex v is a node on the graph, and the edges e are what connects each vertex [29].

This dual graph can be k -partitioned to create k districts, these partitions are represented as a collection of disjoint subsets, $\{V_1, V_2, \dots, V_k\}$, each containing a number of vertices. These subsets should cover the entire dual-graph, $\bigcup_{i=1}^k V_i = V$. A set of all possible redistricting plans is written as $\mathcal{P}_k(G)$.

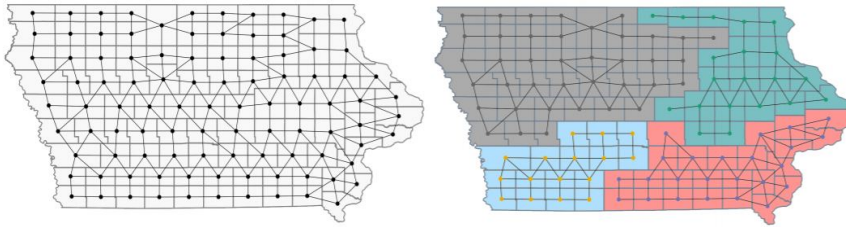


Figure 14: Example dual graph for the state of Iowa. The left picture shows the current congressional districts [27].

8.2 Implementing the rules

There are several laws that every voting district map must abide by (as mentioned in section 4). These are listed in DeFord, Duchin and Solomon’s paper on Markov chains for redistricting [27], these are as follows:

- **Population balance.** All districts should have a similar population size.
- **Contiguity.** Each district must be entirely connected.
- **Compactness.** Each district should have a compact and regular shape i.e. there should not be too many edges bordering two separate districts.
- **Splitting rules.** Most districts have counties, municipalities and/or other areas of interest that they wish to be kept in the same district rather than ‘split’ during redistricting.
- **Voting Rights Act (VRA).** Districts must be made so that minority groups (race, age, etc.) have voting power.
- **Neutrality.** The redistricting plan must be entirely neutral, i.e. there should not be any bias when drawing the new district lines.

Rules such as neutrality can easily be implemented, the associated is simply not included in the process. Others are harder to enforce.

Each node does not have an equal population density hence to calculate the *population balance* we need to know population weightings on each node hence we create a function $w : V \rightarrow \mathbb{R}$ to represent this. Using this and unit of error, ε (it is very difficult to have an exact balance of population hence a slight error is included, it is usually around 0.05) the population balance can be controlled. The lower and upper bound being

$$(1 - \varepsilon) \frac{\sum_V w(v)}{k} \leq |V_i| \leq (1 + \varepsilon) \frac{\sum_V w(v)}{k}$$

where $v \in V$ and i denotes an index between 1 and k .

Let P be a labelling function, $P : V \rightarrow \{1, 2, \dots, k\}$, for the k -partitions of G , so that if for example $u \in V_1$ then $P(u) = 1$ where $u \in V$. We can now measure the *compactness* of the plan by finding the cut edge count, $|\partial P|$. The set of cut edges can be calculated as follows

$$\partial P = \{(u, v) \in E : P(u) \neq P(v)\}$$

where $u, v \in V$.

Using the previous calculations *contiguity* can now also be implemented. We know that two nodes of the same district are connected if

$$P(u) = P(v)$$

where $(u, v) \in E$. Hence we can check this factor for each node of every district.

All these equations are applied to the state space $\mathcal{P}_k(G)$ of all redistricting maps, filtering out any maps that would not be accepted by the court, and leaving us with a (very large) collection of valid redistricting possibilities.

8.3 MCMC and random walks

Now we have our dual graph as our area to be partitioned, and our rules surrounding the partitions, where do the Markov chains come in? If we look at $\mathcal{P}_k(G)$ as the entire state space, we can see the Markov chain as the stochastic process of moving between different redistricting plans one node at a time. A Markov chain in which we perform a *random walk*. A random walk starts at some state i and moves to state j at probability $p_{i,j}$ (from the Markov probability matrix, see section 5).

The Metropolis-Hastings algorithm is used to check the plan at each state and see if there is a better state for it to switch to for sampling. However, x will only switch to the proposed state \hat{x} if $\hat{x} \in \mathcal{P}_k(G)$ (and hence abides by all the redistricting laws stated in the section above).



Figure 15: In a recent paper on redistricting using different types of Markov chains [27], an MCMC process known as *ReCom* is run on 4 districts on a dual graph of Arkansas. Starting at the initial partition (first map), the state of the map is shown every 2,500 steps until 10,000 (final map). This implementation has a population bound error of 5% ($\varepsilon = 0.05$) and up to 5% cut edges (compactness constraint).

8.4 Mixing times

The mixing time of a Markov chain determines the number of steps a chain must take, starting at any initial state, for it to reach its stationary distribution. When finding Markov chains for redistricting, a low mixing time is very important for efficiency and to guarantee that the chain will always reach its stationary distribution. To fully understand the mathematical meaning of mixing times several prior definitions must be made. Firstly we define *total variation distance*, which is a probabilistic maximum distance between two different probability distributions [20]:

Definition 8.2. Let μ and ν be probability distributions on finite state space S . The total variation distance between μ and ν is

$$\|\mu - \nu\|_{TV} = \max_{A \subseteq S} |\mu(A) - \nu(A)|.$$

Note. In this definition $\mu(A) = \sum_{x \in A} \mu(x)$ [32].

Now we are able to calculate the distance between two distributions, we can find the distance between a Markov probability distribution and the stationary distribution.

Definition 8.3. If there exists an irreducible Markov chain with transition matrix P and stationary distribution $\pi \in S$, then we define the distance from stationarity as

$$d(t) := \max_{x \in S} \|P^t(x, \cdot) - \pi\|_{TV}$$

where t is the number of steps.

So, mixing times can fully be defined as follows:

Definition 8.4. If there exists an irreducible Markov chain with transition matrix P and stationary distribution $\pi \in S$, then we define the mixing time as

$$t_{mix}(\epsilon) := \min\{t : d(t) \leq \epsilon\}$$

where $\epsilon \in [0, 1]$.

From these definitions we can describe the mixing time as the smallest number of steps, t , required such that the distance from stationarity is less than a certain parameter, ϵ . Usually, to find the mixing time of a Markov chain, $\epsilon = \frac{1}{4}$ is used [20]. Finally, we get

$$t_{mix} := t_{mix}(\frac{1}{4}).$$

8.5 Flip and ReCom

As mentioned in section 3.3 there are different approaches to Markov chains for redistricting, each one attempting to find the best selection of maps with the lowest mixing time [9] [11]. One of the first and more simple random walks created for redistricting was the *Flip* chain [12].

At each step the Flip chain proposes a *node flip* where an edge node from a district is switched to its neighbouring district. The resulting state is known as the *flip proposal* and through MCMC is then either rejected or accepted depending on the constraints that are set (population bound, contiguity, compactness, etc.).

This appears to be a logical way of redistricting however it does require a large amount of steps to reach a the stationary distribution, or in other words it

has a very high *mixing time*. I created a simple program using [MGGG](#)'s open source python package, [GerryChain](#) to view the results of the Flip redistricting method which can be seen in figure 16.



Figure 16: *Flip* implemented on a simple 40×60 grid. The result of the partition is shown every 50,000 steps starting with the initial partition at step 0 and finishing with the resulting map at 250,000 steps.

Another technique I studied was the more recent *ReCom* chain [12]. The ReCom proposal function is far more complex than that of the Flip chain and hence the process requires less steps and has a much lower mixing time. Rather than proposing a simple node flip, at each step 2 connected districts are picked which are merged together before, via a spanning tree, a whole new partition between the two districts is made. This means that the map evolves and changes much more rapidly hence reaching its stationary distribution much quicker. As I did with Flip I attempted to perform this technique, the results can be viewed in figure 17.

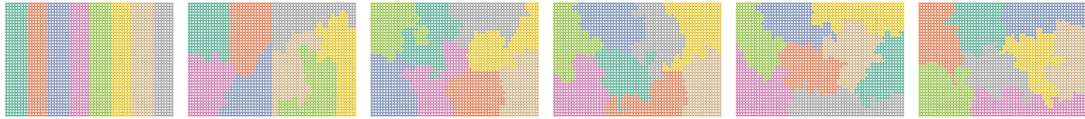


Figure 17: *ReCom* implemented on a simple 40×60 grid. The result of the partition is shown every 20 steps starting with the initial partition at step 0 and finishing with the resulting map at 100 steps.

As we can see by figure 16 and figure 17 the flip algorithm takes a far larger amount of steps (approximately 2,500 as many) to reach a valid state compared to ReCom. My full program implementing the two techniques can be found [here](#).

8.6 Outlier Analysis

Finally, we have our ensemble of valid redistricting plans that accurately represent the range of all the possibilities. This can now be used for analysis. Often the purpose is to check if a map accurately represents the public voting data, or if it appears to be a gerrymander. In this case the common procedure would be to produce a outlier analysis graph from the resulting ensemble. If the map

being challenged appears to be a severe outlier, it is highly likely to be a case of gerrymandering. With the aid of MGGG’s open source [GerryChain](#) Software, I created a small ensemble of 100 ReCom maps (each having taken 100 steps) and created a histogram of the results, as seen in figure 18. This is an extremely simplified version of the full process however the result showed that a normal distribution around the mean was starting to form. The full code can be viewed [here](#).

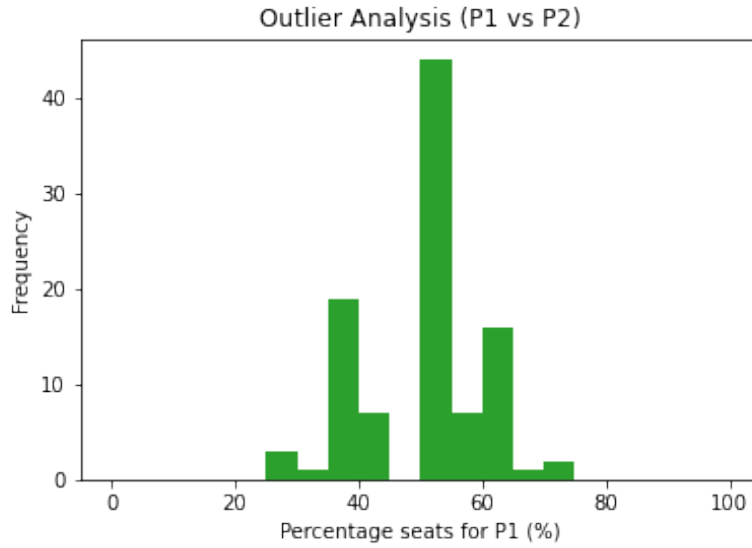


Figure 18: The histogram above displays the results from an election between hypothetical parties $P1$ and $P2$. Each party started with exactly 50% of the votes over the whole voting area. The area was then partitioned into 8 districts and 100 maps were created each displaying a different result after redistricting. This analysis shows that the curve is still centred around the mean of 50% seats for each party, any of the maps showing a result of $x < 30\%$ or $x \geq 70\%$ were further from the mean than 95% of the other maps and hence could be considered outliers.

9 Conclusion

Despite being a rather recent tool, redistricting via MCMC sampling appears to be gaining increasingly more attention and praise. Although not being a compulsory part of redistricting yet, MCMC approaches have been used in court. One example being Jonathan Mattingly, whose analysis demonstrated that the enacted plan of North Carolina was an extreme outlier based on his own approach to MCMC [10].

This field is constantly progressing as mathematicians are continuously attempting to find more efficient algorithms for handling the issues surrounding redistricting, particularly in terms of decreasing mixing times and incorporating the required constraints [33] [34]. However my analysis of the techniques used shows that MCMC succeeds in the difficult task of gathering a valid and broad range of samples that are representative of the incalculable number of possibilities that exist. This could be the key to finding a mathematical solution to the societal real world problem of detecting gerrymandering.

Summary

This paper discussed how MCMC methods work and gives a thorough mathematical background to Markov chains, Monte Carlo simulations and the Metropolis-Hastings algorithm:

- *Markov chains* are stochastic processes on a finite set S , commonly referred to as the state space. Markov chains can be represented by a probability (or transition) matrix, P , and a vector, X . In this sequence each state X_n solely depends on the state before it X_{n-1} .
- The goal of *Monte Carlo simulations* is to approximate an expected value from an unknown distribution. This is done by sampling from random data points in the distribution and finding the expected value of each point one by one, through a simulation of the scenario.
- The *Metropolis-Hastings algorithm* is a Markov Chain Monte Carlo method that draws samples from a Markov chain and attempts to propose a new sample each time. At each step the proposed state is either accepted or rejected. This process results in a Markov chain with a valid and representative stationary distribution that can be used in analysis.

Combinations of these techniques are used to produce a representative range of valid redistricting possibilities across a certain area. This ensemble can then be used to analyse currently used maps and to support court action when a case of gerrymandering may be occurring. A simple set of steps for the process of checking redistricting plans is shown below.

1. An initial partition is created on a dual graph of the area.
2. Markov chains in the form of random walks are run on the initial plan.
3. Node by node the districts are altered.
4. The Metropolis-Hastings algorithm is used to check whether the plan is valid ($\hat{x} \in \mathcal{P}_k(G)$) at each step.

5. The algorithm is halted once enough steps have been completed for the chain to reach a stationary distribution.
6. An ensemble of valid redistricting maps created from samples.
7. Analysis performed on ensemble.

Acknowledgements. *I would like to thank Dr Graeme Wilkins for taking the time to support my project with his advice and feedback. In addition, a thank you to the organisation [MGGG](#) for providing an immense amount of information on the MCMC methods and redistricting. In particular their founder Moon Duchin, and MGGG collaborator Daryl DeFord who have published many interesting and incredibly useful papers on the topic.*

Appendices

The raw Python code created to implement the Flip and ReCom chain on a grid and to create the outlier analysis is shown below. The full details and instructions on how to use this code can be found on my [GitHub Repository](#). Please note, that if used this code may deliver different results to those shown in the text due to the random element of Flip and ReCom.

```
from gerrychain import MarkovChain
from gerrychain.constraints import (
    Validator,
    single_flip_contiguous,
    within_percent_of_ideal_population,
    UpperBound)
from gerrychain.proposals import propose_random_flip
from gerrychain.accept import always_accept
from gerrychain.updaters import Election, Tally, cut_edges
from gerrychain.partition import Partition
from gerrychain.proposals import recom
import random
import matplotlib.pyplot as plt
from functools import partial
import networkx as nx

k_partitions = 8
node_size = 30
grid_height = 40
grid_width = 60
```

```

graph = nx.grid_graph([grid_height, grid_width])

random_boundary = 0.50
P1 = "skyblue"
P2 = "navy"
for node in graph.nodes():
    graph.nodes[node]["population"] = 1
    if random.random() < random_boundary:
        graph.nodes[node][P1] = 1
        graph.nodes[node][P2] = 0
    else:
        graph.nodes[node][P1] = 0
        graph.nodes[node][P2] = 1
vote_dict = {1: P1, 0: P2}

def step_num(partition):
    parent = partition.parent
    if not parent:
        return 0
    return parent["step_num"] + 1

updaters = {
    "population": Tally("population"),
    "cut_edges": cut_edges,
    "step_num": step_num,
    "P1 vs P2": Election("P1 vs P2", {"P1": P1, "P2": P2}),}

part_dict = {x: int(x[0]/(grid_width/k_partitions))
for x in graph.nodes()}
init_part = Partition(graph,
    assignment=part_dict,
    updaters=updaters)

def p1_wins(partition):
    if partition["P1 vs P2"].wins("P1")+
partition["P1 vs P2"].wins("P2") > k_partitions:
        return equal_check(partition, "P1")
    else:
        return (partition["P1 vs P2"].wins("P1")
/k_partitions)*100

def p2_wins(partition):
    if partition["P1 vs P2"].wins("P1")+

```

```

partition["P1 vs P2"].wins("P2") > k_partitions:
    return equal_check(partition, "P2")
else:
    return (partition["P1 vs P2"].wins("P2")
            /k_partitions)*100

def equal_check(partition, party):
    for number in range(1, k_partitions+1):
        if partition["P1 vs P2"].wins("P1")+
            partition["P1 vs P2"].wins("P2") ==
            k_partitions + number:
            return
            ((partition["P1 vs P2"].wins(party)/k_partitions)*
             100)-((100/k_partitions)/2)*number
    raise error

ideal_population =
sum(init_part["population"].values()) / len(init_part)
popbound =
within_percent_of_ideal_population(init_part, 0.1)
cutedgebound =
UpperBound(lambda part: len(part["cut_edges"]), 400)

flip_steps = 250000
flip_chain = MarkovChain(
    propose_random_flip,
    Validator([single_flip_contiguous, popbound, cutedgebound]),
    accept = always_accept,
    initial_state = init_part,
    total_steps = flip_steps, )

flip_p1_seats = []
for part in flip_chain:
    flip_p1_seats.append(part["P1 vs P2"].wins("P1"))
plt.figure()
    nx.draw(
        graph,
        pos = {x: x for x in graph.nodes()},
        node_color = [dict(part.assignment)[x]
for x in graph.nodes()],
        node_size = node_size,
        node_shape = "p",
        cmap = "Set2",)

```

```

plt.show()

tree_proposal = partial(
    recom,
    pop_col = "population",
    pop_target = ideal_population,
    epsilon = 0.1,
    node_repeats = 1,)
recom_steps = 100
recom_chain = MarkovChain(
    tree_proposal,
    Validator([popbound]),
    accept = always_accept,
    initial_state = init_part,
    total_steps = recom_steps,)

recom_P1_seats = []
for part in recom_chain:
    recom_P1_seats.append(part["P1 vs P2"].wins("P1"))
plt.figure()
nx.draw(
    graph,
    pos = {x: x for x in graph.nodes()},
    node_color = [dict(part.assignment)[x]
for x in graph.nodes()],
    node_size = node_size,
    node_shape = "p",
    cmap = "Set2",)
plt.show()

seats_won = []
for i in range(1, 101):
    seats_won.append(p1_wins(part))
    recom_P1_seats = []
    for part in recom_chain:
        recom_P1_seats.append(part["P1 vs P2"].wins("P1"))
plt.plot(100, 2, 3)
plt.hist(seats_won)
plt.title("Outlier Analysis (P1 vs P2)")
plt.xlabel("Percentage seats for P1 (%)")
plt.ylabel("Frequency")
plt.show()

```

References

- [1] Jowei Chen and David Cottrell, *Evaluating partisan gains from Congressional gerrymandering: Using computer simulations to estimate the effect of gerrymandering in the US House*, Electoral Studies **44** (2016), 329–340.
- [2] Ron Johnston, *Which Map? Which Government? Malapportionment and Gerrymandering, UK-Style:(The Government and Opposition/Leonard Schapiro Memorial Lecture, 2014)*, Government and Opposition **50** (2015), no. 1, 1–23.
- [3] Ron Johnston, David Rossiter, and Charles Pattie, *When Is a Gerrymander Not a Gerrymander: Who Benefits and Who Loses from the Changed Rules for Defining Parliamentary Constituencies?*, The Political Quarterly **88** (2017), no. 2, 211–220.
- [4] Larry M Schwab, *The impact of the 1980 reapportionment in the United States*, Political Geography Quarterly **4** (1985), no. 2, 141–158.
- [5] Zeph Landau, Oneil Reid, and Ilona Yershov, *A fair division solution to the problem of redistricting*, Social Choice and Welfare **32** (2009), no. 3, 479–492.
- [6] Daryl DeFord, Moon Duchin, and Justin Solomon, *A Computational Approach to Measuring Vote Elasticity and Competitiveness*, Statistics and Public Policy **7** (2020), no. 1, 69–86.
- [7] Kristopher Tapp, *Measuring political gerrymandering*, The American Mathematical Monthly **126** (2019), no. 7, 593–609.
- [8] Moon Duchin, *Gerrymandering metrics: How to measure? What’s the baseline?*, arXiv preprint arXiv:1801.02064 (2018).
- [9] Daniel Carter, Gregory Herschlag, Zach Hunter, and Jonathan Mattingly, *A merge-split proposal for reversible monte carlo markov chain sampling of redistricting plans*, arXiv preprint arXiv:1911.01503 (2019).
- [10] Moon Duchin, *Geometry v. Gerrymandering*, The Best Writing on Mathematics 2019 **10** (2019), 1.
- [11] Gerdus Benade and A Procaccia, *Abating gerrymandering by mandating fairness*, Preprint (2020).
- [12] Daryl DeFord, Moon Duchin, and Justin Solomon, *Recombination: A family of Markov chains for redistricting*, arXiv preprint arXiv:1911.05725 (2019).
- [13] Gregory Herschlag, Robert Ravier, and Jonathan C Mattingly, *Evaluating partisan gerrymandering in Wisconsin*, arXiv preprint arXiv:1709.01596 (2017).
- [14] Daryl DeFord, *Introduction to Discrete MCMC for Redistricting* (2019).
- [15] Paul A Gagniuc, *Markov chains: from theory to implementation and experimentation* (2017).
- [16] Richard Serfozo, *Basics of applied stochastic processes*, Springer Science & Business Media, 2009.
- [17] Persi Diaconis, *The markov chain monte carlo revolution*, Bulletin of the American Mathematical Society **46** (2009), no. 2, 179–205.
- [18] Chad Casarotto, *Markov chains and the ergodic theorem*, University of Chicago **8** (2007).
- [19] J R Norris, *Markov chains*, Cambridge university press, 1998.

- [20] David A Levin and Yuval Peres, *Markov chains and mixing times*, Vol. 107, American Mathematical Soc., 2017.
- [21] Samik Raychaudhuri, *Introduction to monte carlo simulation*, 2008 Winter simulation conference, 2008, pp. 91–100.
- [22] *Deterministic Solitaire Monte Carlo example*. Accessed: 14-04-2021.
- [23] Rik King and Peter Anderson, *Monte Carlo simulations to estimate Pi*, Electronic Journal of Informatics (2020), 84.
- [24] Werner Krauth, *Statistical mechanics*, Oxford Master Series in Physics. Oxford University Press, Oxford. Algorithms and computations, Oxford Master Series in Statistical Computational, and Theoretical Physics. (2006).
- [25] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller, *Equation of state calculations by fast computing machines*, The journal of chemical physics **21** (1953), no. 6, 1087–1092.
- [26] David B Hitchcock, *A history of the Metropolis–Hastings algorithm*, The American Statistician **57** (2003), no. 4, 254–257.
- [27] Daryl De Ford, Moon Duchin, and Justin Solomon, *Recombination: A family of Markov chains for redistricting* (2020).
- [28] Eric W Weisstein, *Dual graph*, <https://mathworld.wolfram.com/> (2007).
- [29] Lorenzo Najt, Daryl DeFord, and Justin Solomon, *Empirical Sampling of Connected Graph Partitions for Redistricting*, arXiv preprint arXiv:2012.04564 (2020).
- [30] Daryl De Ford, *How to build districting ensembles: A guide to GerryChain* (2019). Accessed: 2021-4-25.
- [31] David A Levin and Yuval Peres, *Markov chains and mixing times*, Vol. 107, American Mathematical Soc., 2017.
- [32] Anuran Makur, *Mixing of Markov Chains*, [Mixing%20of%20Markov%20Chains.pdf](#) (2017).
- [33] Benjamin and Higgins Fifield, *Automated redistricting simulation using Markov chain Monte Carlo*, Journal of Computational and Graphical Statistics **29** (2020), no. 4, 715–728.
- [34] Eric A and Carter Autry Daniel and Herschlag, *Multi-scale merge-split Markov chain Monte Carlo for redistricting*, arXiv preprint arXiv:2008.08054 (2020).