CONGRESSIONAL REDISTRICTING IN THE UNITED STATES

USING MARKOV CHAIN MONTE CARLO TO IDENTIFY PARTISAN GERRYMANDERS

GREGORY W. HEADLEY
201362739

SUPERVISED BY
DR. MATTHEW ALDRIDGE

SEPTEMBER 2020

SUBMITTED IN ACCORDANCE WITH
THE REQUIREMENTS OF THE MODULE

MATH5872M: DISSERTATION IN DATA SCIENCE AND ANALYTICS

AS PART OF THE DEGREE OF

MASTER OF SCIENCE

IN

DATA SCIENCE

AND ANALYTICS

SCHOOL OF MATHEMATICS
UNIVERSITY OF LEEDS

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

For my father, George, who always offered his
encouragement, vision, and curiosity

1963 – 2017

ACKNOWLEDGMENTS

My sincerest thanks goes to the many people who supported me throughout this process. To Hannah, for her constant support, magnanimity around the house, and patience with my working hours. To Melina and Ben, for listening and helping me clarify my ideas. To Raye, Heidi, Jerry, and Jonny for making sure it all made sense. Finally, and most importantly, to Matt, for his engaged and continuous support, and for putting up with my ramblings.

Special thanks also to André Miede and Ivo Pletikosić for the brilliant `classicthesis` package, which was extensively adapted to produce this document.

ABSTRACT

Many, if not all, representative democracies utilise geographic boundaries to define electoral populations. In the system of the United States, on which this work focuses, these so-called districts are subject to abuse. Partisan gerrymandering is the process of drawing electoral districts such that they favour the party authoring the map. With voters changing their political views, party preference, and even physical location, it can be difficult to determine if a district is gerrymandered. We seek to answer the question: how can one identify a partisan gerrymander? To do so, we deploy the technique of Markov chain Monte Carlo to produce a distribution sampled uniformly at random from the set of all legal districting plans. This is done using a bespoke library newly written in `python` by the author. We leverage this distribution in our analysis of a suspected gerrymander to assemble evidence to make our case. We find that, while difficult to prove definitively, enough statistics and tools exist to produce a compelling set of evidence for or against a gerrymander claim. We use these tools to demonstrate that a proposed districting plan is a partisan gerrymander. We conclude with a discussion of additional options available for future work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

VRA	Voting Rights Act
VTD	Voting District
MCMC	Markov chain Monte Carlo
GCD	Greatest Common Divisor
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
GB	Gigabyte

1

INTRODUCTION

PERHAPS THE HIGHEST ideal of a democracy is to ensure that the will of the people is made manifest in the laws of the land. To that end, the electoral process of a democracy is an essential element in making that will heard. Does a democracy then break if the persons favoured for office by the electorate are not the ones elevated to serve in government?

We do not suppose to answer this question directly in this work. We will, however, spend the coming pages looking at one of the ways in which the will of the people is purposefully thwarted. The process of partisan gerrymandering, which will be our primary topic of interest, has received steady and voluminous attention in academic circles in recent decades. In this first chapter, we will look to define it.

With CHAPTER 2, we will describe the mathematical foundations and theory behind Markov chain Monte Carlo (MCMC). This will be our primary tool as we seek to identify partisan gerrymanders. In CHAPTER 3, we will detail our methodology. This will include the description of our adaptation of the Metropolis–Hastings algorithm, which we derived from scratch using `python`. We will see how this algorithm is used to derive a distribution of so-called *districting plans* that can be deployed for analysis. With our distribution in hand, CHAPTER 4 will present our results and analysis. This includes some of the tools available for measuring and identifying gerrymanders. In CHAPTER 5, we will discuss additional such metrics. Moreover, we will look at further considerations and areas of interest that might be valuable in future work.

This document is a hybrid of sorts. Legal and political considerations are blended with the rigidity of mathematics. It offers insight into an interdisciplinary trend. One where statistics can be leveraged to provide quantitative and objective context, where previously only opinion and subjectivity reigned. We hope to put to rest this subjectivity by offering a collection of evidence, grounded in mathematics, which identifies partisan gerrymanders.

1.1 DEMOCRACY IN THE UNITED STATES

An ambitious architect looking to design a governing democratic system might initially be drawn to *direct democracy*. Though such a system can take on different flavours, the essence is the same: the people vote for the policies they wish to be enacted. In contrast, the federal presidential republic created by the United States Constitution is a *representative democracy* [63]. Instead of voting for specific policies, the electorate elevates politicians to serve as their intermediaries. In this regard, politicians are meant to serve as the surrogates of their constituents' interests in the governing bodies of the nation.

While the republican system of the United States allows for a few occasions of direct democracy—in the form of plebiscites, predominantly at the state

level—the vast majority of legislation is driven by representatives. The federal[1] legislative branch, known as Congress, is split into two chambers: the House of Representatives and the Senate [63]. Since it makes little sense to have each legislative member represent the entire nation, individuals are elected to represent subpopulations within the nation.

The Senate chamber is made up of 100 senators while the House is formed by 435 representatives[2] (also known as Congressmen/women) [63]. The subpopulation a given member represents is defined by geography and each chamber has its own geographic definitions which dictate representation. Senators represent the *entire* state population from which they were elected and every state, regardless of population, sends two senators to the Senate [63]. This means that a senator from Wyoming—who represents approximately 550,000 people—has the same legislative power as a senator from California, who represents approximately 40 million people [15].

Conversely, the House of Representatives defines constituent groups by population proportion. That is, each representative stands for a constituency of approximately 1/435th of the nation's total population [63]. These representation blocks are known as *districts* and they are organised at the state level. Specifically, districts are awarded to and defined by individual states based on population. As of the 2010 Census (see SECTION 1.2), any given House district encompasses approximately 711,000 people, according to Burnett [15]. Extending our example, this means that Wyoming elects just one Congressman while California elects 53.

1.2 POLITICAL GEOGRAPHY

Written into the United States Constitution is a mandate to enumerate every 10 years the number of inhabitants in the nation [50], commonly known as the Census. As we saw previously, districts are first and foremost defined by population. This means that population values of the Census are integral to the design of district boundaries.

The results of the Census are used to define: 1) the number of inhabitants required to reside in a district, and 2) the number of districts each state is apportioned [15]. While the Census is administered by the federal government, it constitutes the only element of the redistricting process which occurs at the national level. Once district population size and seat allocations for each state are defined, the practicalities of redistricting proceed at the state level [26].

We have emphasised that districts are defined geographically. In practice, this means each district is formed by an n-sided polygon encompassing a certain geographical subset within a state's boundaries (see FIGURE 1.1). Furthermore, these districts are required to be geographically *contiguous*, meaning that a district cannot be made of multiple discrete polygons [26]. The body which draws these districts after each decennial census varies from state-to-state.

1 While legislatures also exists at the state level for all 50 states, this work is constrained to just the federal legislature.

2 Fixed into law in 1929 [1, 23]

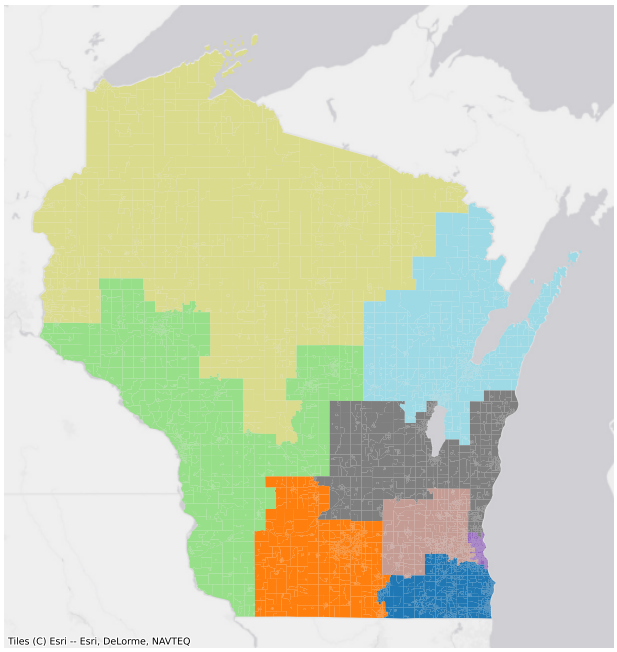


FIGURE 1.1: The eight Congressional districts for the state of Wisconsin as of 2013, with data from Buck and Hully [14].

From the convenient summary prepared by the Brennan Center for Justice [13], we know that for the 2020 Congressional redistricting cycle, the *state* legislatures of 31 states will define district boundaries. In these states, districting plans are passed as conventional legislation and generally the state governor has veto power. Another four states deploy an *advisory commission*, which sometimes includes lawmakers, to recommend district plans for passage by the legislature. Four separate states use an *independent commission* which does not include any public officials or lawmakers. A further four states deploy *commissions of political appointees*, some partisan and others not, to define districts. Missouri modifies this style by making said appointees members of the legislature. Three states define a *backup commission* which draws maps when the legislature fails to or their plan is vetoed by the governor. Lastly, seven states have populations too small to accrue more than one congressional district and hence their singular district defaults to the boundaries of the entire state.

The discerning reader will notice that in the majority of states, districts are defined by politicians. This has led to the cliché that politicians are choosing their voters instead of voters choosing their politicians [60]. This concern is especially warranted in the case of state legislative bodies, which possess and draw separate districts to the Congressional ones on which this work focuses. However, the entrenched two-party system of the United States exists across most levels of government, from local and municipal positions, through state bodies, and up to federal institutions [63]. This means that while a state-level politician will not directly benefit from drawing a congressional district, he or she might be able to induce an advantage for his or her party at the national level [63]. Such concerns of redistricting for partisan advantage will be

developed in detail in SECTION 1.4; however, we will first explore the history of redistricting to establish the rules that apply today.

1.3 A BRIEF HISTORY OF REDISTRICTING

In the modern era, it is taken for granted that districts must be geographically contiguous and uniformly populated, as described in SECTIONS 1.1 and 1.2. However, these sorts of conditions, imposed on the mapmakers of today, are relatively young in comparison to the age of the nation [26]. Indeed, when it comes to the House of Representatives, the United States Constitution lists no specific requirement for single-member districts, population uniformity or other constraints beyond term limits and membership requirements (e.g. minimum age) [26, 50].

As a result, for more than a century of the nation's early history, the mechanics of electing representatives to the House were left up to individual states [26]. Some states chose to select their Congressmen via a 'general ticket' or 'at-large' process, where voters cast as many votes as there were seats alloted to their state. Others deployed the single-member, geographically-bounded districting system we are familiar with from SECTION 1.1.

With general ticket statewide elections, there were few dials that could be turned by party machines to bend the rules to their advantage. However, the same was not true of single-member district elections. Now enshrined in legend, the 1812 state Senate redistricting of Massachusetts laid bare the sort of tactics politicians and political parties could utilise to improve their respective electoral outcomes. After the reapportionment of 1811, mapmakers in the state constructed a set of districts they hoped would enhance the number of seats they acquired [26]. One such district was so contorted that the local Boston Gazette declared it a 'Gerry-mander'; a portmanteau of the governor at the time, Elbridge Gerry, and the salamander shape of the district [43] (see FIGURE 1.2). As we will see in SECTION 1.4, this term has become ubiquitous in the modern context to describe a redistricting map with partisan or racial bias.

For most of the next century and a half, it was the prerogative of each state to determine the means by which it elected its members to the House [26]. The Apportionment Act of 1842 technically mandated the use of single-member districts throughout the union but at-large elections persisted in some cases until 1968 [20, pg. 25][26, 61]. It was not until the landmark legislative acts and judicial rulings of the 1960's Civil Rights Movement that the districting regime of today became the law of the land.

What has come to be known as the Reapportionment Revolution [20] began with a decision of the United States Supreme Court. In *Baker v. Carr* (1962) [11], the Court concluded that matters of state legislative reapportionment were justiciable and that malapportionment violated the voting rights of the citizenry. With *Wesberry v. Sanders* (1964) [9], this concept was extended to House districts and effectively outlawed population differences between districts [20, 26]. Finally, in 1967, Congress passed a statute banning at-large elections and requiring single-member districts [3, 26].

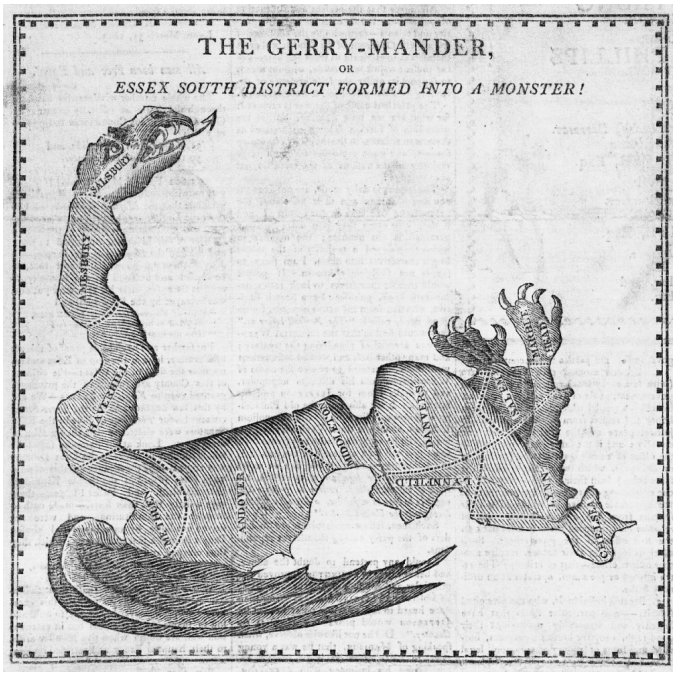


FIGURE 1.2: A cartoonist's depiction of the original gerrymander as published in the Boston Gazette on 2nd April, 1813 [10].

The last major factor contributing to the redistricting picture seen today can be traced back to the Voting Rights Act of 1965. This landmark legislation set out to guarantee enforcement of the 15th Amendment, which ensures that a citizen's right to vote is not denied on account of 'race, color, or previous condition of servitude' [2, 26, 68]. Section 2 of the act in particular establishes that redistricting plans cannot deny or abridge the voting rights of minority groups [68]. Further interpretation of the law in *Thornburg v. Gingles* (1986) [12] and *Bartlett v. Strickland* (2009) [38], established tests to identify and invalidate racially gerrymandered districts [68]. As a result, it is feasible to file suit to have a redistricting plan redrawn if it can be established that the map constitutes a racial gerrymander. That is, a district plan which specifically seeks to deny minority groups representation in the House.

In summary, modern congressional districts must adhere to the following rules and conditions:

1. Each state is geographically partitioned into single-member districts, subject to the number of seats allocated to that state by the Census.

2. Districts must be geographically contiguous.[3]

3. Each district must encompass an equal population of persons, with that number again defined by the Census.

4. Districts cannot be drawn to dilute or obstruct the voting power of minority populations.

3 Many states also have rules about the *compactness* of districts. This is discussed further in CHAPTER 5

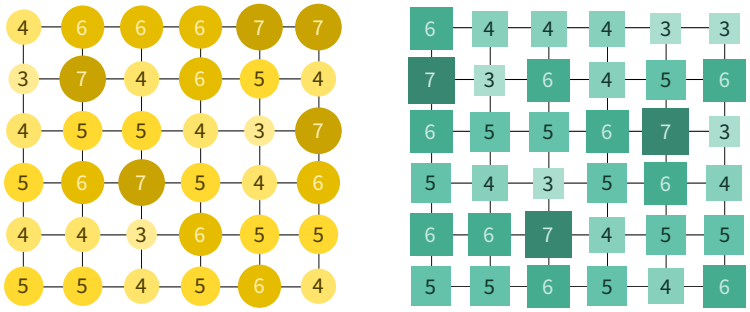
This brief and decidedly incomplete history serves to illustrate that both the law and politics of redistricting in the United States are a constantly moving target. This mercurial reality persists to this day, especially in the context of a recent flurry of Supreme Court cases surrounding partisan gerrymandering, which appear to have ended the judicial discussion for now [6]. It is this phenomenon of partisan gerrymandering, defined in the next section, which will form the nucleus of the rest of this work.

1.4 PARTISAN GERRYMANDERING

Though the 'original' gerrymander described in SECTION 1.3 was not entirely successful in its goal of maximising House seats for Gerry's party, the idea has persisted to this day [26]. The core reason for this is that specifically tailored districts can produce *partisan bias* [20]. To describe the mechanisms by which electoral benefits can be extracted from cleverly devised maps, we offer an example.

Consider an imaginary state, formed of a simple grid, with 36 discrete Voting Districts (VTDs). These VTDs form the atomic components used to 'build' the districts of the state. Since this state is a grid, every VTD neighbors at least two others (corners) and some neighbor up to four (centre). Within each VTD, there are 10 voters, for a total of 360 in the entirety of the state. This state sends four representatives to the House, which means each district is made up of nine VTDs and hence 90 voters.

Within our state there are two parties, each vying for control of these four House districts: the Circle party and the Square party. The vote shares each party will garner in the upcoming election can be seen in FIGURE 1.3. In this figure, each VTD is represented by a node and the edges connecting nodes represent geographic borders. That is, if two nodes are connected by an edge, that means they share a physical border. Each node is annotated with a number which describes how many votes that party will receive in that VTD. Voters can only vote for Circles or Squares, such that if the Circles net 6 votes in a VTD, the Squares get the other 4.



(A) (B)

FIGURE 1.3: Diagrammatic representation of our example 'state' with 36 Voting Districts (VTDs) given as nodes and with geographic boundaries represented by edges. Each number label represents the number of votes the party (Circles in A or Squares in B) receives out of the 10 available voters in each VTD. The size and shade of each node is proportional to the number of votes that party receives in that VTD.

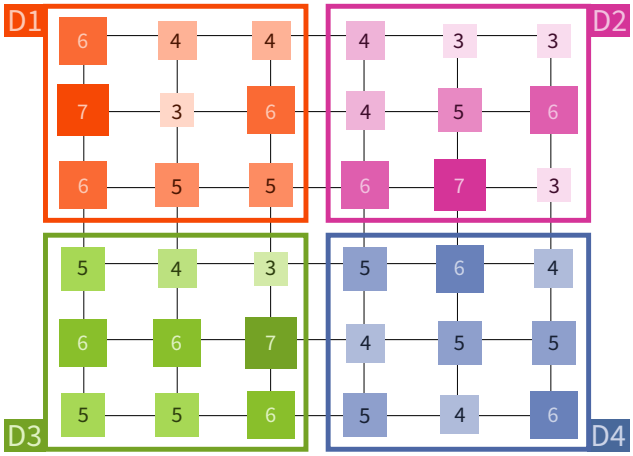


FIGURE 1.4: An example of 'fair' and proportional district plan, with each party winning two of the four districts. Districts 1 (orange) and 3 (green) are won by the Squares party with 46 of 90 and 47 of 90 votes respectively. Districts 2 (pink) and 4 (blue) are won by the Circles party with 49 of 90 and 46 of 90 votes respectively.

In this state, both parties are quite competitive. Statewide, the Circles net a total of 182 (50.6%) votes while the Squares get 178 (49.4%). If the state were districted in a fair and proportional fashion, this highly competitive environment would result in each party winning two districts. This scenario is depicted in FIGURE 1.4, from the perspective of the Squares. However, in our imaginary state, this district plan is not the one in place for the upcoming election.

The last time the districts were drawn, the Square party was in control of the state legislature and hence got to draw the map, as shown in FIGURE 1.5. It leveraged this legislative power to draw a map far more favourable to Squares. Specifically, two commonly used tactics were deployed: so-called 'packing' and 'cracking'. Both techniques seek to maximise the power of the mapmaker's votes while also minimising the impact of the opposition's votes.

With packing, a mapmaker identifies VTDS that favour her opponent and clusters them together. We see in FIGURE 1.5 that the author of the Square's map leveraged this technique for district 2. By doing so, she sacrificed the district, but also ensured that many lot of her opponents votes were wasted. Since U.S. elections are decided by whomever accumulates more than 50% of the total vote, winning with 70% of the vote means almost 20% of the votes cast for the winning party are in effect wasted. This concept of wasted votes will arise again when we consider the primary results of this work in CHAPTER 4.

With cracking, the Square's mapmaker took the remainder of the Circles voters and more or less evenly dispersed them in the remaining three districts. Since many Circle votes were wasted in the sacrificial district 2, she could quite easily encircle VTDS in the remaining three such that the Circles never surpassed 50% of the vote. By utilising the two arithmetically trivial techniques of packing and cracking, the Squares produced a set of geographically contiguous, relatively compact districts. Moreover, this map allowed them to win a majority (3 of 4) of the House seats with a minority of the vote (49.4%).

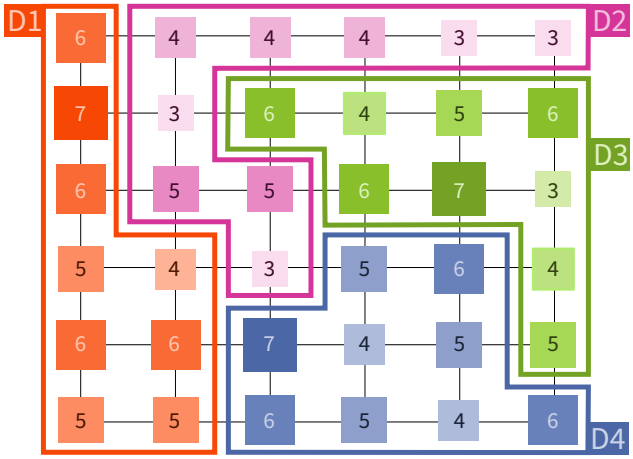


FIGURE 1.5: An example of gerrymandered districting, where the Squares party wins 3 of 4 districts even though statewide they win slightly less than a majority (49.4% or 178 of 360 voters). District 2 *packs* together VTDs that are overwhelming won by the Circles (low Squares numbers imply high Circles numbers), which the Circles win with 56 of 90 votes. The other three districts *crack* the Circles voters such that they do not achieve a majority in any of them. The Squares win district 1 with 50 of 90, district 3 with 46 of 90 and district 4 with 48 of 90.

This simple example is made of only 36 VTDs, while real states have thousands. This means that there are millions, billions, or more ways of drawing the legal district assignments of real states. To illustrate, consider a state with 2,400 VTDs that is awarded six congressional districts by the Census. If we abandoned the aforementioned legal constraints, like geographic contiguity and population balance, this means $2400^6 \approx 1.9 \cdot 10^{28}$ possible permutations to district the state. Enforcing the legal requirements of redistricting would clearly reduce this number by quite a few orders of magnitude, but the scale remains immense.

With this context in mind, how might one prove that a suspected partisan gerrymander is not merely a coincidence, drawn from one of the millions of possible plans? One solution might entail isolating the unbiased distribution of all the legal plans which could possibly be drawn in the given state. If the allegedly gerrymandered plan shows up in this unbiased distribution with high probability, we might conclude it is a manifestation of the political geography of the state and not actually unfair. Conversely, if it falls in the tails of our distribution, we would have a convincing indicator of bias.

For the rest of this work, we will focus on using a popular mathematical technique called Markov chain Monte Carlo (MCMC) to sample uniformly at random from this unbiased distribution of legal districting plans. In CHAPTER 2, we will develop the theory behind MCMC and examine how it can be operationalised in the abstract. We will then, in CHAPTER 3, turn these lessons on the redistricting problem and derive a framework that allows us to use MCMC to generate districting plans. Looking to CHAPTER 4, we will present and use the distribution of plans we generated to analyse a partisan gerrymander. At the end, in CHAPTER 5 we will discuss possible enhancements and additional factors not considered in this work.

2

MARKOV CHAIN MONTE CARLO

In the previous chapter, we laid out the political and legal foundations of redistricting in the United States. We introduced the idea of partisan gerrymandering and suggested that mathematics may offer a means of ferreting out districting plans possessing such biases. In this chapter, we will briefly set aside the redistricting problem and instead provide the formal underpinnings of Markov chain Monte Carlo (MCMC), a class of computational methods used for sampling from a probability distribution.

To fully explain MCMC, we will start by developing the key concepts of the special stochastic process of Markov chains, along with some essential properties. We will then see how these concepts can be extended by the use of Monte Carlo methods to obtain numerical results from problems which are analytically intractable. Finally, we will investigate the exceedingly popular Metropolis-Hastings algorithm—an MCMC method—which will become our key tool for tackling redistricting problems.

Since the contents of this chapter are about establishing a baseline of understanding for the reader, many of the theorems, definitions, and results are adapted from notable works on the subject of Markov chains. Given that A. A. Markov began his study in 1907 of the topic which bears his name, many of these results are well-known and well-founded [29]. As such, we take our cues, inspiration, and critical definitions and theorems primarily from the works of Grimmett and Stirzaker [28] and Serfozo [55]. Supplementary concepts and descriptions are drawn from the recent tutorial paper by DeFord [21]. Chapter one from the book by Norris [49] offers occasional insight, as does the introductory chapter of Levin, Peres and Wilmer [40] and the Markov chain sections from the work of Grinstead and Snell [29].

2.1 MARKOV CHAINS

At the most fundamental level, a *Markov process* describes a stochastic process which is memoryless [49]. By memoryless, we mean that the probability of each event depends only upon the state of the previous event. This is in stark contrast to generalised stochastic probability theory which considers that all past outcomes might have influence on the next outcome [29].

Though Markov processes can be defined in both discrete and continuous time, we will consider only the discrete case in this work. Drawing from the foundational definitions given by all the aforementioned authors, a discrete Markov process, made of a countable or finite set of possible states, it is commonly referred to as a *Markov chain* and defined here.

Definition 2.1 (Markov Chain)**.** Consider a sequence of random variables,

$$X = (X_0, X_1, \dots, X_n) \text{ for } n \in \mathbb{N}.$$

This sequence forms a *Markov chain*, for the countable or finite set of possible states $S = \{s_0, s_1, \dots, s_n\}$, if

$$\begin{aligned} \Pr(X_n = s_n \mid X_0 = s_0, X_1 = s_1, \dots, X_{n-1} = s_{n-1}) \\ = \Pr(X_n = s_n \mid X_{n-1} = s_{n-1}). \end{aligned} \tag{2.1}$$

In other words, the probability of X_n taking on state s_n given all previously held states is the same as the probability X_n taking state s_n given *only* the previously held state.

2.1.1 *Transition Probabilities*

We can extend our understanding of Markov chains by imagining each successive state change as a step or walk. This is commonly described by the anecdote of a frog jumping between a set of lily pads [29, 40]. From an initial state (starting pad), the frog then jumps from one pad to the next with a *transition probability* defined between 0 and 1 inclusively for each possible state change. Such a scenario is depicted in FIGURE 2.1, with the lily pads drawn as green circles and the transition probabilities annotated with arrows between pads.

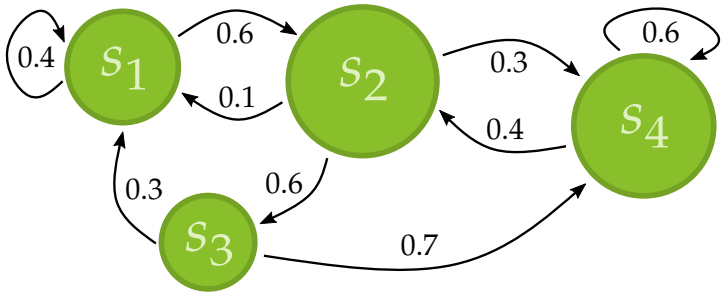


FIGURE 2.1: A schematic representation of four possible states $\{s_1, \dots, s_4\}$, in this case lily pads, between which the frog might jump. Green circles represent states and arrows represent the transitions between states. The numbers annotating each arrow correspond to the transition probability from source state to destination state as the arrow points. Note that states s_1 and s_4 have probabilities of 'transition' onto themselves, meaning the frog stays on that pad for that step in the chain.

When considering these transition probabilities, we must emphasise that this work will only address Markov chains which are stationary or time-homogeneous. This means that transition probabilities remain fixed across time steps. Though by no means a requirement,[1] this holds for the scenarios we will explore throughout the coming pages. The definition of time-homogeneity from Serfozo [55], which we give below, is straightforward and as expected.

Definition 2.2 (Time-homogenous Markov Chain)**.** A stationary or time-homogeneous Markov chain is one where:

$$\mathbb{P}\left(X_{n+1} = s_j \mid X_n = s_i\right) = \mathbb{P}\left(X_n = s_j \mid X_{n-1} = s_i\right) \tag{2.2}$$

for any time n in the sequence of the chain. That is, transition probabilities are independent of time.

1 see **Henry1971** for example

Knowing now that all chains from here on out will be time-homogeneous, let us look to further develop our understanding of transition probabilities. We note that the representation given by FIGURE 2.1 provides visual clarity, but it is not particularly concise. As Grinstead and Snell [29] and Serfozo [55] demonstrate, it is common to encapsulate the transition probabilities in a *transition matrix*.

Definition 2.3 (Transition Matrix)**.** Define the transition matrix P to describe transition probabilities for the Markov chain X. For each i^{th} row and j^{th} column, the elements p_{ij} of matrix P describe the probability of transition from state s_i to state s_j. That is,

$$p_{ij} = \mathbb{P}\left(X_{n+1} = s_j \mid X_n = s_i\right), \tag{2.3}$$

where X_n and X_{n+1} represent sequential time steps of the Markov chain.

Given this definition of the transition matrix, we can reform the annotated probabilities from FIGURE 2.1 as follows:

$$P = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{bmatrix} 0.4 & 0.6 & 0 & 0 \\ 0.1 & 0 & 0.6 & 0.3 \\ 0.3 & 0 & 0 & 0.7 \\ 0 & 0.4 & 0 & 0.6 \end{bmatrix} \end{array}$$

Two key consequences can be drawn from observing this formation of the transition probabilities. First, summing the probabilities of the rows always equates to 1, and hence each row forms a vector representing the transition probability *distribution* of that state [40, 49]. Second, it is clear that this square matrix is not symmetrical, indicating that transitions between two states need not have the same probability in each direction. When reading the matrix, we note that the row index (e.g. s_3 for row 3) describes the state we are 'departing' and the column index (e.g. s_2 for column 2) references the state upon which we 'arrive'.

Describing transitions between states through a matrix offers algebraic advantages when considering incremental time steps. Specifically, Grinstead and Snell [29] offer an important theorem, and we adapt the work from Maltby, Pakornrat and Jackson [42] as proof.

Theorem 2.4 (Time-homogenous Transition)**.** *Let P be the transition matrix of a time-homogeneous Markov chain. The ij^{th} entry p^n_{ij} of the matrix P^n gives the probability that the Markov chain, starting in state s_i, will be in state s_j after n steps.*

Proof. Denote a new matrix $M = P^2$, with P^2 indicating matrix multiplication of P with itself. This gives the elements of M:

$$\begin{aligned} M_{ij} &= \sum_{k=1} p_{ik} p_{kj} \\ &= \sum_{k=1} \mathbb{P}\left(X_{n+1} = s_k \mid X_n = s_i\right) \mathbb{P}\left(X_{n+2} = s_j \mid X_{n+1} = s_k\right) \end{aligned}$$

Simplifying, we see this results in,

$$= \mathbb{P}\left(X_{n+2} = s_j \mid X_n = s_i\right),$$

as required. From this result, M can be extended to include P^n time steps. ■

We have seen that using the transition matrix, we can describe the evolution of the chain probabilistically. As DeFord [21] succinctly states, a row of the matrix describes the probabilities with which we will be in the next state. We labour this point because it is critical to the use of Markov chains in conjunction with Monte Carlo methods. Namely, we can leverage this probabilistic nature to answer questions about specific states after hundreds or thousands of time steps [21].

2.2 PROPERTIES OF MARKOV CHAINS

In this section, we will adapt a series of properties from Grimmett and Stirzaker [28] and Serfozo [55] which can be used to describe Markov chains. In developing these properties, we will style example Markov chains as directed graphs, with nodes representing states and edges representing the transition probabilities between states (as in FIGURE 2.1). In this sense, we consider the chain as a random walk amongst nodes. While not inherently required from the theoretical standpoint, this representation is intuitive and common to our purposes.

This section is concerned namely with certain categories of states. The nature of transition between states can inform us as to how the Markov chain will 'evolve'. We will hone in on such relationships, which allow us to classify states and the Markov chains bearing those states. The definitions, theorems and nomenclature of the rest of this section are drawn in part or in whole from Serfozo [55, pgs. 16-26] and Grimmett and Stirzaker [28, pgs. 213-227], with occasional insight from DeFord [21, pg. 12-13].

2.2.1 *Recurrence*

For a state space $S = \{s_0, s_1, \dots, s_n\}$, we might find value in knowing the amount of time it takes to reach or return to a specific state. Formally, we define the *hitting time* below, as per Serfozo [55, pg. 19].

Definition 2.5 (Hitting Time). A hitting time of a subset $A \subset S$ for a Markov chain X is defined by

$$\tau = \min\{n \geq 1 : s_n \in A\}. \tag{2.4}$$

In other words, τ gives us the minimum number of steps we must iterate through the Markov chain in order to reach state s_n. This definition is useful for considering states to which the chain may or not return. Specifically, one might condition the iteration of stepping through a Markov chain on the hitting time to a particular state or collection of states. This brings us to the concept of recurrence and transience.

Consider the hitting time,

$$\tau_j = \min\left\{n \geq 1 : X_n = s_j\right\}, \tag{2.5}$$

to be the time to reach state s_j for Markov chain X. That is, τ_j follows from DEF. 2.5 with $A = \left\{s_j\right\}$. We can then use an expanded notation taken from Serfozo [55] to describe the transition probability associated with this hitting time. Namely, we use $P_i\left\{X_n = s_j\right\}$ to describe the p_{ij}^n element of the transition matrix. Hence, substituting in τ, we can then describe the probability associated with this hitting time as,

$$f_{ij}^n = P_i\left\{\tau_j = n\right\}, \quad n \geq 1. \tag{2.6}$$

That is, the probability that a chain starting in state s_i reaches state s_j for the 'first time' on the nth step. We might then subsequently define the probability that the chain *ever* reaches state s_j from state s_i as:

$$f_{ij} = P_i\left\{\tau_j < \infty\right\} = \sum_{n=1}^{\infty} f_{ij}^n, \tag{2.7}$$

since each f_{ij}^n is a disjoint probability. This gives us our first category of state.

Definition 2.6 (Recurrence)**.** A state s_i is called *recurrent* if $f_{ii} = 1$; that is the chain *returns* to s_i with probability 1. The state is *transient* if it is not recurrent. A recurrent state s_i is further classified as *positive recurrent* if the expectation $\mathbb{E}_i[\tau_i] < \infty$, meaning the chain returns to state s_i in finite time.

Furthermore, a set of states C is said to be recurrent if all its states are recurrent and similarly for transient. If the set $C = S$ then the recurrence (or transience) definition extends to the entire chain. FIGURE 2.2 gives examples of transient and recurrent states.

The final component to consider is the expectation of recurrence for a state. We do this using the definition from Grimmett and Stirzaker [28, pg. 222] directly:

Definition 2.7 (Mean Recurrence)**.** The *mean recurrence time* μ_i of a state s_i is defined as

$$\mu_i = \mathbb{E}\left(\tau_i \mid X_0 = s_i\right) = \begin{cases} \sum_n n f_{ii} & \text{if } s_i \text{ is recurrent,} \\ \infty & \text{if } s_i \text{ is transient} \end{cases} \tag{2.8}$$

We will see in SECTION 2.3 how this definition can be used to describe distributions associated with the Markov chain in question.

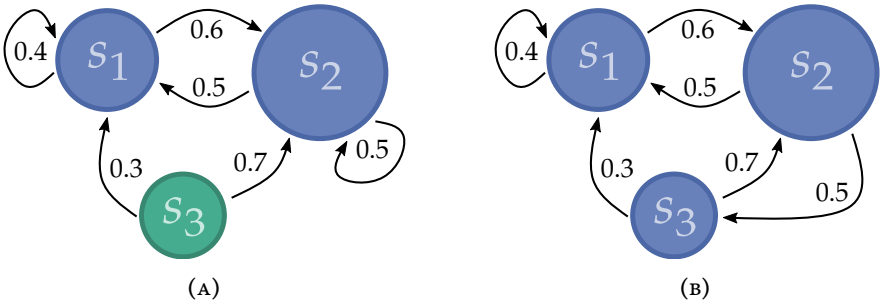


FIGURE 2.2: An pair of diagrams showing transient and recurrent scenarios. FIGURE A shows two positive recurrent states in blue, s_1 & s_2, and one transient state s_3 in teal. FIGURE B shows a Markov chain made of positive recurrent states, making it a positive recurrent Markov chain.

2.2.2 *Irreducibility*

One of the ways in which we can classify the evolution of a Markov chain is by the *accessibility* between arbitrary states s_i and s_j. We say that s_i is accessible from s_j, written $s_i \to s_j$, if $p_{ij}^n > 0$ for $n \geq 1$. That is to say that state s_i can transition to state s_j, if not directly then via other states, in at least one step. Furthermore, we say the states *communicate*, denoted $s_i \leftrightarrow s_j$, if $s_i \to s_j$ and $s_j \to s_i$.

We can define *communication classes* of states by partitioning the state space S into disjoint subsets, as described by FIGURE 2.3. Considering one such set $C \subset S$, we say that C is an *irreducible* set if $s_i \leftrightarrow s_j$ for any $s_i, s_j \in C$. We say the Markov chain is irreducible if this holds for the entire state space S.

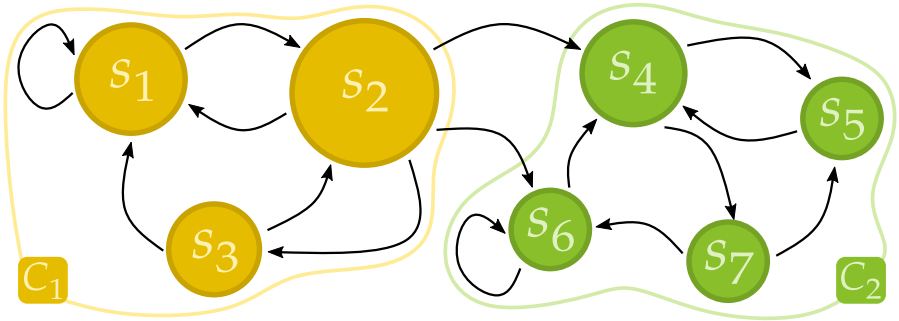


FIGURE 2.3: A Markov chain with two communicating classes, $C_1 = \{s_1, s_2, s_3\}$ and $C_2 = \{s_4, s_5, s_6, s_7\}$. Within each class, all states can be reached from all other states, making each class irreducible. However, the Markov chain as a whole is not irreducible since no state in C_1 can be reached from C_2. Transition probabilities have not been annotated in order to improve clarity; they are not needed to demonstrate reducibility.

2.2.3 *Periodicity*

Another important factor for categorising chains is to consider the time it takes to 'revisit' a state. As DeFord [21, pg. 13] and Serfozo [55, pg. 23] convey, the *period* d_i for a state s_i is the Greatest Common Divisor (GCD) of all n that satisfy $p_{ii}^n > 0$. A state is called *aperiodic* if $d_i = 1$ and otherwise it is called *periodic* with period d_i.

Said differently, the period of a state is the GCD of steps n taken on all possible paths or routes that produce recurrence for the state in question. For example, FIGURE 2.4B displays a set of states with two possible return routes, one which takes three steps and another which takes six steps. Hence, $d_i = 3$ and the states, and chain, have period three. FIGURE 2.5 shows a chain which is aperiodic.

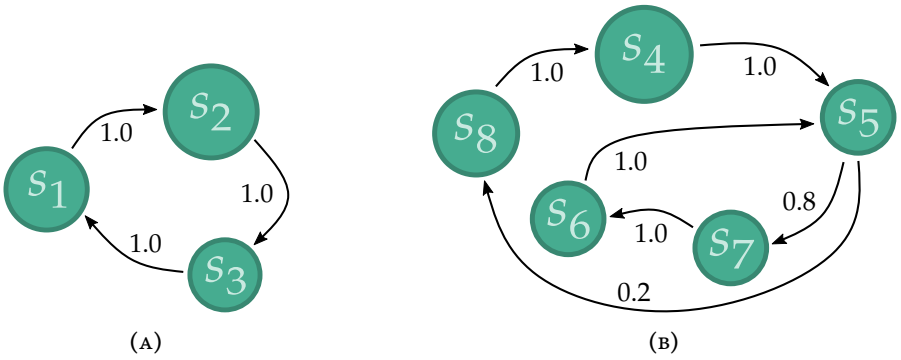


FIGURE 2.4: Two examples of periodic Markov chains. The 'triangle' (FIGURE A) is perhaps the simplest example, where $n = 3$ for p_{ii}^n for all states s_i, and hence the $d_i = 3$. However, more complex Markov chains can also be periodic. FIGURE B shows five states with $n = 3, 6$ for p_{ii}^n for all states s_i. Thus $d_i = 3$ once again. Note that in both examples, all states share the same period and therefore we would say each *Markov chain* is periodic with a period of 3.

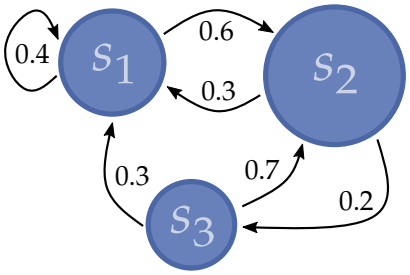


FIGURE 2.5: An example of an aperiodic Markov chain. Looking state by state, s_1 has $n = 2, 3$ for p_{ii}^n. Similarly, s_2 and s_3 also have $n = 2, 3$. Therefore $d_i = 1$ for all states, indicating the entire Markov chain is aperiodic.

2.2.4 *Ergodicity*

Further consideration of the aforementioned properties reveals that they are linked. Specifically, Grimmett and Stirzaker [28] and Serfozo [55] give us a theorem to tie them all together:

Theorem 2.8 (Finite & Irreducible Chain). *Suppose a* finite *set of states C is irreducible; that is, any state can be reached from any other state. Then all states in C are positive recurrent and hence so is C. Furthermore, all states in C have the same period.*

Proof. See Serfozo [55, pg. 24], Grimmett and Stirzaker [28, pg. 225] and Weber [67, pg. 20]

This scenario, where a set of states is both positive recurrent and all states share the same period can be seen again in FIGURE 2.4 and leads naturally to another definition:

Definition 2.9 (Ergodicity). A set of states C which is positive recurrent and aperiodic is called *ergodic*. By extension, a Markov chain X with state space S is called an ergodic Markov chain if S is ergodic.

Note that our first Markov chain described in FIGURE 2.1 is ergodic. The ergodic property for a Markov chain is of critical importance to us. Ergodic Markov chains have the further property that they possess a unique stationary distribution which forms a probability measure on the state space S. These stationary distributions will be the focus of the upcoming SECTION 2.3 and are essential to the rest of this work.

2.3 STATIONARY DISTRIBUTIONS

With this section, we will develop the nature of stationary distributions for Markov chains. As a result, we will round out the basic understanding and tools of Markov chains we require to describe the Metropolis–Hastings algorithm; the MCMC method we will use extensively. As with SECTION 2.2, Serfozo [55, pgs. 35-41] and Grimmett and Stirzaker [28, pgs. 227-236] are adapted in part or in whole for the definitions, theorems and proofs that follow.

We start by defining the concept of a stationary distribution as given by Grimmett and Stirzaker [28, pg. 227]. Note that unless stated otherwise, all vectors are considered row vectors.

Definition 2.10 (Stationary Distribution). The vector π is called a *stationary distribution* of a Markov chain if π has entries $(\pi_j : s_j \in S)$ such that:

1. $\pi_j \geq 0$ for all j, and $\sum_j \pi_j = 1$,
2. $\pi = \pi P$, which is to say that $\pi_j = \sum_i \pi_i p_{ij}$ for all j.

In other words, π forms a probability measure which is fixed, even as the transition matrix P evolves to higher powers. To see this, we need only examine iterating through the chain by incrementing the power of the transition matrix. Consider,

$$\pi P^2 = (\pi P)P = \pi P = \pi.$$

Extending this from P^2 to P^n, we can clearly see

$$\pi P^n = \pi \quad \text{for all } n \geq 0.$$

Armed with this definition of a stationary distribution, we come to a theorem which is central to the Metropolis–Hastings algorithm. Recall from SECTION 2.2.2, that an irreducible Markov chain is one in which all states communicate. Then remember, from SECTION 2.2.1, that a recurrent state is one to which the chain returns with probability equal to 1. With these properties in mind, we give the ergodic theorem as supplied by Grimmett and Stirzaker [28]. Note that the ergodic property of a Markov chain, meaning that it is aperiodic and irreducible, differs from the category of Markov chain given in this theorem.

Theorem 2.11 (Ergodic Theorem). *An irreducible Markov chain X has a stationary distribution π if and only if all of its states are positive recurrent. In this case, the stationary distribution π is unique and has the form:*

$$\pi_i = \mu_i^{-1}, \tag{2.9}$$

for each $s_i \in S$, where μ_i is the mean recurrence time of i (see DEF. 2.7*).*

Proof. See Grimmett and Stirzaker [28, pgs. 228-230]

Reflecting on this theorem, we see that it gives us a powerful result: chains which are irreducible and positive recurrent possess a *unique* stationary distribution. Furthermore, this stationary distribution is populated by the average time spent in each state (μ_i^{-1}). If we extend this theorem a little further to include chains which are aperiodic, we arrive at perhaps the central conclusion we require for Metropolis–Hastings. Once again, we are guided by Grimmett and Stirzaker [28].

Theorem 2.12. *For an irreducible and aperiodic (i.e. ergodic) Markov chain, we have that*

$$p_{ij}^n \to \frac{1}{\mu_j} \quad \text{as } n \to \infty, \quad \text{for all } i \text{ and } j. \tag{2.10}$$

If the chain is positive recurrent, then $p_{ij}^n \to \pi_j = \mu_j^{-1}$, where π is the unique stationary distribution of THEOREM 2.11.

Proof. See Grimmett and Stirzaker [28, pgs. 233-235]

We see in THEOREM 2.12 an enhancement of THEOREM 2.11, which states that P in fact encodes the unique stationary distribution of the Markov chain, provided the chain is irreducible, aperiodic, and positive recurrent. Moreover, as we noted in THEOREM 2.8, when the state space is finite, irreducibility requires positive recurrence. Since we will only consider finite state spaces in this work, we can remain confident that THEOREM 2.12 is satisfied whenever a chain is aperiodic and irreducible.

2.3.1 *Detailed balance*

Before concluding this section, we consider one further property of some Markov chains which is of interest: detailed balance. To demonstrate this property, we start with reversibility, which considers how the chain evolves if we run it in reverse order. We give the definition from Grimmett and Stirzaker [28, pg. 237].

Definition 2.13 (Reversed Chain)**.** Suppose $\{X_n : 0 \leq n \leq N\}$ is an irreducible and positive recurrent Markov chain with transition matrix P and stationary distribution π. The reversed chain Y_n is defined as

$$Y_n = X_{N-n} \quad \text{for } 0 \leq n \leq N. \tag{2.11}$$

The proof of Grimmett and Stirzaker [28, pg. 237] gives us that Y is indeed a Markov chain and goes further to define detailed balance.

Definition 2.14 (Detailed balance)**.** The chain X is called reversible if the transition probabilities are the same, such that

$$\pi_i p_{ij} = \pi_j p_{ji} \quad \text{for all } i, j. \tag{2.12}$$

This relation is called the *detailed balance* equation.

We put attention on reversibility, and specifically detailed balance, because of the following theorem taken directly from Grimmett and Stirzaker [28, pg. 238].

Theorem 2.15. *Let P be the transition matrix of an irreducible Markov chain X, and suppose that there exists a distribution π such that $\pi_i p_{ij} = \pi_j p_{ji}$ for all $i, j \in S$. Then π is a stationary distribution of the chain.*

Proof. See Grimmett and Stirzaker [28, pgs. 238].

Looking ahead to SECTION 2.5, we will see that detailed balance is an essential component of the Metropolis–Hastings algorithm. In fact, satisfying detailed balance is critical to the algorithm allowing us to sample from a target distribution.

Throughout this chapter, we have examined how specific traits of Markov chains can yield discrete behaviour over the 'lifetime' of the chain. This manifests most consequentially in the case of stationary distributions and particularly in the irreducible and positive recurrent case where this distribution is unique. In the coming sections, we will see how we might derive an ergodic Markov chain to inform us about a target distribution when analytical definition of that distribution is intractable.

2.4 MARKOV CHAIN MONTE CARLO

Up until this point, we have considered the first half of the term 'Markov chain Monte Carlo', but made no mention of the second half of the technique. We will remedy this now by introducing Monte Carlo methods and how we might combine them with Markov chains. This will set the stage for arguably the most famous MCMC method, Metropolis–Hastings, addressed in SECTION 2.5. As in previous sections, foundational theory is adapted in part or in whole from Grimmett and Stirzaker [28] and Serfozo [55].

Monte Carlo methods are frequently in the arsenal of anyone looking to generate random samples or processes by leveraging the power of modern computing. This means that they crop up across fields as disparate as operations research, physics, finance and computational statistics [39]. The basic process

is to repeat an experiment or random sampling many, many times to obtain numerical results which approximate values or distributions of interest.

For our purposes, we can immediately see the value of Monte Carlo methods in the redistricting context by remembering the enormity of the state space. As we described briefly in CHAPTER 1, the number of permutations available to district a state can easily exceed 10^{20}. Deriving a probability distribution for a state space of this order of magnitude is clearly unfeasible. Ergo, we turn to sampling techniques.

Formally, per Serfozo [55], we routinely deploy Monte Carlo methods in scenarios when we wish to compute expectations of the form:

$$\mu = \sum_{i \in S} g(i)\pi_i, \tag{2.13}$$

where π is a probability measure and $g : S \to \mathbb{R}$. Per Grimmett and Stirzaker [28], it is common in MCMC methods to derive an ergodic Markov chain (see DEF. 2.9) which has:

1. π as its unique stationary distribution, and
2. transition probabilities of a simple form.

With such a chain, $X = \{X_n : n \geq 0\}$, we have that

$$\hat{\mu}_n = \frac{1}{n} \sum_{m=1}^{n} g(X_m) \to \sum_i g(i)\pi_i, \tag{2.14}$$

as $n \to \infty$.

That is to say, $\hat{\mu}_n$ forms a consistent estimator[2] of μ. Stepping back, we note that simulating the Markov chain, using a suitable Monte Carlo method with large n, produces an estimation of the expectation we sought initially (EQ. 2.13) but could not compute. This is result of Markov chains is one which we deploy with the Metropolis–Hastings algorithm.

2.5 THE METROPOLIS–HASTINGS ALGORITHM

In the preceding sections, we have established the components we need to assemble our most valuable tool: the Metropolis–Hastings MCMC method. This algorithm seeks to resolve a common problem: we have a target distribution π and we wish to produce samples from it. As we saw in SECTION 2.4, sampling can provide a much easier means—or indeed sometimes the only means—of gaining insights into properties of the target distribution when that distribution is not conducive to analytical inspection.

With the Metropolis–Hastings method, we can sample proportionally to π by simulating an ergodic Markov chain with π as its stationary distribution. This is valuable for complex or multi-dimensional distributions or exceedingly large state spaces where computation of the normalising constant is prohibitively expensive. As we alluded to, we will see in CHAPTER 3 that this is exactly the case in the redistricting problem.

2 Per Serfozo [55, pg. 68]: 'A statistic $\hat{\theta}_n$ that is a function of observed values $X_1, \ldots, X_n$ is a *consistent estimator* of a parameter θ if $\hat{\theta}_n \to \theta$ asymptotically as $n \to \infty$'

Though it is not immediately obvious that this process would be straightforward, the Metropolis–Hastings algorithm gives us a surprisingly simple means of achieving our sampling goal. In this section, we will first present the theory behind the algorithm and then its procedure. Since the formulation given by Grimmett and Stirzaker [28, pg. 293] is concise and well-formed, we will restate it, with additional context provided by Robert [54].

2.5.1 *Theory*

Paraphrasing from Grimmett and Stirzaker [28, pg. 293], we seek an ergodic Markov chain X on the state space S with transition matrix P and stationary distribution π, such that we might easily simulate X. In order to define the transition matrix P—with elements p_{ij} as before—of this chain, we split the transition probabilities into two pieces: proposal and acceptance. We define the proposal below using a matrix formulation, but it is equally common to do so with a conditional density (see Robert [54]). The following three definitions come directly from Grimmett and Stirzaker [28, pg. 293].

Definition 2.16. Let $H = \left(h_{ij} : s_i, s_j \in S\right)$ be a matrix called the *proposal matrix*, with elements for the random variable $Y \in S$ given by

$$h_{ij} = \mathbb{P}\left(Y = s_j \mid X_n = s_i\right). \tag{2.15}$$

Hence H gives us the probability of transitioning from state s_i to state s_j, but we stop short of declaring $P = H$. This is because, by combining H with our acceptance probability A and choosing A appropriately, we are able to achieve detailed balance (see DEF. 2.14) for P. That is $\pi_i p_{ij} = \pi_j p_{ji}$, which by THEOREM 2.15 means that π is a stationary distribution of X. This will become clear when we see how P is defined. First, we look to defining the acceptance probability.

Definition 2.17. Let $A = \left(a_{ij} : s_i, s_j \in S\right)$ be a matrix called the *acceptance matrix*, with elements satisfying $0 \leq a_{ij} \leq 1$. Given $Y = s_j$, we set

$$X_{n+1} = \begin{cases} s_j & \text{with probability } a_{ij}, \\ X_n & \text{with probability } 1 - a_{ij}. \end{cases} \tag{2.16}$$

This key factor here is that we sometimes do not accept the proposal. That is, with probability $1 - a_{ij}$, we remain at X_n instead of accepting the proposal given by H. Combining H and A gives us the transition matrix P for the chain X.

Definition 2.18. Given the proposal matrix H with elements h_{ij} and the acceptance matrix A with elements a_{ij}, define the transition matrix P with elements p_{ij} as:

$$p_{ij} = \begin{cases} h_{ij} a_{ij} & \text{if } i \neq j, \\ 1 - \sum_{k \neq i} h_{ik} a_{ik} & \text{if } i = j, \end{cases} \tag{2.17}$$

Now recall that if the transition probabilities p_{ij} satisfy detailed balance for the distribution π, then π is a stationary distribution of the Markov chain. Looking back to the detailed balanced equation:

$$\pi_i p_{ij} = \pi_j p_{ji},$$

we then substitute in our definition of p_{ij}, with $i \neq j$:

$$\pi_i h_{ij} a_{ij} = \pi_j h_{ji} a_{ji}.$$

Rewriting this as a ratio gives,

$$\frac{a_{ij}}{a_{ji}} = \frac{\pi_j h_{ji}}{\pi_i h_{ij}}. \tag{2.18}$$

Metropolis et al. [45] provides us the means of satisfying the above equality by choosing the acceptance rate given by:

$$a_{ij} = \min\left(1, \frac{\pi_j h_{ji}}{\pi_i h_{ij}}\right). \tag{2.19}$$

Hence, either,

$$a_{ij} = 1 \text{ and } a_{ji} = \frac{\pi_i h_{ij}}{\pi_j h_{ji}}, \quad \text{or} \quad a_{ji} = 1 \text{ and } a_{ij} = \frac{\pi_j h_{ji}}{\pi_i h_{ij}}.$$

In both cases, we see that EQ. 2.18 is satisfied and therefore so is detailed balance. With detailed balance satisfied, π is a stationary distribution of the Markov chain X. Furthermore, since we chose X such that it is ergodic, π is the *unique* stationary distribution of X (recall THEOREM 2.11) and hence sampling from the acceptance probability gives us a distribution proportional to π.

Considered differently, it is perhaps more intuitive to think about what sort of acceptance probability is produced for a given proposal. Effectively, the acceptance ratio measures the likelihood of the proposed state against the likelihood of the current state subject to π. In the scenario where the proposed state is more likely, the acceptance probability is 1. When reverse is true, the proposal is less likely to be accepted. As a result, samples from A will tend to frequently land in high-density regions of π and less often in low-density areas, thereby approximating π as required.

2.5.2 *Procedure*

Given this intuition, the three definitions above and the Metropolis ratio given by EQ. 2.19, we can give the Metropolis–Hastings algorithm in full, as follows.

0. First iteration only: initialise Markov chain X with initial state $X_0 = s_0$
1. Given current state $X_n = s_i$, generate proposal state s_j according to h_{ij}
2. Compute acceptance probability of proposed state from

$$a_{ij} = \min\left(1, \frac{\pi_j h_{ji}}{\pi_i h_{ij}}\right)$$

3. Sample a uniform random number $\beta \sim U[0,1]$
4. Update state of X_{n+1} subject to

$$X_{n+1} = \begin{cases} s_j & \text{if } \beta \leq a_{ij} \\ s_i & \text{otherwise} \end{cases}$$

5. Increment $n = n + 1$ and return to step 1

Though the process of acquiring a sample is relatively straightforward with this algorithm, determining when the sampling distribution has converged to the stationary distribution is less trivial. Though not the subject of this work, two manifestations of the practical use of the method are worthy of discussion.

The first is the *burn-in*, which is an arbitrary and empirical period describing how long it takes for the sampling distribution to begin approximating the desired distribution. This can be conceptualised as the number of iterations required for the chain to 'break out' of the neighbourhood of states local to the initial state. This is normally used to combat the practical reality that sometimes one starting state is better than another. It is not uncommon for the burn-in period to be completely removed from sample results.

The second practical component worth consideration is the *mixing time*. This term describes how long it takes, in terms of iterations, for samples to start approximating the stationary distribution. While over the long run, the sampled distribution will follow the stationary distribution, in practice, 'nearby' samples are correlated with one another. For a well parameterised chain, this autocorrelation of neighbouring samples decreases as $n \to \infty$ and can be managed.

In practice, the choice of proposal distribution, and its impact on the acceptance rate, does directly influence the mixing time of the chain. This means that a poorly chosen proposal distribution can result in slow mixing and vice versa. As one might imagine, faster mixing means quicker convergence to the desired stationary distribution. The implication being that with a chain which is mixing well, fewer iterations are required to acquire an adequate sample. Therefore, fewer compute resources are also required. Such considerations will be discussed in detail in CHAPTER 5, after we have seen Metropolis–Hastings in action.

With this chapter, we have witnessed a targeted look at certain types of discrete, time-homogeneous Markov chains. When properly formed, such chains can be used to gain insight into the processes the chains seek to model. We saw a blueprint for this with the introduction of the Metropolis–Hastings algorithm. In the next chapter, we look to codify the task of redistricting into a Markov chain, with a unique stationary distribution that describes the relationship between districts and voters. Upon this chain we will then deploy our MCMC algorithm to inform us about this distribution. From there, we will be in a position to once again centre our focus on scenarios of partisan gerrymandering.

3

SIMULATING REDISTRICTING

We gained an understanding of the political stakes of redistricting in chapter 1. In chapter 2 we garnered a comprehensive insight into the theory of mcmc. In light of each of these, we are now well positioned to tackle the gerrymandering problem in practice. We will complete this task over two chapters. In this first one, we will explain our methodology for using the Metropolis–Hastings algorithm to sample legal districting plans. In the second, we will present our resulting distribution and use it to analyse a suspected partisan gerrymander. This analysis will leverage a few of the tools available for identifying gerrymanders, and more such tools will be discussed in chapter 5.

This chapter is split into a few discrete pieces. We will begin by outlining the tools we used for this work in section 3.1 and then move on to section 3.2 where we introduce our synthetic dataset. With sections 3.3 to 3.7 we address our main methodology in full. This includes how we define our Markov chain, how we propose new states in this chain, how we define our acceptance probability and what our implementation of Metropolis–Hastings looks like. We will conclude with sections 3.8 and 3.9 by exploring parameterisation and the conditions we impose on our methodology to accommodate real-world concerns.

Before diving in, we first note that the redistricting problem does not fit as neatly into the Metropolis–Hastings framework as we might hope. In chapter 2, we described the theoretical concepts using an arbitrary chain and a nebulous target distribution. In this chapter, both those crucial pieces will represent tangible objects with real-world constraints. In the first instance, the Markov chain will be defined over a state space made up of possible districting plans, which will we describe in detail in section 3.4. Where the target distribution is concerned, we take a tiered approached. This means that our mcmc method produces an initial distribution which is then further processed to access the set of *legal* plans. This will be made clearer in section 3.6. However, before tackling such complications, we will set the scene by presenting our tools and dataset.

3.1 TOOLS

We begin reporting our methodology with an outline of the tools used to complete this work. One advantage of using the dataset we created (see section 3.2) is that all computations and simulations could be run on the author's personal computer. However, this is not to trivialise the task; simulation runtimes on this consumer-grade desktop machine ranged from 5 minute tests to 90 minute final runs on a 4.7ghz hexacore Central Processing Unit (cpu).

No high performance computing clusters or Graphics Processing Units (GPUS) were required or used for this work.

On the software side, a wide array of tools were utilised. The vast majority of the 1,000 lines of source code was written in `python` (v. 3.7.4) and can be found in a curated form in APPENDIX A and in full on the author's GitHub repository [31].[1] Where appropriate, selected excerpts from this source code will be presented as code listings. The ubiquitous and industry-standard utility packages of `pandas` [62] and `numpy` [64] were employed throughout this work. For plotting and graphics, we used `matplotlib` [35] in `python` and `tidyverse` [69] in R. To save pages, code for the R plots is not included in this document, but it can be found on the GitHub page. Most critically, the network analysis package `networkx`[30] was used as the predominant resource for structuring, manipulating and simulating the graphs to be discussed in the coming sections.

With the singular exception of a colour bar remapping function [19], all source code is the original work of the author. This includes adaptation and implementation of the Metropolis–Hastings algorithm for the redistricting problem space. It also includes all plots and graphics. Where inspiration or ideas for functions came from the work of others, citations will be supplied. All other functions and algorithms are the original work of the author.

3.2 DATASET

Let us now turn to the dataset we will use for the rest of this document. While the initial expectation might be to use shapefiles and VTD data from real U.S. states, we do not take this approach. There are two primary reasons for this. The first is one of scale. Computation of the many millions of iterations which would be required for real-world data was not feasible on our personal computer. One method for overcoming this constraint, called recombination, is presented in SECTION 5.2.1. The second is one of implementation. As we will also discuss in CHAPTER 5, numerous factors of the real-world—such as the Voting Rights Act (VRA), communities of interest, and compactness—severely complicate design and execution of the Markov chain.

Instead, we chose to start with a synthetic dataset so that we might produce a robust code base that could be scaled up to incorporate real-world datasets. In authoring this code, we encountered more than enough challenges. As a result, upgrading the code base to handle real data exceeded the time frame afforded to this project. However, we fully believe our synthetic dataset to be more than adequate for serving the purposes of this work.

With our dataset, we model a hypothetical state, call it Nodeland, which seeks to mimic the dynamics of a real state. For example, we model city versus rural population stratification with high densities in a few select places and low densities spread out everywhere else. The design and structure is once again a network graph, as with the examples presented in SECTION 1.4. However, we do extend that framework in a number of ways.

Nodeland is comprised of six congressional districts and a total of 60 Voting Districts (VTDS). These are discrete geographic population blocks which

1 https://github.com/HeadCase/dissertation

correspond to voting precincts. Recall that these VTDs are the building blocks of the districts and seek to emulate one of the common types of block used to draw real congressional districts. The six districts, and the specific VTDs assigned to them, make up the initial plan for our state. As we will see in greater detail in SECTION 3.3, this plan is just one of many possible plans that can district the state within the constraints of the law.

We represent the VTDs as nodes on a network graph. When speaking to the political geography of Nodeland, we will reference VTD; when we look to the more abstract nature of the Markov chain and modelling using a directed graph, we will reference nodes. In both cases, we are referring to the same discrete population blocks which make up the state. This way of representing Nodeland holds in the code base, in visualisations and diagrams, and in how we model plans in the Markov chain formulation. We use edges to represent geographic boundaries between VTDs, such that an edge between two nodes in the network graph means those nodes share a physical boundary on the ground. In contrast to the directed graphs given in CHAPTER 2, this edge representation does not use arrows to indicate directionality. Instead, any edge linking two nodes implies bi-directionality.

We include a number of variables for each VTD, to simulate the sorts of relevant data expected in a real state. For example, we include vote shares for the two parties of Nodeland—the Squares and the Circles—which are competitive statewide and district by district. Other variables include population, margin of victory and district assignment for the current plan. Under the initial districting plan, each district has a total of 250 voters, for a statewide total of 1,500. While the district populations are uniform under this starting plan, population between nodes varies significantly.

The full dataset is presented in TABLE 3.1, starting on this page and ending on pg. 27. We give additional statistics in TABLE 3.2 on pg. 28, which summarise population and vote totals per VTD by district and for the state as a whole.

It is worth mentioning that we do diverge from reality in a couple of meaningful ways with this dataset. For example, the vote shares of the two parties are treated as fixed in time, in contrast to the real-world case where each election brings new results. This rigidity extends to population, which also does not change between rounds of redistricting as it commonly does in reality. In addition, specific complexities like candidates and incumbency are outside the scope of this work. This results in a dataset which is quickly and easily interpreted but absent some of the concerns that plague real redistricting. As we intimated, we will address some of these challenges in CHAPTER 5.

VTD	DISTRICT	TOTAL POP.	CIRCLE VOTES (%)	SQUARE VOTES (%)
1	1	16	4 (25%)	12 (75%)
2	1	12	3 (25%)	9 (75%)
5	1	19	7 (37%)	12 (63%)
6	1	21	10 (48%)	11 (52%)
7	1	22	9 (41%)	13 (55%)
8	1	28	13 (46%)	15 (54%)

TABLE 3.1: Synthetic redistricting dataset.

VTD	DISTRICT	TOTAL POP.	CIRCLE VOTES (%)	SQUARE VOTES (%)
9	1	31	16 (52%)	15 (48%)
10	1	36	21 (58%)	15 (42%)
15	1	28	13 (46%)	15 (54%)
16	1	37	21 (57%)	16 (43%)
DIST. 1 SUBTOTAL:		250	117 (47%)	133 (53%)
3	2	23	11 (48%)	12 (52%)
4	2	17	7 (41%)	10 (59%)
11	2	32	17 (53%)	15 (47%)
12	2	22	8 (36%)	14 (64%)
19	2	33	17 (52%)	16 (48%)
20	2	24	11 (46%)	13 (54%)
27	2	29	16 (55%)	13 (45%)
28	2	26	13 (50%)	13 (50%)
29	2	22	10 (45%)	12 (55%)
30	2	22	11 (50%)	11 (50%)
DIST. 2 SUBTOTAL:		250	121 (48%)	129 (52%)
17	3	44	24 (55%)	20 (45%)
18	3	39	21 (54%)	18 (46%)
24	3	43	27 (63%)	16 (37%)
25	3	48	29 (60%)	19 (40%)
26	3	35	18 (51%)	17 (49%)
34	3	41	22 (54%)	19 (46%)
DIST. 3 SUBTOTAL:		250	141 (56%)	109 (44%)
13	4	20	8 (40%)	12 (60%)
14	4	29	16 (55%)	13 (45%)
21	4	21	10 (48%)	11 (52%)
22	4	31	17 (55%)	14 (45%)
23	4	42	24 (57%)	18 (43%)
31	4	25	13 (52%)	12 (48%)
32	4	36	21 (58%)	15 (42%)
40	4	26	13 (50%)	13 (50%)
49	4	20	8 (40%)	12 (60%)
DIST. 4 SUBTOTAL:		250	130 (52%)	120 (48%)
35	5	40	24 (60%)	16 (40%)
36	5	35	20 (57%)	15 (43%)
37	5	25	13 (52%)	12 (48%)

TABLE 3.1: Synthetic redistricting dataset continued.

VTD	DISTRICT	TOTAL POP.	CIRCLE VOTES (%)	SQUARE VOTES (%)
38	5	20	9 (45%)	11 (55%)
39	5	21	9 (43%)	12 (57%)
45	5	30	17 (57%)	13 (43%)
46	5	23	12 (52%)	11 (48%)
47	5	14	5 (36%)	9 (64%)
52	5	23	10 (43%)	13 (57%)
53	5	19	10 (53%)	9 (47%)
DIST. 5 SUBTOTAL:		250	129 (52%)	121 (48%)
33	6	23	12 (52%)	11 (48%)
41	6	17	10 (59%)	7 (41%)
42	6	19	11 (58%)	8 (42%)
43	6	22	12 (55%)	10 (45%)
44	6	28	16 (57%)	12 (43%)
48	6	13	4 (31%)	9 (69%)
50	6	15	8 (53%)	7 (47%)
51	6	17	9 (53%)	8 (47%)
54	6	16	7 (44%)	9 (56%)
55	6	15	7 (47%)	8 (53%)
56	6	14	4 (29%)	10 (71%)
57	6	16	5 (31%)	11 (69%)
58	6	11	4 (36%)	7 (64%)
59	6	12	5 (42%)	7 (58%)
60	6	12	6 (50%)	6 (50%)
DIST. 6 SUBTOTAL:		250	120 (48%)	130 (52%)
GRAND TOTAL:		1500	758 (50.5%)	742 (49.5%)

TABLE 3.1: Synthetic dataset, with initial district assignments, used to demonstrate the process of redistricting using MCMC. The competitiveness within these initial districts is such that both parties win three districts. Note that each district has exactly 250 voters and that both parties are competitive in each district. However, individual VTDS possess significant variance in both population and vote proportions.

DISTRICT	TOTAL POP		SQUARE VOTE		CIRCLE VOTE	
	MEAN	STD. DEV.	MEAN	STD. DEV.	MEAN	STD. DEV.
1	25	7.9	11.7	6.0	13.2	2.2
2	25	4.8	12.1	3.4	12.9	1.7
3	41.7	4.1	23.5	3.7	18.2	1.3
4	27.8	7.1	14.4	7.1	13.3	2
5	25	7.5	12.9	5.5	12.1	2.33
6	16.7	4.5	8	3.5	8.7	1.7
STATEWIDE	25.0	9.3	12.7	6.3	12.4	3.30

TABLE 3.2: Summary statistics of VTD variables by district and for the entire state under the initial plan. For example, the average total population for the VTDs of district 1 is 25 persons. District 3 in particular stands out for its high mean population relative to other districts. We also note that the standard deviations of the Square's vote share are considerably higher than those of the Circle's.

To compliment the raw data presented in TABLES 3.1 and 3.2, we offer a set of four figures. FIGURES 3.1 and 3.2 on pg. 29 show two 'global' looks at the data; first by population of each VTD (FIGURE 3.1) and again by population but with district assignments overlaid (FIGURE 3.2). With FIGURES 3.3 and 3.4 on pg. 30, we depict the margins of victory (or loss) for each party by VTD.

Close examination of TABLES 3.1 and 3.2 and FIGURES 3.1 to 3.4 offers some important preliminary conclusions. We note from FIGURES 3.1 and 3.2 that the centre of Nodeland possesses a high density population. This is captured in particular in district 3, where the entire district is defined by just six nodes with a district mean population of 41.7. As expected, this density decays as we extend to the peripheries and the number of nodes populating each district increases. This is characteristic of real world population and districting dynamics, as we saw in FIGURE 1.1.

Observing TABLES 3.1 and 3.2 in concert with FIGURES 3.3 and 3.4, we gain insight into the competitive nature of the state. Each party wins three districts but in contrasting fashion. The Circle party appears strong in higher population areas while the Square party wins out in the rural areas. This is further reinforced by FIGURES 3.3 and 3.4 where we can clearly see the margins afforded to each party.

These margin plots illustrate that it is not only population values which vary but vote shares between the parties. In fact, comparing the Circle's best VTD to their worst, we see a 75 point difference. The combination of the variance in population and the variance in vote margin across nodes is critical to gerrymandering the state. As the reader will recall from SECTION 1.4, these differences are the building blocks of packing and cracking.

The holistic view of the dataset presented in this section provides the context we require going forward. Understanding the interplay between node populations and district assignments is essential for our next sections, where we will see how redistricting plans are generated using MCMC.

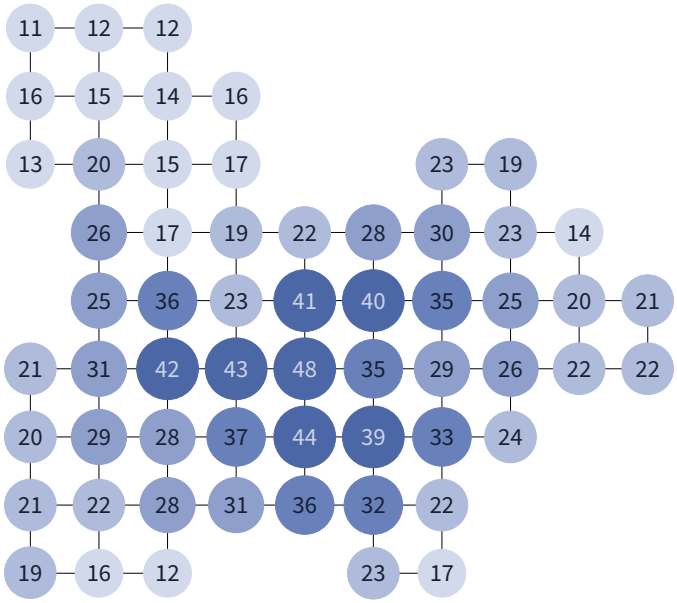


FIGURE 3.1: Total voting population for each VTD is annotated over the top of each node. Population values do not vary under any simulations. Shading and node size correspond proportionally to the population value. Note that the smallest VTD has 11 voters while the largest has 48.

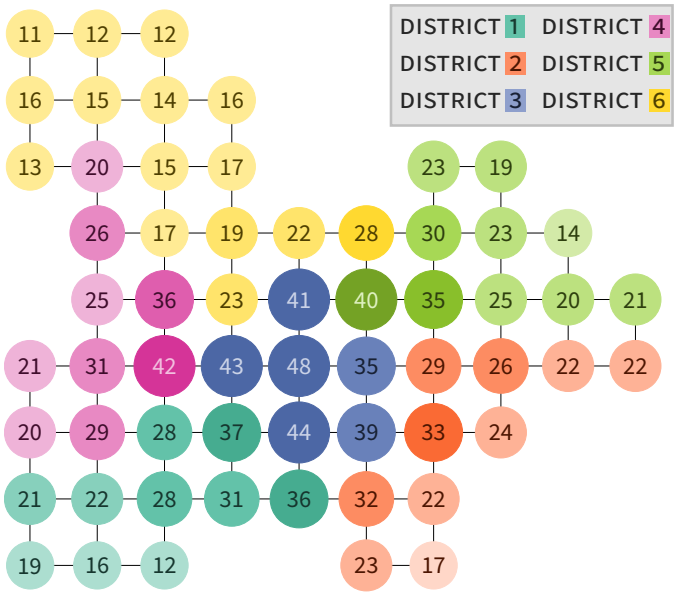


FIGURE 3.2: Total voting population for each VTD is annotated numerically and district assignments under the initial plan are provided in colour. Due to population dynamics across the state, the number of VTDS per district varies considerably. District 3 reaches 250 persons with just six nodes, while it takes 13 nodes for district 6.

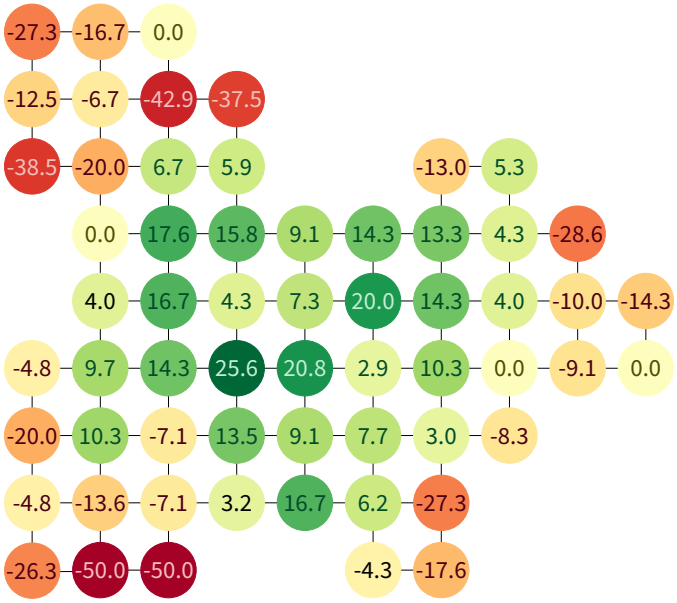


FIGURE 3.3: Vote margin for the Circle party presented as a heat map. Negative numbers with orange or red circles correspond to VTDs lost by the Circles. Conversely, positive numbers with green or yellow circles are VTDs the Circles have won. Vote margin is computed as the difference of the Circle's vote proportion minus the Square's vote proportion. Though visually it might appear the Circle party is a heavy favourite, in fact they accrue a slim majority with 50.5% (758 of 1500) of the statewide vote.

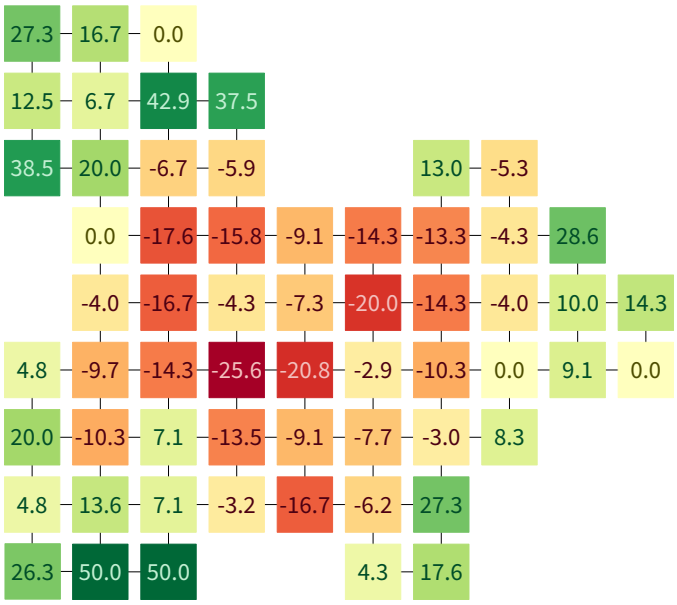


FIGURE 3.4: Vote margin for the Square party presented as a heat map. Negative numbers with orange or red circles correspond to VTDs lost by the Squares. Conversely, positive numbers with green or yellow circles are VTDs the Squares have won. Vote margin is computed as the difference of the Square's vote proportion minus the Circle's vote proportion. Though visually it might appear the Square party is at a disadvantage, in fact they accrue 49.5% (742 of 1500) of the statewide vote.

3.3 GENERATING REDISTRICTING PLANS

As we alluded to at the beginning of this chapter, designing a model for the redistricting problem such that it conforms to the requirements of the Metropolis–Hastings algorithm is not entirely straightforward. This is largely due to the nature of our goal. We desire a distribution representing the set of *legal* and unbiased districting plans of Nodeland, so that we might test a given plan against this distribution. That is, given a plan suspected of being a gerrymander, we wish to gain evidence of its partisan bias by examining with what probability it falls in the unbiased distribution of valid plans.

How then might we go about producing this unbiased distribution? We know from CHAPTER 1 that there are some constraints on what a valid and legal plan looks like. For instance, districts must be contiguous and each one must be of roughly the same population. Since we are seeking to avoid any bias, so long as a plan meets the legal requirements, we make no other judgements on its quality or character. All legal plans are created equal. However, as we will see in SECTION 3.6, practical factors mean producing this set of entirely legal plans will require us to first generate many thousands of illegal plans.

To start generating samples using the Metropolis–Hastings algorithm we require three key components: an ergodic Markov chain, a proposal mechanism and an acceptance probability. We will describe each of these in detail in separate upcoming sections. The simple summary is as follows. Our Markov chain is designed such that the nodes of Nodeland are fixed; the state space is defined by plans which map nodes to districts. Proposals for new states are generated by perturbing the district assignment of one node in the graph. Acceptance or rejection is defined by a scoring function that is calculated on both the current and proposed plans.

3.4 DEFINING THE MARKOV CHAIN

In order to make use of the essential results we presented in CHAPTER 2, we need a Markov chain. For a stochastic process to be a Markov chain, we need the future state of the chain to be independent of the past given the present. As we saw in CHAPTER 2, this means the probability that we arrive in future state s_j depends only on our current state s_i. Moreover, if the chain is ergodic, we can use it to inform us about its underlying unique stationary distribution. In this section, we will introduce how we model the redistricting problem as a Markov chain. With that established, SECTION 3.4.1 will look at satisfying the ergodicity requirement.

One way to think of a Markov chain is by its essential components: the state space and the transition probabilities. In the redistricting case, we model states of the chain as a function ϕ which assigns each fixed node to a specific district. With the nodes treated as fixed, any ϕ_k forms a state of the chain by mapping those nodes to specific districts. This allows us now to define the state space of our Markov chain.

Definition 3.1. We have that the state space S of the redistricting Markov chain X is given by:

$$S = \{\phi_1, \phi_2, \dots, \phi_k\}, \tag{3.1}$$

where each ϕ_k is a distinct function,

$$\phi : V \subset \mathbb{N} \to D \subset \mathbb{N}, \tag{3.2}$$

and where V is the set of all nodes and D is the set of assignable districts.[2] This gives us the function notation,

$$\phi(v) = d \tag{3.3}$$

for an arbitrary node $v \in V$ and district assignment $d \in D$.

By defining the state space using a mapping from nodes to districts, we concisely encapsulate the key component of redistricting. That is, each VTD in Nodeland is given a district assignment by ϕ. However, it is worth being especially clear that the state space is made up of a series of these mappings $\phi_1, \phi_2, \dots, \phi_k$, which allows us to capture the full range of possible plans.

With our state space defined, we also need to define transition probabilities between states to remain consistent with the Markov property. To transition between two arbitrary states, we need them to be *adjacent*. We define adjacency as follows.

Definition 3.2. Consider two distinct districting plans ϕ and ϕ', mapping a fixed set of nodes to a number of districts. These plans are called *adjacent* if they differ by one and only one district assignment, otherwise they are called *non-adjacent*.

Having a definition for adjacency, we can now define the transition probabilities for the chain.

Definition 3.3. Consider two districting plans ϕ and ϕ'.

1. If they are identical or non-adjacent, the probability of transitioning from ϕ to ϕ', given by $T_{\phi,\phi'}$, is always 0.
2. If they are adjacent, where ϕ assigns node v to district d_i and ϕ' assigns v to district d_j, then:

$$T_{\phi,\phi'} = \frac{1}{|V|} \frac{\deg_j(v)}{\deg(v)} \tag{3.4}$$

where $|V|$ is the cardinality of V (total number of nodes), $\deg(v)$ is the number of neighbours of v, and $\deg_j(v)$ is the number of neighbours of v belonging to district d_j.

These transition probabilities will be made clearer when we look at generating proposals in SECTION 3.5. However, the intuition for DEF. 3.3 effectively comes down to the probability of altering the district mapping of one node in ϕ

2 V for VTD and D for district

to give ϕ'. It is easiest to think of this by starting at ϕ and seeing how we would arrive at ϕ'. Start by selecting a random node $v \in V$, with probability $1/|V|$, where v is mapped to district d_i by ϕ. Then select a node w mapped to district d_j (with $i \neq j$) from the open neighbourhood $N_G(v)$ and alter the district assignment of v to d_j. This alteration occurs with probability $\deg_j(v)/\deg(v)$ and hence results in a new districting plan ϕ' which is adjacent to ϕ.

With DEFS. 3.1 to 3.3, we have adapted the redistricting problem space into a Markov chain. We have a sequence of plans, the probabilities of which depend only on the previous plan, and a set of transition probabilities to move between these plans. Let us now look to satisfying the properties of SECTION 2.2 that we require.

3.4.1 *Satisfying Chain Properties*

With our Markov chain model of the redistricting problem, we see similarities to the chains presented previously, especially when it comes to the network graph model. However, as we have just seen, the state space of this new chain is decidedly more complex than in the examples of CHAPTER 2. For instance, where the transition probabilities in FIGURES 2.1 to 2.5 were conjured from thin air by the author, those given by EQ. 3.4 are inferred from the district assignments of neighbouring nodes.

At first glance, the added complexity of the redistricting Markov chain would seem to make it more difficult to demonstrate properties of the chain. However, this is not the case. Keying off of the graphical representations given by FIGURES 3.1 to 3.4, we can see some of the properties we presented in SECTION 2.2.

Since the graph is a grid and all nodes are connected to their nearest neighbours with bi-directional edges, we can see clearly that this chain is irreducible: all nodes are accessible from all other nodes. With this in mind, it follows directly from THEOREM 2.8 that this chain is also positive recurrent, since the state space, while extremely large, is still finite. Lastly, aided by the irreducibility of the chain, we note the absence of any periodicity, making the chain aperiodic. With these properties, we have satisfied the requirements for ergodicity (see SECTIONS 2.2.4 and 2.3). This is a requirement of the Metropolis–Hastings algorithm and will be discussed once again in SECTION 3.6 when we look at the acceptance probability used to sample from our stationary distribution π. However, we will first look at how we propose new states.

3.5 PROPOSING NEW STATES

As we saw in SECTION 2.5, the first step (after initialisation) of the Metropolis–Hastings algorithm is to propose a new state in the chain. In the redistricting case, we produce these proposals using what DeFord, Duchin and Solomon [22] and Ellenberg [25] call the *flip* method. It works by selecting a node on the graph at random, which then has its district assignment altered to match the assignment of one of its neighbours. If this sounds familiar, that is because it is; the transition probabilities presented in SECTION 3.4 give us the mechanism we use for proposing new states.

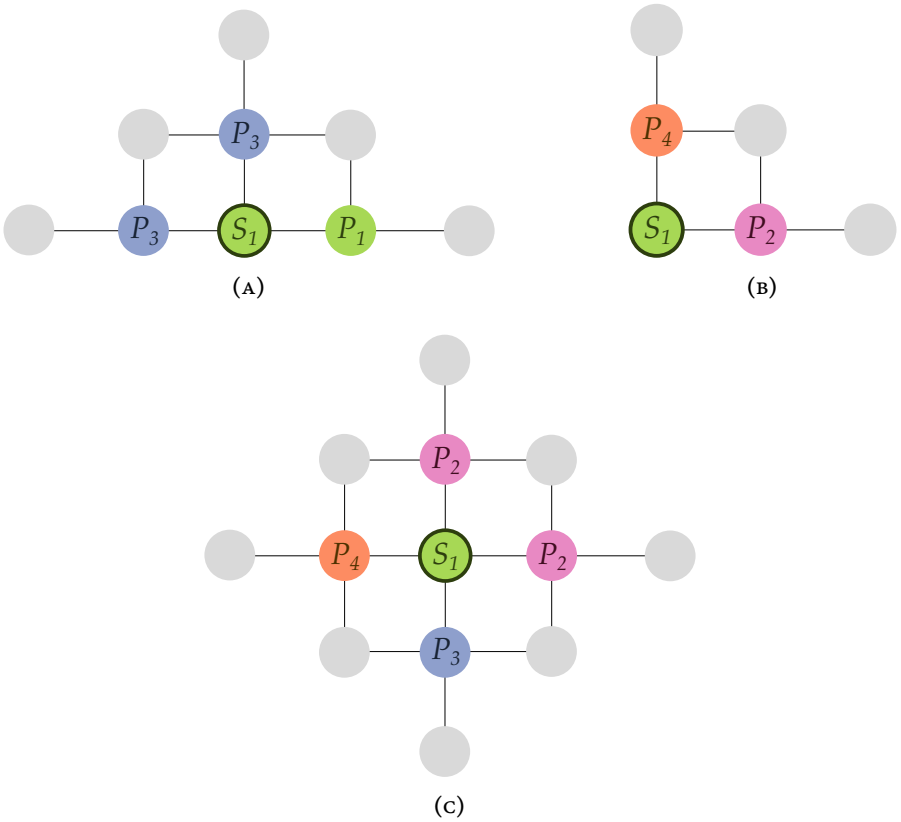


FIGURE 3.5: Three scenarios illustrate the basic intuition behind the *flip* proposal. Green S nodes represent the *source* node in each case. The P nodes are possible *proposal* nodes. Node colour and subscript label describe the district assignment of the given node. FIGURE A gives the 'perimeter' case, where the source node can access three proposal nodes. In this case, district d_3 (blue) is the only possible assignment change for S_1. FIGURE B offers the 'corner' case, where the source node can only access two proposals. Here district d_2 (pink) or district d_4 (orange) are available flips. Finally, FIGURE C shows the 'core' case, where the source node can access proposals in all directions, with districts d_2 (pink), d_3 (blue) or d_4 (orange) on offer.

As DeFord [21] mentions, it is not uncommon to use this method of proposal, where the transitions at each state are also the means of generating proposals. While by no means the only method of proposing new states, as we will see in CHAPTER 5, flip does have the advantage of being easier to implement and describe. As we have already shown how we generate proposals from the mathematical perspective—by describing the transition probabilities in EQ. 3.4—in this section we will give more intuition and show how it is executed in the source code.

Looking to FIGURE 3.5, we see a trio of simple diagrams, showing schematically how a proposal is implemented. It effectively begins with selection of a *source node* at random from the pool of available nodes in the current graph ϕ (using the notation of SECTION 3.4). From this source node's open neighbourhood, we select a *proposal node*. We then propose changing the district assignment of the source node to the current district assignment of the pro-

```python
from random import randint as rint

def transistion(graph):

    ncount = len(list(graph.nodes))
    sourceNode = rint(1, ncount)
    nbors = list(graph.neighbors(sourceNode))
    nbors_distrs = []
    for node in nbors:
        nbors_distrs.append(graph.nodes[node]["distr"])

    cur_distr = graph.nodes[sourceNode]["distr"]
    prop_distr = nbors_distrs[rint(0, len(nbors_distrs) - 1)]

    trans_out = nbors_distrs.count(prop_distr) / len(nbors)
    trans_in = nbors_distrs.count(cur_distr) / len(nbors)

    trans = {
        "node": sourceNode,
        "prop_distr": prop_distr,
        "trans_out": trans_out,
        "trans_in": trans_in,
    }

    return trans
```

LISTING 3.1: A simple function for proposing a new state which is adjacent to the provided state. Since each state in the Markov chain is a districting plan, and in the code base each plan is represented by a network graph, states are called graphs here. A source node is selected at random using `randint`. This standard library method is used again to select a proposal node randomly from the open neighbourhood of the source node. Transition probabilities in both directions are computed. The source node, the randomly selected proposal district, and transition probabilities are returned in a dictionary for later use.

posal node. This becomes the new proposal state ϕ', which we take forward to our Metropolis–Hastings algorithm, given in full in SECTION 3.7.

In LISTING 3.1, we give the `python` implementation of flip using our function called `transition`. The result of this function is the same as that outlined in the previous paragraph, but implementation is slightly different. It works as follows.

The function is called by providing a graph, which represents ϕ in the notation of SECTION 3.4. From this graph, a source node is selected at random by its integer label, using the random integer generator of the standard library. The neighbourhood of the source node is accessed and the district assignments of all nodes in the neighbourhood are logged. One of these district assignments is selected at random and becomes the proposal district. This defines ϕ' and hence the transition probabilities in both directions (ϕ to ϕ' and ϕ' to ϕ) are

computed. The source node, the proposal district, and the transition probabilities are returned in dictionary form.

Using this short function we are able to produce proposals relatively quickly, which is imperative when we might be calling this function thousands or millions of times. Once a new state has been proposed given the current state, we progress to the next stage of the Metropolis–Hastings algorithm: computing the acceptance probability.

3.6 ACCEPTING AND REJECTING PROPOSALS

Having established our method for generating proposal states, we have now arrived at the point where we need to define our acceptance probability. Recall from SECTION 2.5 that the acceptance probability is defined by the transition probabilities between two states and the probabilities of those states subject to the target distribution π. The reader will also remember that provided we have selected an ergodic Markov chain, this acceptance probability allows us to produce samples proportional to π. However, as we foreshadowed at the start of this chapter, our target distribution is made complicated by the legal concerns of the redistricting problem.

Our main desire is to produce samples from the distribution of *legal* plans of Nodeland. For our purposes, we consider two legal constraints: 1) all districts in a given plan must be contiguous, and 2) all districts must be of equal or near equal population. In the first instance, we define contiguity by all nodes in the district being connected by edges; no 'orphans' or 'islands' are allowed. In the second instance, we allow some tolerance on the population constraint due to the limited number of VTDS in Nodeland.

At first glance, one might think to define a distribution π which describes these two constraints and be done. That is, all non-contiguous and/or poorly population-balanced plans have density of zero and hence are always rejected by the acceptance probability. However, if we consider this in practice, we realise it will not work. The first—or indeed any—proposal we make will violate one or both of these constraints. Since the initial districting plan is perfectly population-balanced and contiguous, changing the district assignment of any one node will violate the population constraint. Hence we will be unable to motivate acceptance of *any* proposal.

Our solution to this quandary is to open up the distribution π to tolerate illegal plans. This allows for population variation to extend into illegal population proportions but in doing so also allows for much more complete exploration of the state space. We can think of this as softening or rounding off the edges of our target distribution, while still biasing in favour of contiguous and well-balanced plans.

Under this paradigm, we use Metropolis–Hastings to sample from a distribution containing both legal and illegal plans. This is our π from which we sample proportionally. We then deploy a sort of rudimentary importance sampling to extract only the plans which satisfy all legal requirements. Hence, we produce a so-called π', which is effectively a uniform random sampling of all legal districting plans for Nodeland. We will set aside π' for now and come back to it after we have presented our complete Metropolis–Hasting al-

gorithm in SECTION 3.7. For now, we will concern ourselves exclusively with first showing how to sample proportionally to π.

3.6.1 *Target Distribution*

In defining our target distribution π, we seek a means of discriminating between legal and illegal plans. We want to sometimes accept the latter, as a means of exploring the state space. However, we also want a centre of mass that draws us back to more reasonable plans after venturing out. With inspiration from DeFord [21], we arrived at an 'energy' function as a good representative of these goals. The 'template' or parent of this function takes the form $g(x) = e^{-x}$ and is shown in FIGURE 3.6 for context. One major value of using such an exponential is that it only returns values between 0 and 1, making the computation of the ratio between two results rather approachable.

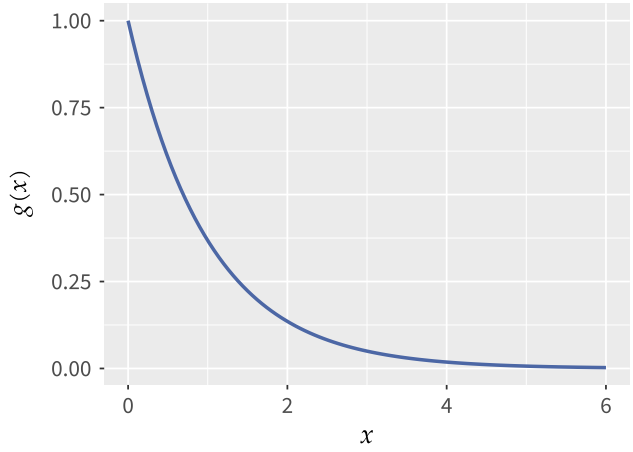


FIGURE 3.6: Plot demonstrating the behaviour of the parent function, $g(x) = e^{-x}$, of our energy function.

We can recast this energy function as a *score function*, given by $f(\phi)$, which accepts a districting plan ϕ and evaluates it based on contiguity and population metrics. This means that when considering our score function, we need to imagine two outputs: the value of $f(\phi)$ produced by the current plan ϕ and the value produced by the proposal plan ϕ'. Recall from SECTION 2.5 that the target distribution arrives in the acceptance probability in the numerator and denominator and thus gets evaluated as a ratio of the two scores of ϕ and ϕ'.

We define our score function with a multiplicative constant and a series of sub-scores as follows.

Definition 3.4. Define $f(\phi)$, for a districting plan ϕ and a finite number of sub-scores σ_i, which are functions of ϕ, as

$$f(\phi) = \exp\left\{-c \cdot \prod_{i=1} \sigma_i\right\}, \tag{3.5}$$

where c is a user-controlled normalising constant that adjusts the acceptance rate.

Under this definition, a perfect score will yield a product of sub-scores equal to zero, such that $f(\phi) \to 1$ as the products in the exponential tend to 0. All other scores will be between 0 and 1 exclusively. For our purposes, we use one so-called *sigma function* sub-score for scoring population and another for scoring district contiguity. This are described in detail in the next two subsections.

3.6.2 *Scoring Population*

The first sigma function, σ_p, computes a score for population parity between districts. This takes the form of a simple variance computation.

Definition 3.5. Define the population sigma function σ_p by,

$$\sigma_p = \frac{1}{D} \sum_{i=1}^{D} (z_i - \bar{z})^2, \tag{3.6}$$

where $\bar{z} = 250$ is the mean district population, z_i is the population of district d_i and D is the number of districts.

Note that a low value of σ_p means the districts are near population equality, which in turn produces a better (meaning closer to 1) output from the score function $f(\phi)$.

Variance was specifically chosen as the statistic for this sub-score because it ranges over multiple orders of magnitude. This results in more acute discrimination between differing proportions of population balance. Through fine-tuning of the normalising constant c, we can achieve scores that result in enough illegal plans being accepted to adequately explore the state space while still regularly identifying well-balanced, legal plans. This relationship is addressed in detail in SECTION 3.8.

3.6.3 *Scoring Contiguity*

The second sigma function, σ_c, analyses the supplied districting plan ϕ for district contiguity and effectively bans the plan if any districts are non-contiguous. To deduce that an entire district is contiguous, two numbers are required. The first is the total count of nodes in the district of interest. The second—call it the *traversal count*—is the number of nodes which can be reached, without leaving the district, from an arbitrary node within the district.

If the district is contiguous, these numbers should be the same. Otherwise, if it is split into at least two pieces, the traversal count will be less than the total. Computation of the total number of nodes in a district is a trivial affair. For the traversal count, we use the following algorithm adapted from Taniguchi [58] for our purposes.

An iteration of a while loop begins after an arbitrary starting node is added to the traversal *queue*. From this node, all nodes adjacent to it are identified and nodes of the same district as the starting node are added to the queue. Hence, the queue tracks nodes which need to be *explored*. For a node to be considered explored, its full set of neighbours must be registered and any of the same

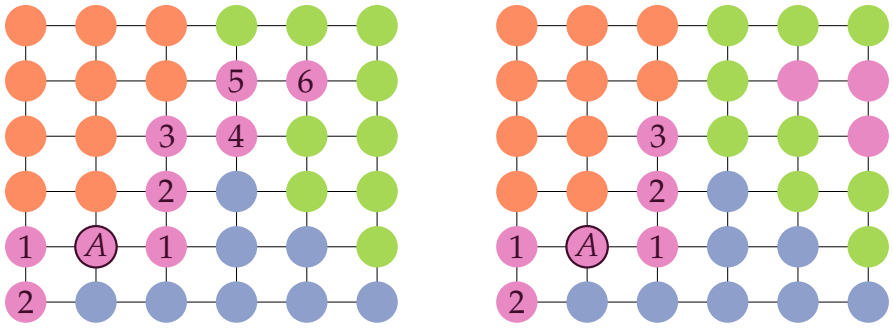


FIGURE 3.7: Two possible contiguity scenarios for the pink district. Starting at the arbitrary node A, numerals indicate at which stage of the search the given node is added to the queue. For example, the two nodes directly adjacent to A are added to the queue on the first iteration of the search. With FIGURE A, starting from node A, the search function returns nine connected nodes after six total iterations. Since this equals the total number of pink nodes, the function concludes this district is contiguous. Conversely, FIGURE B returns six connected nodes, which does not match the total number of pink nodes. Hence, the function concludes in this case that the district is non-contiguous.

district appended to the queue. Once a node is fully explored, it is removed from the queue and added to the *visited* list. The while loop finishes when the queue is empty, and it then returns the visited list. Counting the number of elements in this list gives us the traversal count. A simple comparison of this count to the total number of nodes in the district tells us if the entire district is contiguous. The process described in text here is presented in graphical form in FIGURE 3.7 and in `python` source code in LISTING 3.2 on pg. 40.

When it comes to the sigma function for district contiguity, computation is very straightforward in comparison to the search algorithm itself. After considerable testing, we concluded that allowing non-contiguous districts in the chain iteration produced results with very few legal plans. As a result, we wrote the contiguity sigma function as a ban on any ϕ which contains one or more non-contiguous districts.

Definition 3.6. Define the contiguity sigma function σ_c, for the number of non-contiguous districts d_{nc} in plan ϕ, as:

$$\sigma_c = \begin{cases} 1 & \text{if } d_{nc} = 0 \\ 10^6 & \text{otherwise} \end{cases} \tag{3.7}$$

This means that anytime a plan ϕ possesses one or more non-contiguous districts, the score returned by σ_c is so high that $f(\phi)$ effectively equals 0. Plans with full contiguity have no impact on the final score of $f(\phi)$.

```python
from collections import deque

def bfs(node, graph):

    # Initialise required data structures
    queue = deque([node])
    visited = [node]
    distr = graph.nodes[node]["distr"]

    while queue:
        curr_node = queue.popleft()
        nbors = list(graph.neighbors(curr_node))
        d_nbors = [
            item for item
            in nbors
            if graph.nodes[item]["distr"] == distr
        ]
        for n in d_nbors:
            if n not in visited:
                queue.append(n)
                visited.append(n)

    return visited
```

LISTING 3.2: A breadth-first search algorithm for traversing the contiguous nodes of a district as implemented by the author in `python`.

Putting it all together, the two sigma functions are then combined into the final score function.

Definition 3.7. Define the complete score function $f(\phi)$ for districting plan ϕ as:

$$f(\phi) = \exp\left\{-c \cdot \sigma_c \cdot \frac{1}{D}\sum_{i=1}^{D}(z_i - \bar{z})^2\right\}, \tag{3.8}$$

with $\sigma_c, \bar{z}, z_i$, and D defined as before. Our chosen value for c is 0.0025.

With this definition of the score function, we now have the target distribution π which we will sample from using Metropolis–Hastings. As a consequence, we can now present our final algorithm in full.

3.7 METROPOLIS–HASTINGS FOR REDISTRICTING

In the preceding sections, we defined our redistricting Markov chain and showed that it is ergodic. We presented our methodology for proposing new states. We gave our target distribution and acceptance probability. With these components in hand, we have what is required to populate the Metropolis–Hastings algorithm presented in SECTION 2.5. We offer it now, step-by-step, as it is used to sample proportionally to the unique stationary distribution π of the redistricting Markov chain.

0. First iteration only:
   a) Set iteration counter $i = 1$
   b) Initialise with districting plan $\phi = \phi_0$, where ϕ is the current plan and ϕ_0 is the pre-defined initial plan
1. Select a *source* node v from the set of all nodes V at random with probability $\frac{1}{|V|}$, where $|V|$ is the cardinality of V
2. Select a *proposal* node w from the open neighbourhood $N_G(v)$ of the source node, with probability $\frac{1}{\deg(v)}$, where $\deg(v)$ is the degree of v
3. Generate proposal plan ϕ', which mirrors ϕ except that $\phi'(v) = d_w$, where d_w is the district assignment of the proposal node w
4. Compute transition probabilities $T_{\phi,\phi'}$ and $T_{\phi',\phi}$, subject to EQ. 3.4.
5. Compute scores $f(\phi)$ and $f(\phi')$, as given by EQ. 3.8
6. Compute acceptance rate α using transition probabilities and scores, given by
$$\alpha = \min\left(1, \frac{f(\phi')}{f(\phi)} \frac{T_{\phi',\phi}}{T_{\phi,\phi'}}\right), \tag{3.9}$$
   - *Note: when a node is the last of its district, or orphaned as its own island, $T_{\phi',\phi} = 0$ and therefore $\alpha = 0$. Hence, proposals in these scenarios are always rejected*
7. Sample a uniform random number $\beta \sim U(0, 1)$
8. Either:
   a) *Accept* proposed districting plan ϕ' if $\alpha > \beta$, and:
      - Set $\phi = \phi'$
      - Save ϕ' to set of accepted plans G
      - Increment i by 1
   b) Or *reject* proposed districting plan ϕ' is $\alpha < \beta$, and:
      - Set $\phi = \phi$
      - Increment i by 1
9. Repeat, until i exceeds chosen threshold

By iterating this algorithm for large i, the list of accepted plans G will tend proportionally to the stationary distribution of π. However, as we described previously, π is not our final distribution of interest. It will invariably contain many illegal plans which are not desirable for our purposes. To derive the distribution we will use for analysis, we will filter π to produce π'. This is the subject of SECTION 3.9. Before this, however, we will take a brief interlude to explore our value for the constant c.

3.8 PARAMETERISATION

The constant c of the scoring function is unobtrusive but it carries significant weight. Effectively, it controls the scaling of the variance value returned by the population sub-score σ_p. Technically, it scales the result of multiplying the population *and* contiguity sub-scores. However, as we saw in SECTION 3.6.3, when all districts are contiguous, the value of σ_c is 1. When even one of the plan's districts is non-contiguous, the value returned is 10^6, which is high enough to 'override' the constant c. Let us look at this in more detail.

In choosing the value of this constant, our primary goal is to scale the results of the population sub-score such that low-variance, well-balanced plans produce a high—that is closer to 1—score out of $f(\phi)$. This means we need to know the range and order(s) of magnitude we expect to see from σ_p. To this end, we jump ahead slightly and provide a look into our final results with FIGURE 3.8. This figure depicts results of the third set[3] of 500,000 iterations of the chain. With FIGURE 3.8A, we show the scoring results given by $f(\phi)$ and with FIGURE 3.8B, we give σ_p, which outputs population variance. Both plots are sorted in descending order of largest to smallest final score.[4]

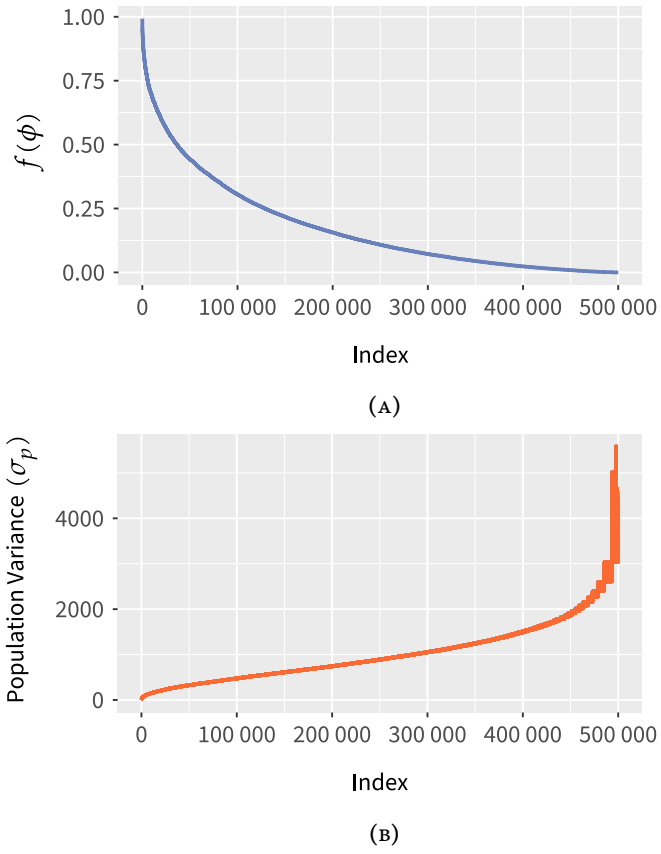


FIGURE 3.8: Two plots shows the relationship between the final score returned by $f(\phi)$ (FIGURE A) and the population variance given by σ_p (FIGURE B). Both plots are sorted by score ($f(\phi)$) in descending order.

3 Logging the entire simulation in one file became untenable, so we split it into eight pieces of 500,000 each

4 That is, these are both sorted in descending order of the value of $f(\phi)$

We can see from these plots that the value of c we chose for scoring induces the behaviour we desire. Namely, the steep gradient of FIGURE 3.8A shows that variance values are weighted increasingly heavier as they get smaller. Conversely, we note that by index ~200,000, $f(\phi)$ is returning scores of about 0.12 to 0.15.

This illustrates a pair of consequences. First is that we do not discriminate excessively between high variance values. For instance, retrieving variances of 1,200 and 1,700 for two discrete plans produces scores of ~0.04 and ~0.01 respectively. Secondly, discrimination is more acute for low variance values. This serves to accentuate disparities between plans. Recall that the Metropolis–Hastings acceptance probability uses a ratio of proposal plan over current plan. This means that the steepness of FIGURE 3.8A makes us increasingly likely to accept proposal plans as they return lower variance values.

Having described the impacts of the constant c, the natural question is: how did we select our final value of 0.0025? On this topic, we offer FIGURE 3.9 to provide context. This figure illustrates why considerable testing was required to arrive at our final value. Choosing a large value of 0.02, for example, effectively bans plans with variances above 250 and severely limits the exploration of the state space. Conversely, a very small value does not adequately penalise bad population balance, reducing the desire for the chain to magnetise towards the well-balanced legal plans we seek. All of which serves to illustrate that the choice of c has a tremendous impact on the acceptance rate of any given plan.

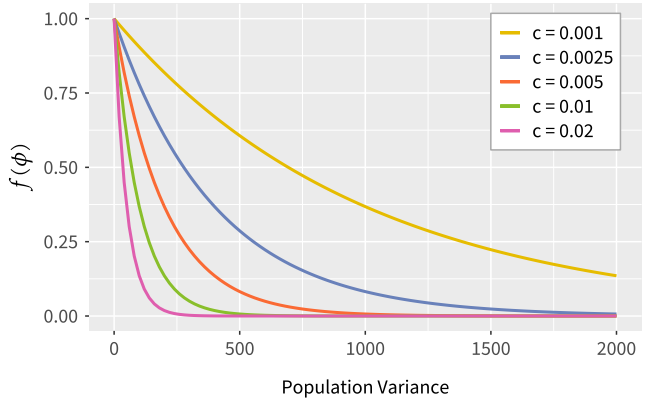


FIGURE 3.9: Five curves show the impact differing values of c have on the relationship between σ_p and $f(\phi)$.

3.9 CONDITIONING

As we have said, in order to derive the final distribution π' which we will use for analysis, we must filter out some of the plans present in π. There are two categories present in π that we wish to filter out: illegal plans and duplicate plans. We will address each of these in turn.

The reader will recall that we are banning non-contiguous districts from being accepted as per SECTION 3.6.3. Therefore we require no extra handling for this type of illegal plan. However, we are more generous in our scoring when

it comes to population imbalances. This means that many plans get accepted which violate even the most generous interpretations of the word balanced.

While the real congressional districts of the United States are almost exclusively equal in population to within one person, we are slightly more open in our filtering. This generosity is driven by the fact that there are very few, if any, plans outside the initial one which maintain exact population balance across all the districts of Nodeland. As such, we reject plans where any district exceeds ± 5% of the ideal average district population. This means that any plan with one district or more of under 237 or over 263 total persons is rejected. Thus we arrive at a subset from all the plans we sampled which is made entirely of legal districting plans.

The second filter we use is one to weed out duplicates. With so many iterations, it is not uncommon for the chain to produce duplicate plans which meet all other conditioning criteria. By duplicates, we mean that for two district mappings ϕ_1 and ϕ_2,

$$\phi_1(v_i) = \phi_2(v_i) \quad \forall v_i \in V.$$

We remove such duplicates to avoid repeat data points that would bias statistical measures in our analysis. That is, as we stated in SECTION 3.3, all legal plans are created equal and we want the final distribution to reflect this. To identify these duplicates, we compute a *signature* for each of the legal plans that pass the population filter. This signature is simply the integer label of each node followed by its district assignment in sequence. For example, with three nodes labelled 1, 2, and 3, mapped to districts 4, 4, and 5 respectively, our signature would be 142435. Since the final set of legal plans is relatively small, comparison of every signature to every other signature is not computationally intractable.

The discerning reader will notice that we do not have a mechanism for removing plans with the same 'skeleton' but different district assignments. By this we mean that the shapes of each district are the same between two plans, it is just the numerical district assignment which differs. One might refer to these as isomorphic plans. We have left these plans in primarily due to the complexity of identifying and removing them. In practice, leaving the plans in does not meaningfully bias results, as we will see from the consistency of the district statistics given in SECTION 4.3.

After our filtering process, we are left with the final distribution π' of legal districting plans which represent the electoral potential of Nodeland. By this we mean a representation of the different ways in which the state can be carved into six contiguous districts of near-equal population. In the next chapter, we will explore this distribution in detail. We will present a bespoke plan we have created that seeks to gerrymander the state. Then, we will compare the two through a series of analyses to demonstrate whether or not the plan is indeed a gerrymander.

4

RESULTS & ANALYSIS

With our methodology described fully in chapter 3, we now come to the second half of our two chapter discourse on MCMC and redistricting. In this chapter, we will present the results of simulating our Markov chain to sample districting plans. More specifically, we will examine the post-processed distribution π' which represents the set of legal plans we sampled. We will present this distribution as a side-by-side comparison, looking at an allegedly gerrymandered plan against π'.

Presenting results of the distribution and a specific plan together allows us to emphasise the unbiased nature of π'. It also triggers the first in a series of analyses we will conduct in our mission to gather evidence which shows the plan in question is gerrymandered. In fact, it is our primary goal to show that comparison against π' demonstrates the plan is gerrymandered. We will introduce a statistic called the *efficiency gap* to further reinforce this claim. Finally, we will conclude with a brief analysis of the proposal results of our Markov chain.

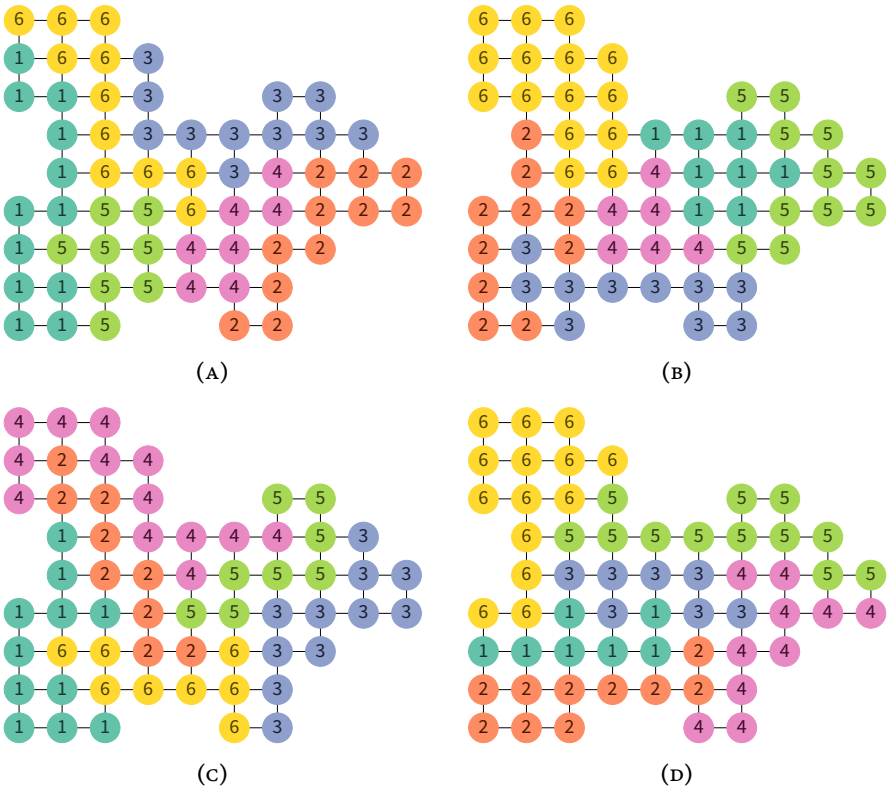
4.1 SIMULATING PLANS IN PRACTICE

In order to produce an adequate sampling of legal districting plans for our synthetic dataset, we had to iterate the Markov chain many, many times. In fact, after considerable testing, we arrived at four million iterations for our final result. While it may have been possible to extract a few more viable samples with a couple million more iterations, we reached the limit of our machine's 32GB of Random Access Memory (RAM) at just over four million.

From these four million iterations, 412,774 raw plans were accepted, which make up our sample proportional to the distribution π. After filtering and removal of duplicates as per section 3.9, we were left with 1194 unique, legal districting plans for Nodeland. This is our π'. A random selection of some of these plans is presented in figure 4.1.

We then held elections under each plan individually. This was done by summing the vote share variable, for each party, district-by-district. Hence, under a given plan, election results give the number of votes each party received in each district in that plan. From these vote shares, we were able to compute the number of districts each party won under each plan. The end result is an ensemble of election results representing the potential electoral success (or failure) of each party.

Looking ahead, we will consider the dataset of districting plans sampled from our Markov chain in their entirety, instead of at the individual plan level. This is due to the fact that, taken in isolation, a single plan is not instructive of the potential of the state. Indeed, looking at the full suite of results matches exactly with our goal of analysing a gerrymandered plan against an unbiased



(A) (B) (C) (D)

FIGURE 4.1: Four of almost 1,200 legal districting plans produced by our MCMC redistricting method. District assignments are described with numerical annotations as well as colour. In contrast to our initial plan seen in FIGURE 3.1, these plans show a range of 'styles' for districting the state. For example, FIGURE A contains a handful of 'spindly' districts that snake around the state, often contracting down to just one node in width.

distribution of legal plans. Before conducting this analysis however, we will first introduce the gerrymandered plan we have devised.

4.2 DRAWING A GERRYMANDER

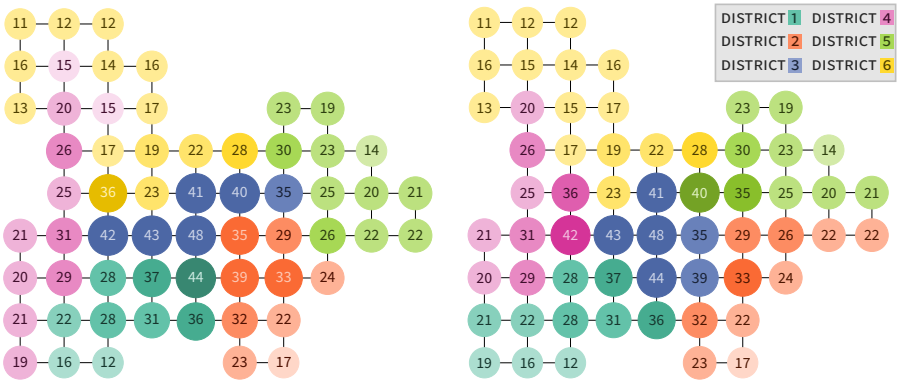
In order to test that our π' may be used to identify gerrymandered districting plans, we need such a plan. In this section, we will derive a plan which, at least on its face, plainly appears to be a gerrymander. In SECTION 4.3, we will then deploy two statistical measures, which leverage π', to reinforce this claim. In the end, we hope to demonstrate that our suspicions of a gerrymander are indeed correct.

As we know from SECTION 1.4, there are a few ingredients that are at least helpful, if not required, for a mapmaker wishing to gerrymander a state. Most obviously, one's party must be competitive in the state. As Duchin et al. [24] show, consistent VTD vote shares much below 40% make the process exceedingly difficult.

It is also helpful to have at least some form of moderate sorting within the electorate. Let us consider this with two examples for a party X which nets 48% of the statewide vote. First, imagine this 48% scales down uniformly to all

DISTRICT	POP.	CIRCLE VOTES (%)	SQUARE VOTES (%)	WINNER
1	254	124 (48.8%)	130 (51.2%)	Square
2	254	126 (49.6%)	128 (50.4%)	Square
3	*249*	*146 (58.6%)*	*103 (41.4%)*	*Circle*
4	242	117 (48.3%)	125 (51.7%)	Square
5	245	119 (48.6%)	126 (51.4%)	Square
6	256	126 (49.2%)	130 (50.8%)	Square

TABLE 4.1: District by district summary of election results for the gerrymandered plan. As indicated by column two, the total population of each district is relatively well balanced. Vote shares indicate the effects of packing and cracking: district 3 is offered to the Circle party in order to produce razor thin, but still victorious, margins in the other five districts.



(A) (B)

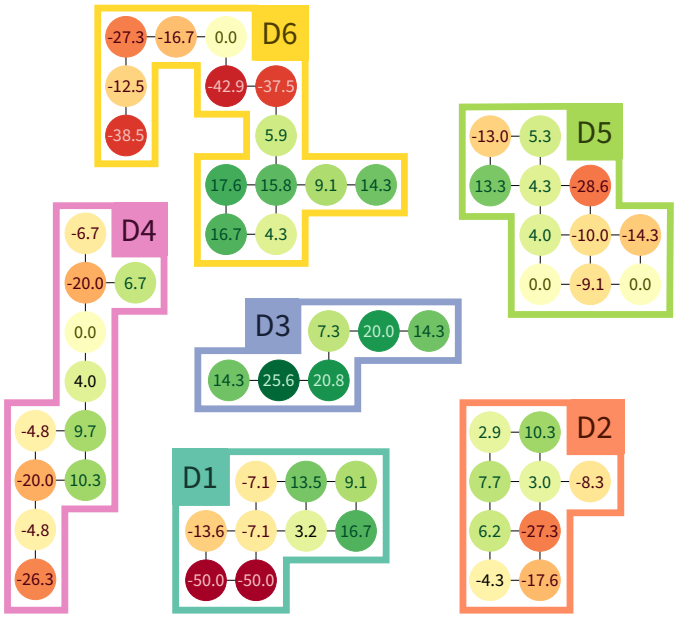
FIGURE 4.2: Side-by-side graphics of the gerrymandered plan we devised (FIG. A) and our initial plan (FIG. B), showing the total population and district assignments of each VTD. Districts 3, 4, and 6 of the gerrymandered plan are made of contorted shapes that standout in contrast to the compact equivalents in the initial plan.

VTDs. That is, party X wins 48% in every VTD and hence 48% statewide. In this scenario, no amount of cunning can produce districts which convert that 48% vote share into a higher proportion of congressional seats. Conversely, in the second scenario, where the statewide 48% is the result of summing votes from VTDs with proportions ranging from say 30% to 70%, the door to gerrymandering opens. Review of SECTION 3.2 shows that Nodeland presents us with this second reality.

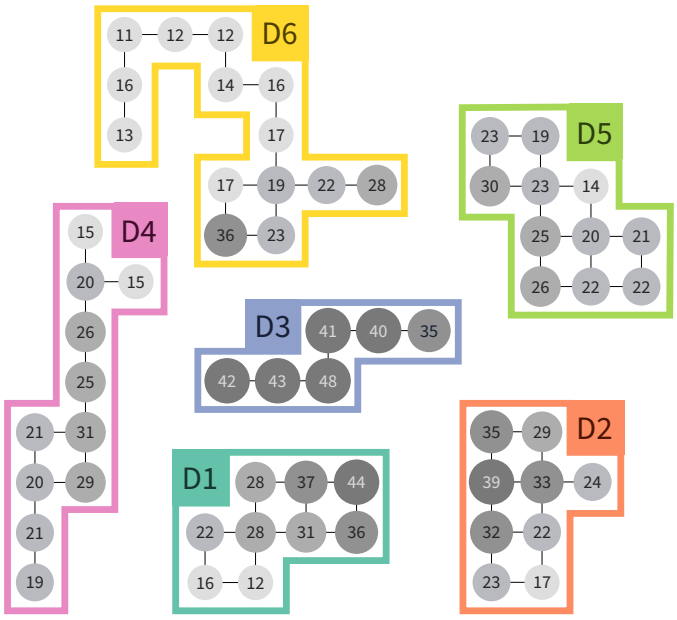
For our gerrymander, we selected the Square party as our beneficiary since they net a slim minority of the total vote share (742 votes for 49.5% overall) and gerrymandering is often considered from the perspective of the minority party. After all, it is a process most commonly deployed to convert a minority of votes into a majority of seats. To produce our gerrymandered plan, we deployed the simple but effective tools of packing and cracking (recall SECTION 1.4). In TABLE 4.1 we give a summary of key variables and in FIGURE 4.2 on this page and FIGURE 4.3 on pg. 49 we show visually how the district boundaries carve up the state.

Our strategy was first to pack together into district 3 as many VTDs with high margins for the Circle party as we could. This became the singular district sacrificed in order to win the other five for the Square party. Then, with significant effort, we devised five districts which crack the Circle vote in just the right proportions to ensure they cannot quite reach a majority in any of them. To illustrate this, FIGURE 4.3 shows the Circle margin plot once again with our gerrymandered districts overlaid.

Taken at face value, it would be reasonable to conclude that the plan we have presented here is gerrymandered. We have spent the entirety of this section referring to it as such. However, results as lopsided as winning five of six (83%) congressional seats with a minority (49.5%) of the state wide vote share is not evidence enough to *prove* a gerrymander. Would not such a scenario be feasible under circumstances of particularly beneficial geographical sorting of one's voters? While a singular and definitive means of proving a plan is gerrymandered remains elusive [18], we can assemble a ream of evidence to make the case.



(A)



(B)

FIGURE 4.3: Two diagrams show the relationships between districts and the underlying variables of their VTDs by exploding the state apart into district pieces. In FIGURE A, vote margins for the Circle party are annotated over each VTD. This gives one look at the dynamics of packing and cracking, where district 3 is packed full of high margin Circle VTDs and the other districts are effectively cracked. Looking to FIGURE B, we gain more insight. Comparing each of the two figures for district 6 shows that the Circles win with a slight margin in high population VTDs but lose big in enough of the low popluation VTDs to lose the district overall.

4.3 ANALYSING A GERRYMANDER

After spending many pages emphasising that our MCMC methodology produces an unbiased distribution of legal districting plans, we now put it to use. In this section, we will analyse our gerrymandered plan against the ensemble of plans produced with MCMC. By doing so, we will demonstrate that the gerrymandered plan is so anomalous that it cannot possibly be the result of unbiased map making. Instead, it constitutes a flagrant attempt to cheat the electoral process through devious use of political geography.

We previously elevated the notion that VTD population and vote share form the key building blocks of gerrymandering. With FIGURE 4.4, we get our first indication that something is amiss with the gerrymandered plan. Three of six districts have populations on the edge or outside the interquartile range of the distribution of 1,200 plans. This can also be said of the vote share plots of FIGURE 4.5, with district 3 sticking out as a clear outlier.

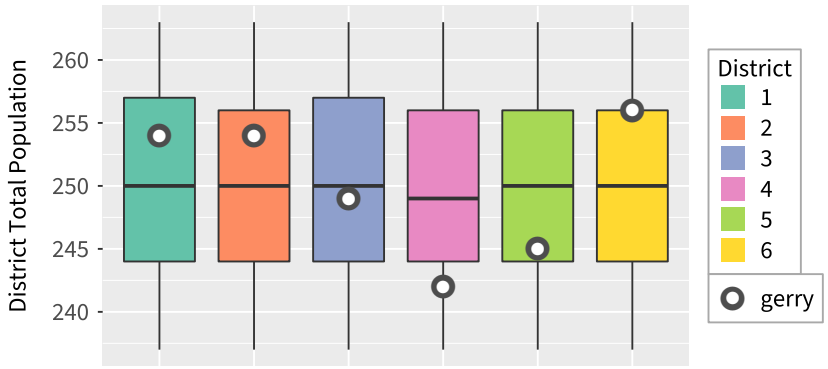


FIGURE 4.4: Boxplots of district populations across all legal plans, with values for the gerrymandered plan annotated in white. The black crossbar corresponds to the median, while the ends of the box represent the first and third quartiles (25th & 75th percentile respectively). Minimum and maximum are shown by the tails on top and bottom.

However, the clearest and most concrete indication of a biased plan is given in FIGURE 4.6. Here we see arguably the most important manifestation of our sampling distribution: how many districts each party wins under each plan. In this instance, it is unequivocal that the gerrymandered plan is an outlier. With four million iterations producing over 400,000 plans and 1,200 legal ones, MCMC returns a distribution that shows us what we already presumed. The state of Nodeland, which appears very competitive looking at vote shares, is very competitive in terms of seat allocation under hundreds of unique districting plans. In almost 800 of 1,200 cases, both parties each walk away with half the congressional seat allocation of the state. In contrast, even with those many hundreds of sampled plans, only two times does the Square party win five seats. Said differently, fully 98.7% of simulated plans produce between two and four seats for the Square party.

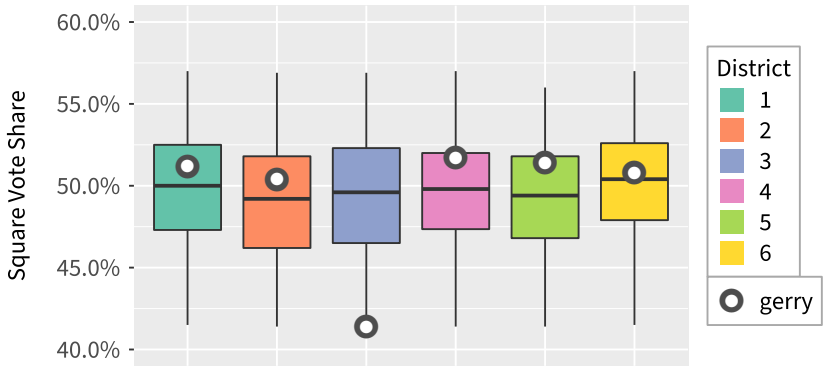


FIGURE 4.5: Boxplots of Square vote share by district across all legal plans, with values for the gerrymandered plan annotated in white. As with FIGURE 4.4, the black crossbar corresponds to the median, while the ends of the box represent the first and third quartiles (25th & 75th percentile respectively). Minimum and maximum are shown by the tails on top and bottom. We see the sacrificial (from the Square perspective) district 3 shows up as a major outlier in the gerrymandered plan.

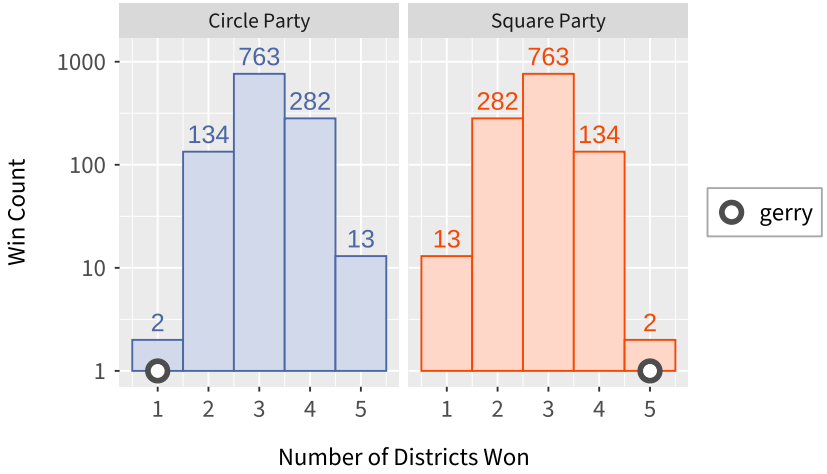


FIGURE 4.6: Histogram of the number of districts won per plan, drawn by party for the full suite of legal districting plans. The results of the gerrymandered plan are annotated in white. Note that the y-axis is drawn with a log scale. From this plot, we clearly see that by far the most common result is for each party to win three districts. Occasionally one party wins four and the other wins two. Almost never do we see the result of the gerrymandered plan where the Squares win five.

4.3.1 *Significance Tests*

In order to produce a distribution of legal districting plans, we extracted π' from our proportional sample of π derived using MCMC. This means that π' is, in effect, drawn uniformly at random from the set of all legal districting plans of Nodeland. Recall that our MCMC methodology knows nothing about the vote shares or parties of Nodeland. Hence, the distribution π' is unbiased in the context of partisanship.

NUM. DISTRICTS WON	COUNT OF PLANS
1	12
2	246
3	779
4	155
5	2

TABLE 4.2: Tabular representation of FIGURE 4.6, showing under how many plans the Square party won a given number of districts.

With this in mind, we define two hypotheses to test a given plan ϕ for partisan bias.

- H_0: The plan ϕ was drawn at random according to π'
- H_A: The plan ϕ was not drawn at random according to π'

Since we do not know the true distribution of all legal districting plans, our uniform random sample π' of this distribution will stand-in as an approximation. This is why our hypotheses are defined against π'.

In order to conduct significance testing, we require a suitable test statistic. We will orient our statistic from the perspective of the Square party, since they are the ones favoured by our gerrymandered plan. We define this as $T(\phi)$, given by

$$T(\phi) = \text{number of districts won by the Square party under plan } \phi. \tag{4.1}$$

We will test our hypotheses at the $\alpha = 0.05$ significance level. Under the gerrymandered plan, the Square party wins five districts. Hence, for our p-value, we wish to know the probability, under H_0, of an arbitrary plan producing five (or more) wins for the Square party. Looking to TABLE 4.2, we see the tabular form of FIGURE 4.6, which gives us π' from the perspective of the Square party. This table informs us of the values we need to compute our p-value.

$$\mathbb{P}\left(T(\phi) \geq 5 \mid H_0\right) = \frac{2}{1194} = 0.00168$$

This is quite clearly well below our significance level and gives us very strong evidence that we should reject our null hypothesis.

Though such a small p-value would indicate very strong support of the alternative hypothesis, how might we respond to a contrarian argument that the political geography of Nodeland simply makes it exceedingly difficult for the Squares to win five districts? One option is to construct a two-tailed hypothesis test considering *either* party winning five districts. To do this using the same test statistic, we recognise that Squares winning one district is equivalent to Circles winning five. Hence we compute our p-value.

$$\mathbb{P}\left(T(\phi) \leq 1 \cup T(\phi) \geq 5 \mid H_0\right) = \frac{14}{1194} = 0.01173.$$

Once again our p-value lies well below our significance level of 0.025 under a two-tailed test. Hence, we find confidence rejecting the null hypothesis. Given each of these tests, we can rest assured that any plan yielding five district wins for one party is well outside the window of likely outcomes for Nodeland.

4.3.2 *Relative Efficiency Gap*

While we consider the evidence from the significance tests of SECTION 4.3.1 compelling on its own, a particularly intransigent skeptic might require further convincing. Indeed, this is the case with many of the judges and justices that have reviewed numerous real-world cases on gerrymandering [7, 34, 57].

Beyond the sort of outlier detection we have done in SECTION 4.3.1, there are myriad statistics which have been proposed to quantify partisan gerrymandering. See Tapp [59] and Warrington [66] for two excellent discourses on a wide range of these measures. However, perhaps the most famous, and most scrutinised, is the *efficiency gap* as proposed by Stephanopoulos and McGhee [56]. This statistic seeks to quantify packing and cracking of voters by measuring so-called *wasted votes*. We paraphrase the definition given by Stephanopoulos and McGhee [56]:

Definition 4.1. Define the number of *wasted votes* for a given party in a given district as either:

1. The number of votes, above 50%, cast for the party winning the district, or;
2. All votes cast for the party losing the district.

Given this definition, we might consider the example where the Square party wins 56 of 100 votes for district 2. In this scenario, 6 of these Square votes are wasted while the entire 44 votes cast for the Circles are also wasted.[1] In effect, the wasted vote quantities tally up the results of all cracking and packing which occurs in a districting plan. These wasted votes then form a ratio the authors called the *efficiency gap*, which we give as adapted from McGhee [44].

Definition 4.2. For a districting plan ϕ, with wasted votes for parties A and B across all districts in ϕ given by W^A and W^B, and total vote count V^t across the entire state, define the efficiency gap EG as:

$$EG = \frac{W^A - W^B}{V^t} \tag{4.2}$$

With this definition, the efficiency gap gives a measure of possible map manipulation by either party. Namely, if the ratio is positive, then party A wasted more votes than party B and would indicate B manipulated the map to their advantage. When EG is negative, the reverse holds.

Tapp [59] points out that considering wasted vote *proportions* is better than mere absolute vote counts. In a fair districting plan, we would expect both parties to waste the same proportion of their total vote count even if the exact number of votes differs. To this end, we look to Nagle [46] for derivation of the so-called *relative efficiency gap*.

1 One would think the Square wasted count should be 5, not 6, since 51 votes are required to win. This is not, however, how the authors define wasted votes

Definition 4.3. Define the *relative efficiency gap* (*REG*) as,

$$REG = \frac{W^A}{V^A} - \frac{W^B}{V^B}, \tag{4.3}$$

where W^A and W^B are wasted votes[2] and V^A and V^B are the total number of votes cast for parties A and B respectively.

Using DEF. 4.3, we computed *REG* values for every plan produced by our MCMC algorithm and the gerrymandered plan presented in SECTION 4.2. The results are given in FIGURE 4.7. As is immediately apparent from this plot, the relative efficiency gap tells exactly the same story as the significance tests presented in SECTION 4.3.1: the gerrymandered plan is clearly manipulated to favour the Square party.

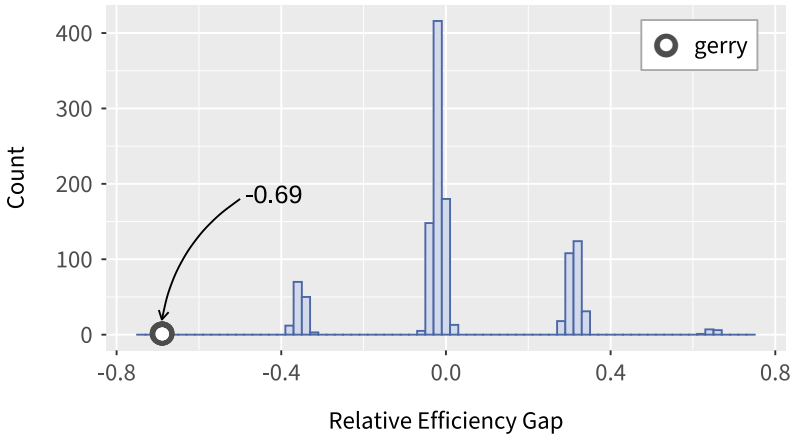


FIGURE 4.7: Histogram of the relatively efficiency gap (REG) for all plans in π'. The gerrymandered plan is annotated in white. Each of the five discrete clusters correspond to the different ratios of districts won. From left to right with Square wins followed by Circle wins these are: 5:1, 4:2, 3:3, 2:4, 1:5. We can immediately see that the gerrymandered plan lands in the minuscule tails of this distribution.

This is shown by the fact that the gerrymandered plan lands in the tails of the distribution with an exceedingly anomalous value. The fair plans which split the districts evenly between parties at 3:3 show up in the centre with a near-zero *REG*, and each increasingly lopsided ratio (4:2 and 5:1) of district wins produces a more extreme *REG* value. By the fact that the values of *REG* mimic the distribution of FIGURE 4.5, we gain reinforcing evidence of the gerrymander claim.

With each of these analyses, significance tests and relative efficiency gap, we have shown the anomalous nature of our gerrymandered plan. Though it is hard to argue any collection of statistical evidence is incontrovertible, we do believe the results presented here are compelling. However, additional diagnostic tools are available and will be discussed in CHAPTER 5.

2 Computation of wasted votes varies slightly from that given in DEF. 4.1 ; see Tapp [59, pgs. 602-603] for a complete definition of the computation. We used $\lambda = 2$ in our computation.

4.4 MIXING

Before closing out this chapter on results, we believe a brief presentation and discussion of the mixing of our redistricting Markov chain is warranted. Mixing can be described as the efficacy with which the chain explores the entire state space. A chain which is mixing poorly may remain localised, repeatedly visiting the same states and never reaching others. Conversely, a chain which mixes well is one that successfully visits states in the chain proportionally to their density in the target distribution.

We do not have any direct measurements of the mixing quality of our chain. However, we do recognise some empirical evidence that our chain mixes well. The plans given in FIGURE 4.1 show that district assignments meander throughout Nodeland. That is, while district 5, for example, starts in the top left (recall FIGURE 3.2), it does not remain local to that area. This points to the fact the chain, at least anecdotally, appears to be effectively exploring the state space.

Another more practical consideration where mixing is concerned is the number of iterations required to successfully approximate the target distribution. This is the mixing time, or in other words, how quickly it converges. On this front, we find our methodology lacking. This poor performance comes down to the way in which we make proposals and the fact that they are routinely not accepted.

The reader will recall that at the top of the chapter we gave one indicator of this poor performance: out of four million iterations, 412,774 plans were accepted subject to our defined acceptance ratio. Taken on instinct alone, this ratio of 1:10 accepted plans per iteration does not seem particularly good. However, a couple subtleties hide behind these headline numbers.

The first is that we record an iteration every time a plan is proposed but we only push new plans forward to the acceptance calculation. Let us illustrate what is meant by this. By new plan, we mean a plan ϕ' which produces districting assignments that differ from the current plan ϕ. Said specifically, this means $T_{\phi,\phi'} \neq 0$. The reader will recall from SECTIONS 3.4 and 3.5 that under our proposal mechanism, it is possible to pick a source node which is surrounded entirely by nodes of its current district and hence produce a proposal plan ϕ' which offers the exact same district assignments as ϕ. Under this scenario, we define $T_{\phi,\phi'} = 0$ (see DEF. 3.3). In handling it, we deem any ϕ' of this form as a 'repeat' and, as a matter of computational economy, elect to conclude the iteration there at the proposal stage and proceed to the next iteration.

Since the acceptance ratio dictates these repeat plans will always be rejected ($T_{\phi,\phi'} = 0$), we save ourselves some overhead in bypassing them. Looking to FIGURE 4.8, we see that a sizeable majority of iterations are spent producing such repeat plans. This makes sense given that districts commonly have many 'interior' nodes which share no neighbours from other districts.

The second component driving the aforementioned 1:10 ratio is the rejection rate of the algorithm. Looking once again to FIGURE 4.8, we see that 68.6% (903,715 of 1,316,488) of non-repeated plans are rejected. This speaks directly to the balancing act we alluded to in CHAPTER 3. Namely, there must be some elasticity in the range of population scores which can be accepted in order for the chain to adequately explore the state space. Said differently, transitioning to a plan with districts of balanced population requires transiting through plans

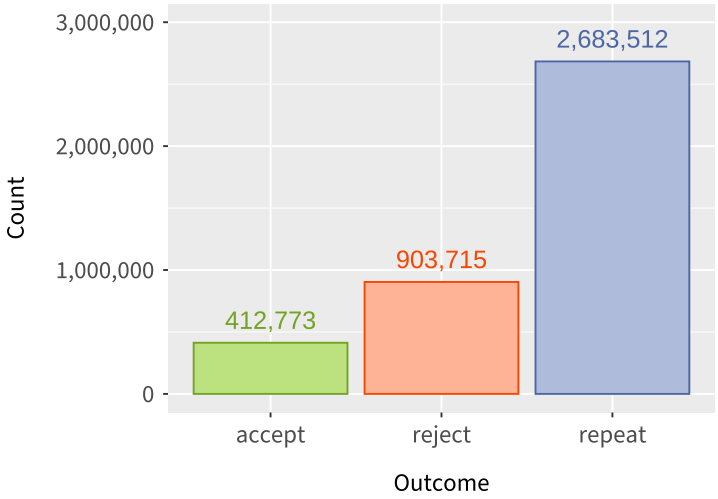


FIGURE 4.8: Histogram of outcomes for all four million iterations of our simulation. Repeats describe proposal plans which are exactly equal to the current plan. Rejects are those plans which fail the acceptance test and accepted plans are those which pass this test.

with significant imbalance. As we see, this results in almost 70% of non-repeat plans being rejected.

While one could tweak the population sigma function, or the normalising constant embedded in the score function, it would almost certainly be more efficient to alter the proposal mechanism. The reader will note that we described and employed the flip proposal structure but made no mention of other types of proposal. We will address this, and a number of other considerations, in our coming final chapter.

5 FURTHER CONSIDERATIONS

Before we conclude our lengthy discourse on congressional redistricting, Markov chain Monte Carlo and partisan gerrymandering, we will spend a few pages looking to the considerations we did not tackle. It is rare that any work of this duration reaches its deadline feeling completely explored. Many questions get posed but left unanswered and ideas are swept aside because they do not suit the time allocated. This work is no different. While we have covered many topics in great detail, there are a number of nuances, new ideas, and improvements we might have included.

To this end, this chapter will consider four areas that merit further discussion. In section 5.1, we will look at some of the variables and legal constraints not considered by our dataset. With section 5.2, we will look at an alternative means of proposing new districting plans and what that implies. Extending this discussion of methodology, section 5.3 will consider additional parameters that could be used in scoring. Turning toward analysis tools, we will investigate further statistics for measuring gerrymandering in section 5.4. At the end, we will offer our conclusion.

5.1 OUR DATASET VS. REALITY

Perhaps the most obvious area in which this work could be advanced is through the use of a real-world dataset. While there was tangible value in leveraging a synthetic dataset, by making concepts approachable and interaction between variables simpler, congressional redistricting occurs in the real world. There are three arenas that earn our consideration: vote shares, communities of interest, and the implications of the Voting Rights Act (VRA).

5.1.1 Vote Shares

In our dataset, we treated vote shares as fixed values that are not impacted by the passage of time. Similarly, we make no attempt to discriminate between candidates. However, real elections are subject to many different dynamics. To start, House elections occur every two years while the presidential election occurs every four. As has long been established (see Wolfinger, Rosenstone and McIntosh [70]), this can result in a substantially different electorate in the non-presidential[1] years. The consequence being that a party's vote share can swing significantly between presidential and midterm elections.

Extending this complication is the question: which vote share result does one use in one's model? As the seminal paper by Chen and Rodden [17], as well as others (see Bangia et al. [4], Best et al. [8] and Herschlag, Ravier and Mattingly [33]) has shown, a common choice is to use vote shares from the

1 Often referred to as the *midterms*

most recent presidential election.[2] This provides values for the vote share variable across all the VTDs of the state. In turn, it is then easier to approximate, for example, the electoral outcome of creating a new district by fusing together pieces of two current districts. However, as Best et al. [8] specifically addresses, it does not take into account the dynamics of incumbents—or otherwise popular local politicians—who may be of the opposite party to the presidential candidate that won their district. This concept, where an electorate votes for one party at one level of the representative hierarchy and for a different party at another level is called *ticket splitting*.

In addition to ticket splitting, a competitive state can produce surprisingly volatile results. The state of Wisconsin is illustrative on this point. In 2010, which was a midterm year, the state elected Republican Governor Scott Walker. Conversely, in 2012, the state reelected President Barack Obama, and elected Democratic Senator Tammy Baldwin to her first term. Judged by the DW-NOMINATE[3] statistic, she ranks as one of the top 10 most liberal/progressive members of the Senate. In 2016, President Donald Trump and Republican Senator Ron Johnson were narrowly elected by the state. This trend of volatility continued in 2018 when Democratic Governor Tony Evers ousted Governor Scott Walker and Baldwin won reelection.

The Wisconsin example serves to illustrate how competitive states can be hard to pin down. That is, with so much volatility present at the statewide level, do statewide vote shares adequately model the disposition of the state when it is broken down into districts by a prospective plan? In this arena, Best et al. [8] is instructive in showing that the presidential vote share at least forms a decent proxy. There is also the fact that no better options are readily apparent. All of which serves to illustrate that while our synthetic dataset overlooked these dynamics, it also simplified the model and allowed us to maintain focus on the mathematical fundamentals. These are, after all, essential to get correct.

5.1.2 *Communities of Interest*

Another practical consideration we avoided in our synthetic dataset was the notion of municipalities or *communities of interest*. As Powell, Clark and Dube [53] point out, some states require that redistricting maps honour existing community boundaries when drawn. This adds yet another layer of constraints to the list one must consider, either in simulating proposals for new districting plans, scoring those plans, or when assessing their legality. In our synthetic case, we were freed from such burdens. We did not model any communities or municipalities since we chose VTDs as our smallest unit. However, were we to turn our algorithm to a real dataset, adjustments to handle communities of interest might be required.

2 Some also use Senate elections

3 See Poole and Rosenthal [52] for more details on this multidimensional scaling measure of ideology

5.1.3 *Voting Rights Act*

Given that we have not mentioned it since SECTION 1.3, the reader would be forgiven for not remembering the specifics of the Voting Rights Act (VRA). As a reminder, when it comes to redistricting, the implications of the act mean that districts cannot be drawn to dilute or obstruct the voting power of racial minorities. That is, *racial gerrymandering* is illegal in ways that partisan gerrymandering is not. The flip side of this ban is that some states are effectively required to produce majority-minority districts in order to satisfy the VRA.

These districts pose a fiendish challenge. As Chen and Cottrell [16] point out, these VRA districts often end up producing partisan makeups that bias in favour of one party (Democrats) in a state or locality that would otherwise tend to favour the other (Republicans). This is of course the whole reason behind the mandate, but it does introduce an extra layer of bias into the redistricting process. One way to handle this, as did Cho and Liu [18], is by treating the majority-minority districts as fixed and use MCMC to draw all the rest. Another option, per Chen and Cottrell [16], is effectively to ignore the racial component and then adjust the analysis to account for the majority-minority districts. Perhaps most promising is the work of Powell, Clark and Dube [53], which utilises an algorithm that accounts for the requirements of the VRA. As with SECTION 5.1.2, our algorithm would almost certainly be ill-equipped to handle these concerns without significant new adaptations.

5.2 PROPOSALS

In SECTION 4.4, we looked more closely at the operational behaviour of our Markov chain as it relates to convergence and mixing. The most consequential lesson from that examination was that the flip proposal method we have employed is slow to converge. This is seen most directly in the proportion of samples which constitute 'repeats'. That is, approximately 67% of the time, a proposal for iteration i produces a plan ϕ' which is exactly the same as the current plan ϕ. This raises the question: is there a proposal mechanism that converges more quickly?

In light of the pernicious and persistent presence of gerrymandering in American democracy, the last decade has seen many academics and mathematicians seeking to 'solve' the gerrymandering problem. Though some make use of alternative methods like optimisation or assembly (see DeFord, Duchin and Solomon [22, pg. 4] for a short summary), many use the Markov chain random walk framework we have presented here. In our estimation, most of them, including but not limited to Bangia et al. [4], Chen and Rodden [17], Herschlag et al. [32], Herschlag, Ravier and Mattingly [33] and Liu, Cho and Wang [41], use the flip proposal method in some form or fashion. While Fifield et al. [27] presents an advancement using a so called 'graph-cut' framework, we find that the even more novel approach of DeFord, Duchin and Solomon [22] offers the most encouraging methodology to tackle the failings of flip.

5.2.1 *Recombination*

In stark contrast to the node-by-node proposal system we presented in our work, DeFord, Duchin and Solomon [22] use a combine-and-split methodology which they call *recombination*. The basic intuition considers districts as a series of subgraphs which are merged together and then split back apart to yield a new districting plan, or so called *partitioning* of the state. It proceeds in the following fashion.

Start with a dual graph[4] G which represents the state in question, and is partitioned in P districts. Then select 2 districts from P and merge them into an intermediate district W. Deploy a spanning tree algorithm[5] on W, which yields a subgraph of all the nodes of W connected by as few edges as possible. Analyse the edges of this subgraph to find the edge which, when cut, produces two districts with the best population balance, subject to constraints. Hence, a new districting plan is produced.

The advantages of this methodology are immediately apparent. Looking to FIGURE 5.1, we see a schematic of a single iteration of recombination. Considering this proposal against our method, DeFord, Duchin and Solomon [22] produce in one iteration what would take ours a *minimum* of four iterations in the ideal case. As the authors demonstrate in their paper, the convergence of the chain under recombination offers orders of magnitude improvement when measured against iteration count. They also find that this new method offers more robust qualitative results when it comes to satisfying legal requirements. We discuss these within the context of scoring in our next section.

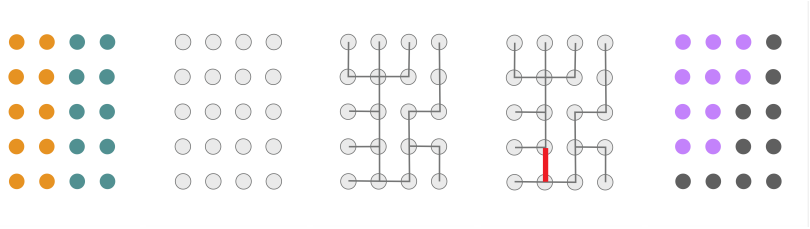


FIGURE 5.1: A schematic representation of recombination, taken directly from the paper by DeFord, Duchin and Solomon [22]. Step A gives the starting pair of districts which are then merged in step B. Step C is the subgraph derived using the spanning tree algorithm. Step D shows the edge selected to be cut and E shows the two new districts which result.

4 Dual graph is the author's choice of term for a graph made of nodes and edges, as we have seen throughout this work

5 Spanning Tree offers an approachable definition and example

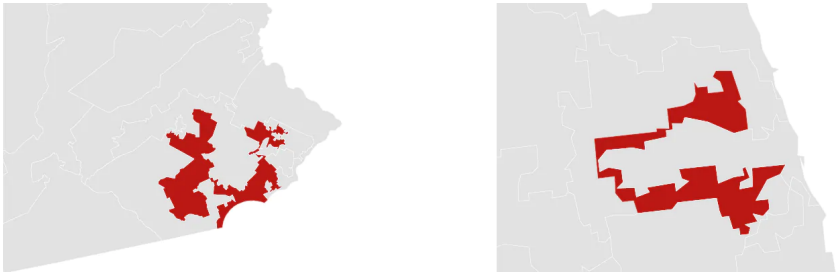
5.3 SCORING

As we addressed in SECTION 3.6, we define the target distribution for our Markov chain by modelling districting constraints in a series of sub-score functions. In our work, we modelled requirements of contiguity and population balance between districts, but these are not the only factors that one might consider. Other common, and more qualitative, conditions include metrics known as *compactness* and *competitiveness*.

5.3.1 *Compactness*

Compactness looks to assess the regularity of a district's shape. That is, a spindly and contorted district with lots of concavities, like that of the original gerrymander (recall SECTION 1.3), would score very poorly on compactness. Conversely, districts with good compactness form convex polygonal shapes that 'seem' appropriate. This qualitative language is indicative of the way compactness is treated in reality, where it is usually assessed with the 'eyeball test' [22].

The intuition of compactness is this: if one is drawing a district so contrived that it has bad compactness, then there were probably external factors influencing that district. Said differently, if one were drawing unbiased maps, one would very rarely have any good reason to draw districts like those presented in FIGURE 5.2. Considering compactness as a measure of potential bias has led to a wealth of research looking at it as a way to measure gerrymandering [5, 34, 37, 48, 51, 71].



(A) (B)

FIGURE 5.2: Two real districts with very poor compactness. FIGURE A shows Pennsylvania's former 7th district, which was forced to be redrawn by the state Supreme court in 2018. FIGURE B shows Illinois's current fourth district. These images were taken directly from Ingraham [36].

To assess compactness, a variety of measures have been proposed. The commonly cited and deployed ones include the area/perimeter ratio of Polsby and Popper [51], the Roeck area/bounding circle ratio given by Young [71], and the convex hull measure from Niemi et al. [48]. There are concerns with each of the measures and also with their implementation. As Barnes and Solomon [5] summarises, handling of districts with holes, legal non-contiguity (resulting from islands, say), and even map projections can result in radically different results using the same metric.

Considering compactness within the scoring function adds additional constraints to which kinds of plans get accepted. From the perspective of our Mar-

kov chain, this is liable to *impair* mixing rather than improve it. In this regard, we find more reinforcement for the recombination methodology, which DeFord, Duchin and Solomon [22] show natively produces more compact districts. Similarly, the woolly and qualitative assessments of compactness (i.e. the aforementioned 'eyeball test') further deterred us from looking to implement a compactness measure. If the goal of simulating plans is to identify gerrymanders, better measures than compactness exist, as we saw in CHAPTER 4 and will examine in SECTION 5.4. If the goal is to identify a robust districting plan to implement, then eye-balling the best looking plan of an ensemble of samples seems entirely adequate.

5.3.2 *Competitiveness*

While most, if not all, states have requirements for contiguous, population-balanced, community-preserving, and compact districts, some enforce additional constraints. By our estimation, four states – Arizona, Colorado, New York and Washington – also have provisions for competitiveness on the books (see National Conference of State Legislatures [47]). This measure looks to balance the partisan make up of districts, in addition to balancing population.

With the exception of Arizona, each of these states passed legislation adding competitiveness to their districting guidelines since the last round of redistricting in 2011 [47]. As a result, it does not yet appear to show up as a topic of interest in the research realm. That said, it is possible that with more states considering passing the responsibility of redistricting off to independent commissions, it will gain momentum. For our purposes, we see no particular added value in considering competition in our scoring function. This is in part a result of the fact that we designed our synthetic dataset to be very competitive. Moreover, we are not sure that inducing competition through boundary drawing is the wisest idea in a democracy.

5.4 GERRYMANDERING METRICS

Our final item for discussion concerns the analysis of districts suspected of gerrymandering. In SECTION 4.3, we presented the efficiency gap metric that gives an indication of the amount of packing and cracking present in a districting plan. Since it was proposed by Stephanopoulos and McGhee [56], the efficiency gap has seen its stature raised through argument in the halls of numerous state Supreme Courts and even the venerable U.S. Supreme Court. This has also resulted in criticism from the likes of Katz, King and Rosenblatt [37], Nagle [46], Tapp [59] and Warrington [66], among others.

For our purposes, we believe that deploying an ensemble of metrics is the wisest course of action for analysis. As clearly evidenced by the number of decades the issue has plagued the courts and the national discourse, the nature of gerrymandering is fluid and difficult to concretely prove. As is routine practice in the court of law, we believe it better to provide a wealth of evidence that all points in the same direction. To this end, we see particular value in the relative efficiency gap of Nagle [46], as used in this document, as well as the intriguing declination measure derived by Warrington [65]. We propose that combining

quantitative assessments like these with analysis using simulated, unbiased distributions of plans forms a suitable toolbox for detecting and demonstrating partisan gerrymanders.

5.5 CONCLUSION

With this work, we have sought to answer the question: how can one identify a partisan gerrymander? We began our extended discourse on this subject by outlining some of the basic tenets and laws governing democracy and redistricting in the United States. After establishing the ways in which gerrymandering occurs and how it can be used to manipulate an election, we turned to mathematics.

With Markov chain Monte Carlo, and more specifically, the Metropolis–Hastings algorithm, we presented a tool to simulate sample districting plans. We derived a code base from scratch, written in `python`, and a synthetic dataset to pair with it to produce these samples subject to a proposal distribution and suitable acceptance probability. From the resulting distribution, we extracted the plans which satisfied all legal constraints and called this our distribution of interest.

From our synthetic dataset, we devised a gerrymander with a severely lopsided result. Using our sampled distribution, we showed through the use of significance tests and the relative efficiency gap statistic that the plan was indeed a partisan gerrymander. We finished by examining areas of improvement and nuance that would warrant further exploration in any future work.

In the end, we conclude that mathematics and modern advances in computational power offer a powerful tonic to the affliction of gerrymandering. It is not clear that any talisman exists to concretely prove that partisan gerrymandering was the intent of a given map. However, we do believe it is achievable to compile enough evidence to be convinced of serious bias. It is not clear that the political and legal systems of the United States are primed to tackle the problem head-on, but we do believe the academic rigour exists to meet the challenge.

BIBLIOGRAPHY

[1] 71st Congress of the United States. *2 U.S. Code § 2a - Reapportionment of Representatives; Time and Manner; Existing Decennial Census Figures as Basis; Statement by President; Duty of Clerk*. [Online]. June 1929. URL: https://www.law.cornell.edu/uscode (visited on 01/07/2020).

[2] 89th Congress of the United States. *52 U.S. Code § 10301 - Denial or Abridgement of Right to Vote on Account of Race or Color through Voting Qualifications or Prerequisites; Establishment of Violation*. [Online]. August, 1965. URL: https://www.law.cornell.edu/uscode (visited on 14/07/2020).

[3] 90th Congress of the United States. *2 U.S. Code § 2c - Number of Congressional Districts; Number of Representatives from Each District*. [Online]. Dec 1967. URL: https://www.law.cornell.edu/uscode (visited on 01/07/2020).

[4] S. Bangia, C. V. Graves, G. Herschlag, H. S. Kang, J. Luo, J. C. Mattingly and R. Ravier. 'Redistricting: Drawing the Line'. In: *arXiv: Statistics - Applications* ([Online] may 2017).

[5] R. Barnes and J. Solomon. 'Gerrymandering and Compactness: Implementation Flexibility and Abuse'. In: *arXiv: Computer Science - Computers and Society* ([Online] March 2018).

[6] R. Barnes. 'Supreme Court Says Federal Courts Don't Have a Role in Deciding Partisan Gerrymandering Claims'. In: *Washington Post* ([Online] June 2019). URL: https://www.washingtonpost.com/politics (visited on 02/09/2020).

[7] M. Bernstein and M. Duchin. 'A Formula Goes to Court: Partisan Gerrymandering and the Efficiency Gap'. In: *arXiv: Physics - Physics and Society* ([Online] May 2017).

[8] R. E. Best, J. S. Krasno, D. B. Magleby and M. D. McDonald. 'Detecting Florida's Gerrymander: A Lesson in Putting First Things First'. In: *Social Science Quarterly* 101.1 (2020), pp. 37–52.

[9] H. Black. *Wesberry v. Sanders, 376 U.S. 1*. [Online] Feb 1964. URL: https://supreme.justia.com/cases (visited on 01/07/2020).

[10] Boston Gazette. *Cartoon, "The Gerry-Mander", 1813*. Apr. 1813. URL: https://americanhistory.si.edu/collections/search/object/nmah_509530 (visited on 15/07/2020).

[11] W. J. Brennan. *Baker v. Carr 369 U.S. 186*. [Online] March 1962. URL: https://supreme.justia.com/cases (visited on 01/07/2020).

[12] W. J. Brennan. *Thornburg v. Gingles, 478 U.S. 30*. [Online] June 1986. URL: https://supreme.justia.com/cases (visited on 14/07/2020).

[13] Brennan Center for Justice. *Who Draws the Maps? Legislative and Congressional Redistricting*. [Online] Jan, 2019. URL: https://www.brennancenter.org/our-work (visited on 24/06/2020).

[14] R. Buck and M. Hully. *Wisconsin Shapefiles*. Metric Geometry and Gerrymandering Group, May 2019. URL: https://github.com/mggg-states/WI-shapefiles (visited on 02/09/2020).

[15] K. D. Burnett. *Congressional Apportionment*. Briefing. United States Census Bureau, [Online] Nov 2011. URL: https://www.census.gov/prod/cen2010 (visited on 02/09/2020).

[16] J. Chen and D. Cottrell. 'Evaluating Partisan Gains from Congressional Gerrymandering: Using Computer Simulations to Estimate the Effect of Gerrymandering in the U.S. House'. In: *Electoral Studies* 44 (Dec. 2016), pp. 329–340.

[17] J. Chen and J. Rodden. 'Cutting Through the Thicket: Redistricting Simulations and the Detection of Partisan Gerrymanders'. In: *Election Law Journal: Rules, Politics, and Policy* 14.4 (Dec. 2015), pp. 331–345.

[18] W. K. T. Cho and Y. Y. Liu. 'Toward a Talismanic Redistricting Tool: A Computational Method for Identifying Extreme Redistricting Plans'. In: *Election Law Journal* 15.4 (1st Dec. 2016), pp. 351–366.

[19] H. Christian and P. H. *remappedColorMap*. [Online] Aug 2014. URL: https://github.com/TheChymera/chr-helpers.

[20] G. W. Cox and J. N. Katz. *Elbridge Gerry's Salamander: The Electoral Consequences of the Reapportionment Revolution*. Cambridge: Cambridge University Press, 2002.

[21] D. DeFord. *Introduction to Discrete MCMC for Redistricting*. [Online] 06/2019. URL: https://people.csail.mit.edu/ddeford/MCMC_Intro.pdf.

[22] D. DeFord, M. Duchin and J. Solomon. 'Recombination: A Family of Markov Chains for Redistricting'. In: *arXiv: Computer Science - Computers and Society* ([Online] Oct 2019).

[23] T. Downey. *Why 435?* [Online] Jan 2020. URL: https://democracyjournal.org/magazine/55/why-435/ (visited on 01/07/2020).

[24] M. Duchin, T. Gladkova, E. Henninger-Voss, B. Klingensmith, H. Newman and H. Wheelen. 'Locating the Representational Baseline: Republicans in Massachusetts'. In: *Election Law Journal: Rules, Politics, and Policy* 18.4 (Dec. 2019), pp. 388–401.

[25] J. S. Ellenberg. 'Geometry, Inference, Complexity, and Democracy'. In: *arXiv: Physics - Physics and Society* ([Online] June 2020).

[26] E. J. Engstrom. *Partisan Gerrymandering and the Construction of American Democracy*. Legislative Politics and Policy Making. Ann Arbor: University of Michigan Press, 2013. 227 pp.

[27] B. Fifield, M. Higgins, K. Imai and A. Tarr. 'Automated Redistricting Simulation Using Markov Chain Monte Carlo'. In: *Journal of Computational and Graphical Statistics* 0.0 (Mar. 2020), pp. 1–14.

[28] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. 3rd ed. Oxford ; New York: Oxford University Press, 2001. 596 pp.

[29] C. M. Grinstead and J. L. Snell. *Introduction to Probability*. 2., rev. ed., reprinted with corr. Providence, RI: American Mathematical Society, 1998. 510 pp.

[30] A. A. Hagberg, D. A. Schult and P. J. Swart. 'Exploring Network Structure, Dynamics, and Function Using NetworkX'. In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught and J. Millman. Pasadena, CA USA, 2008, pp. 11–15.

[31] G. W. Headley. *Source Code for Congressional Redistricting in the United States*. [Online] Sept 2020. URL: https://github.com/HeadCase/dissertation.

[32] G. Herschlag, H. S. Kang, J. Luo, C. V. Graves, S. Bangia, R. Ravier and J. C. Mattingly. 'Quantifying Gerrymandering in North Carolina'. In: *arXiv: Physics - Physics and Society* ([Online] Jan 2018).

[33] G. Herschlag, R. Ravier and J. C. Mattingly. 'Evaluating Partisan Gerrymandering in Wisconsin'. In: *arXiv: Statistics - Applications* ([Online] Sept 2017).

[34] M. Humphreys. 'Can Compactness Constrain the Gerrymander?' In: *Irish Political Studies* 26.4 (1st Dec. 2011), pp. 513–520.

[35] J. D. Hunter. 'Matplotlib: A 2D Graphics Environment'. In: *Computing in Science Engineering* 9.3 (May 2007), pp. 90–95.

[36] C. Ingraham. 'America's Most Gerrymandered Congressional Districts'. In: *Washington Post* ([Online] May 2014). URL: https://www.washingtonpost.com/news (visited on 01/07/2020).

[37] J. N. Katz, G. King and E. Rosenblatt. 'Theoretical Foundations and Empirical Evaluations of Partisan Fairness in District-Based Democracies'. In: *American Political Science Review* 114.1 (Feb. 2020), pp. 164–178.

[38] A. Kennedy. *Bartlett v. Strickland, 556 U.S. 1*. [Online] Mar 2009. URL: https://supreme.justia.com/cases (visited on 14/07/2020).

[39] D. P. Kroese, T. Brereton, T. Taimre and Z. I. Botev. 'Why the Monte Carlo Method Is so Important Today'. In: *WIREs Computational Statistics* 6.6 (2014), pp. 386–392.

[40] D. A. Levin, Y. Peres and E. L. Wilmer. *Markov Chains and Mixing Times*. Providence, R.I, USA: American Mathematical Society, 2009. 371 pp.

[41] Y. Y. Liu, W. K. T. Cho and S. Wang. 'PEAR: A Massively Parallel Evolutionary Computation Approach for Political Redistricting Optimization and Analysis'. In: *Swarm and Evolutionary Computation* 30 (1st Oct. 2016), pp. 78–92.

[42] H. Maltby, W. Pakornrat and J. Jackson. *Markov Chains*. URL: https://brilliant.org/wiki/markov-chains/ (visited on 20/07/2020).

[43] K. C. Martis. 'The Original Gerrymander'. In: *Political Geography* 27.8 (Nov. 2008), pp. 833–839.

[44] E. McGhee. 'Measuring Partisan Bias in Single-Member District Electoral Systems'. In: *Legislative Studies Quarterly* 39.1 (Jan. 2014), pp. 55–85.

[45] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller. 'Equation of State Calculations by Fast Computing Machines'. In: *The Journal of Chemical Physics* 21.6 (June 1953), pp. 1087–1092.

[46] J. F. Nagle. 'How Competitive Should a Fair Single Member Districting Plan Be?' In: *Election Law Journal: Rules, Politics, and Policy* 16.1 (Jan. 2017), pp. 196–209.

[47] National Conference of State Legislatures. *Redistricting Criteria*. Apr. 2019. URL: https://www.ncsl.org/research/ (visited on 28/08/2020).

[48] R. G. Niemi, B. Grofman, C. Carlucci and T. Hofeller. 'Measuring Compactness and the Role of a Compactness Standard in a Test for Partisan and Racial Gerrymandering'. In: *The Journal of Politics* 52.4 (Nov. 1990), pp. 1155–1181.

[49] J. R. Norris. *Markov Chains*. Cambridge: Cambridge University Press, 1997.

[50] Philadelphia Convention. *The Constitution of the United States: A Transcription*. Sept. 1787. URL: https://www.archives.gov/founding-docs/constitution-transcript (visited on 23/06/2020).

[51] D. D. Polsby and R. Popper. 'The Third Criterion: Compactness as a Procedural Safeguard Against Partisan Gerrymandering'. In: *Yale Law and Policy Review* 9.9 (Mar. 1991).

[52] K. T. Poole and H. Rosenthal. 'A Spatial Model for Legislative Roll Call Analysis'. In: *American Journal of Political Science* 29.2 (May 1985), pp. 357–384.

[53] R. J. Powell, J. T. Clark and M. P. Dube. 'Partisan Gerrymandering, Clustering, or Both? A New Approach to a Persistent Question'. In: *Election Law Journal: Rules, Politics, and Policy* 19.1 (Feb. 2020), pp. 79–100.

[54] C. P. Robert. 'The Metropolis-Hastings Algorithm'. In: *arXiv: Statistics - Computation* ([Online] January 2016).

[55] R. Serfozo. *Basics of Applied Stochastic Processes*. Berlin: Springer, 2009. 443 pp.

[56] N. Stephanopoulos and E. McGhee. 'Partisan Gerrymandering and the Efficiency Gap'. In: *University of Chicago Law Review* 82.2 (Mar. 2015).

[57] N. Stephanopoulos and E. McGhee. 'The Measure of a Metric: The Debate Over Quantifying Partisan Gerrymandering'. In: *Stanford Law Review* 70.1503 (May 2018).

[58] Y. Taniguchi. *Understanding the Breadth-First Search with Python*. Feb. 2019. URL: https://medium.com/@yasufumy/ (visited on 13/07/2020).

[59] K. Tapp. 'Measuring Political Gerrymandering'. In: *The American Mathematical Monthly* 126.7 (Aug. 2019), pp. 593–609.

[60] The Editorial Board. 'Politicians Can Pick Their Voters, Thanks to the Supreme Court'. In: *The New York Times. Opinion* ([Online] June 2019). URL: https://www.nytimes.com/2019 (visited on 02/09/2020).

[61] *The Historical Atlas of United States Congressional Districts, 1789-1983*. In collab. with K. C. Martis, C. L. Lord, R. A. Rowles and N. Historical Records Survey (New York. New York : London, 1982.

[62] The pandas development team. *Pandas-Dev/Pandas: Pandas 1.1.0*. Version latest. Feb. 2020.

[63] U.S. Government. *Branches of the U.S. Government*. Oct. 2019. URL: https://www.usa.gov/about-the-us (visited on 02/09/2020).

[64] P. Virtanen et al. 'SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17.3 (Mar. 2020), pp. 261–272.

[65] G. S. Warrington. 'Quantifying Gerrymandering Using the Vote Distribution'. In: *Election Law Journal: Rules, Politics, and Policy* 17.1 (Mar. 2018), pp. 39–57.

[66] G. S. Warrington. 'A Comparison of Partisan-Gerrymandering Measures'. In: *arXiv: Physics - Physics and Society* ([Online] April 2019).

[67] R. Weber. *Markov Chains*. Sept. 2012. URL: http://www.statslab.cam.ac.uk/~rrw1/markov/M.pdf.

[68] L. P. Whitaker. *Congressional Redistricting and the Voting Rights Act: A Legal Overview*. Report. Washington, D. C.: Congressional Research Service, [Online] Feb 2014. URL: https://digital.library.unt.edu/ark:/67531/metadc462616/ (visited on 14/07/2020).

[69] H. Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Springer, 2016.

[70] R. E. Wolfinger, S. J. Rosenstone and R. A. McIntosh. 'Presidential and Congressional Voters Compared'. In: *American Politics Quarterly* 9.2 (Apr. 1981), pp. 245–256.

[71] H. P. Young. 'Measuring the Compactness of Legislative Districts'. In: *Legislative Studies Quarterly* 13.1 (Feb. 1988), pp. 105–115.

A

SOURCE CODE

This appendix contains a curated selection of the source code used in this work. Included are the essential functions used to execute the redistricting Markov chain. Absent are the many hundreds of lines of code used to aid this process, including initialisation, logging, plotting, and design. For the complete code base, including both `python` and `R` source files—as well as supplementary support files—please see the aforementioned GitHub page: https://github.com/HeadCase/dissertation.

A.1 MAIN

```
############################################################
###################       main.py      #####################
############################################################
#!/usr/bin/env python
""" Main function """

# Imports
import sys

from init import init_expanded
from chain import chain
from reject import reject_islands
from reject import reject_by_pop
from utils import remove_dups
from election import election
from log import to_json


def main():
    """ Main function """

    # Set basename for the run and initialise the first
    # graph with the starting plan
    basename =
        "4mil-const-pt0025-ban-ncontig-variance-fixed-distr1"
    S = init_expanded()
```

```
    # Initiate the run with the starting graph, an iteration
    # count, constant value, and log file
    plans = chain(
            S,
            4000000,
            0.0025,
            "logs/{}".format(basename)
            )

    # Log results, filter out illegal plans, remove
    # duplicates, write legal plans to file, and conduct and
    # log election results
    with open("logs/{}.txt".format(basename), "a+") as f:
        sys.stdout = f
        print("\nRun complete.\n")

        contiguous, reject = reject_islands(plans)
        apport, reject2 = reject_by_pop(contiguous)

        print("{} raw plans".format(len(plans)))
        print("Kept {} clean plans".format(len(apport)))
        print(
            """Rejected {} malapportioned plans
               and {} non-contiguous plans
            """.format(
                    len(reject2),
                    len(reject)
                )
        )

        no_dups = remove_dups(apport)
        to_json(
            no_dups,
            "plans/{}.json".format(basename)
        )
        election(
            no_dups,
            "elections/{}.csv".format(basename)
        )


if __name__ == "__main__":
    main()
```

A.2 PROPOSE

```
############################################################
###################     propose.py     ####################
############################################################
#!/usr/bin/env python
""" Generate proposals for Markov chain """

# Imports
from random import randint as rint


def transistion(graph):
    """ Computes transition of a random source node to a new
    district and returns the proposed new district, the
    node, and its transition probabilities in both
    directions """

    # Acquire a source node at random from the full set of
    # available nodes and produce a list of districts of its
    # neighbouring nodes
    ncount = len(list(graph.nodes))
    sourceNode = rint(1, ncount)
    nbors = list(graph.neighbors(sourceNode))
    nbors_distrs = []
    for node in nbors:
        nbors_distrs.append(graph.nodes[node]["distr"])

    # Log the current and proposed new districts for
    # computing transition probabilities and sending forward
    # to acceptance probability
    cur_distr = graph.nodes[sourceNode]["distr"]
    prop_distr = nbors_distrs[rint(0, len(nbors_distrs) - 1)]

    # Compute 'outbound' transition probability as number of
    # neighbours in the proposed district over the total
    # number of neighbours
    #
    # Compute 'inbound' transition as the number of
    # neighbours in the original source district over the
    # total number of neighbors
    trans_out = nbors_distrs.count(prop_distr) / len(nbors)
    trans_in = nbors_distrs.count(cur_distr) / len(nbors)
```

```
    trans = {
        "node": sourceNode,
        "prop_distr": prop_distr,
        "trans_out": trans_out,
        "trans_in": trans_in,
    }

    return trans
```

A.3 SCORE

```
##############################################################
###################      score.py      #######################
##############################################################
#!/usr/bin/env python
""" Source code for scoring districting plans. This includes
population, contiguity and the full score which combines
them """

from math import exp as e
from statistics import pstdev

from utils import distr_count
from utils import distr_pop
from utils import contig_distr


def score_plan(plan, const):
    """ Accepts a districting plan in the form of a networkx
    graph and scores it using two sub-scores: population
    balance and district contiguity. The score function
    takes the form of an energy function: e^-x. A
    parameterised constant is used to appropriately scale
    the results of the sub-scores """

    # Get population score parameter
    pop_param = score_pop(plan)

    # Get contiguity score parameter
    contig_param = score_contig(plan)

    # Energy function with constant to adjust 'strength' of
    # scoring
    score = e(-const * pop_param * contig_param)

    return score
```

```
def score_pop(plan):
    """ Score a supplied plan for its population balance.
    That is, check the population variance of each district
    from the ideal average. Returns a variance value """

    # Retrieve the number of districts in the given plan and
    # create empty list for population values from each
    # district
    num_distrs = distr_count(plan)
    distr_pops = []

    for i in range(1, num_distrs + 1):
        # Tabulate population of each district and append to
        # list
        distr_pops.append(distr_pop(i, plan))

    # pstdev is the standard library standard deviation
    # function for an iterable input
    std_dev = pstdev(distr_pops)
    var = std_dev ** 2

    return var


def score_contig(plan):
    """ Scores a supplied plan based on the number of
    contiguous districts present. If all are contiguous,
    returns 1; elsec returns 10^6 """

    num_distrs = distr_count(plan)

    # Build up a list of results for the contiguity test
    contig_list = []
    for i in range(1, num_distrs + 1):
        contig_list.append(contig_distr(i, plan))

    # Count the number of disconnected districts and set
    # score value
    if all(contig_list):
        score = 1
    else:
        score = 1000000

    return score
```

A.4 CHAIN

Logging functionality has been removed from this file to ease presentation. The complete source file is available for review on the author's GitHub page.

```
############################################################
###################      chain.py      #####################
############################################################
#!/usr/bin/env python

# Imports
from copy import deepcopy as dc
from random import uniform

from propose import transistion
from score import score_plan
from score import score_contig
from score import score_pop

# Markov Chain Simulation
def chain(init_plan, n, const, fname):
    """ Markov chain function for computing new
    redistricting plans based off of an initial districting,
    an iteration count, and a constant value """

    # While loop to control number of iterations
    i = 1

    # Place initial graph at the start of list of graphs
    # output by the chain and initialise it as the current
    # plan
    plans = [init_plan]
    curr_plan = init_plan

    while i <= n:

        # Get a proposal using transition() function
        trans = transistion(curr_plan)

        # Extract relevant bits from the transition() return
        sourceNode = trans["node"]
        prop_distr = trans["prop_distr"]
        trans_out = trans["trans_out"]
        trans_in = trans["trans_in"]

        # Get the current source node's district assignment
        curr_distr = curr_plan.nodes[sourceNode]["distr"]
```

```
        # If the proposal and current districts are
        # different, we have a non-identical but adjacent
        # plan, so proceed to acceptance probability
        if prop_distr != curr_distr:

            # Deep copy makes sure proposal is a new python
            # object
            prop_plan = dc(curr_plan)
            # Set proposal district
            prop_plan.nodes[sourceNode]["distr"] = prop_distr

            # Score up both plans
            score_curr = score_plan(curr_plan, const)
            score_prop = score_plan(prop_plan, const)
            pop_curr = score_pop(curr_plan)
            pop_prop = score_pop(prop_plan)
            contig_curr = score_contig(curr_plan)
            contig_prop = score_contig(prop_plan)

            # Compute acceptance
            alpha = min(
                        1,
                        (
                            (score_prop / score_curr)
                            * (trans_in / trans_out)
                        )
                    )
            beta = uniform(0, 1)

            # Accept or reject
            if alpha > beta:
                curr_plan = prop_plan
                plans.append(curr_plan)
                flag = "accept"
            else:
                flag = "reject"

        # If the proposal and current districts are the
        # same, then the plans are identical and we have a
        # repeat
        else:
            flag = "repeat"

        i += 1

    return plans
```

A.5 REJECT

```
###############################################################
##################       reject.py       ######################
###############################################################
#!/usr/bin/env python
""" File containing functions to reject illegal districting
plans produced by the Markov chain """

from utils import distr_count
from utils import contig_distr


def reject_islands(plans):
    """ Accepts a plans list, containing a series of graphs
    of possible districting plans, and rejects any plans
    where at least one district is non-contiguous (i.e. has
    an island). Returns a list of plans with non-contiguous
    plans removed """

    clean_plans = []
    reject_plans = []

    # Get a districting plan (graph) from the list of plans
    for graph in plans:
        # Retrieve the number of districts in the given plan
        num_distrs = distr_count(graph)

        # Create an empty list to hold the boolean returns
        # from contiguity check
        graph_check = []

        # Iterate through each distract in the plan
        for distr in range(1, num_distrs + 1):
            graph_check.append(contig_distr(distr, graph))

        if False in graph_check:
            reject_plans.append(graph)
        else:
            clean_plans.append(graph)

    return clean_plans, reject_plans
```

```
def reject_by_pop(plans):
    """ Accepts a plans list, containing a series of graphs
    of possible districting plans, and rejects any plans
    where the population of any district exceeds: total
    population * 1/number of districts (+/- 5%).  Returns a
    list of plans with non-contiguous plans removed """

    clean_plans = []
    reject_plans = []

    # Get a districting plan (graph) from the list of plans
    for graph in plans:
        # Retrieve the number of districts in the given plan
        num_distrs = distr_count(graph)

        # Create empty list to be populated with booleans
        # for each district
        graph_check = []

        # Iterate through districts in current plan
        for i in range(1, num_distrs + 1):

            # Assume pop. balance
            distr_check = True

            nodes_in_distr = [
                node for node, attrs in
                graph.nodes(data=True) if
                attrs["distr"] == i
            ]
            total_pop = 0
            for node in nodes_in_distr:
                total_pop += graph.nodes[node]["pop"]

            # Reject if district pop. is more than +/- 5% of
            # average district pop (250)
            if total_pop > 263 or total_pop < 237:
                distr_check = False

            graph_check.append(distr_check)

        if False in graph_check:
            reject_plans.append(graph)
        else:
            clean_plans.append(graph)

    return clean_plans, reject_plans
```

A.6 ELECTION

```
##############################################################
###################     election.py    ######################
##############################################################
#!/usr/bin/env python
""" Source code for calculating voting totals and
statistics"""

import pandas as pd

from utils import distr_count
from utils import distr_nodes


def election(graph_list, fname=None):
    """ Accepts a list of graph(s) and computes results of
    an elections under each districting plan in the list.
    Returns a dataframe of election results """

    results = {}
    i = 0
    pcount = 0

    # Iterate over each graph in the list
    for graph in graph_list:
        num_distrs = distr_count(graph)
        plan_id = id(graph)
        plan_label = pcount

        # Iterate over each district in the graph
        for distr in range(1, num_distrs + 1):
            nodes = distr_nodes(distr, graph)
            vote_circ = 0
            vote_sqre = 0
            total = 0

            # Access individual nodes for their variable
            # values and use them to calculate district
            # totals
            for node in nodes:
                vote_circ += graph.nodes[node]["vote_circ"]
                vote_sqre += graph.nodes[node]["vote_sqre"]
                total += graph.nodes[node]["pop"]
```

```
            # Each district result becomes an entry in a
            # dictionary which gets converted to a dataframe
            results[i] = [
                    plan_id,
                    plan_label,
                    distr,
                    vote_circ,
                    vote_sqre,
                    total
            ]
            i += 1

        pcount += 1

    # Convert results dictionary to a dataframe of all
    # election results
    df = pd.DataFrame.from_dict(
        results,
        orient="index",
        columns=[
            "plan_id",
            "plan_label",
            "distr",
            "vote_circ",
            "vote_sqre",
            "total"
        ],
    )

    # Write election results to specified file
    if fname:
        df.to_csv(fname)

    return df
```

A.7 UTILITIES

```
##############################################################
###################      utils.py      #######################
##############################################################
#!/usr/bin/env python
""" Utility and helpful functions used across the
application """

from collections import deque


def distr_nodes(distr, graph):
    """ Accept a district from a graph and parse, returning
    a list of all nodes in that district """

    nodes_in_distr = [
        node for node, attrs in
        graph.nodes(data=True) if
        attrs["distr"] == distr
    ]

    return nodes_in_distr


def distr_count(graph):
    """ Function to retrieve the number of districts present
    in a supplied graph """

    distrs = []
    for _, values in graph.nodes.data():
        distrs.append(values["distr"])

    count = set(distrs)
    count = len(count)
    return count


def distr_pop(distr, graph):
    """ Tabulates the number of residents in a supplied
    district of a supplied graph """

    nodes_in_distr = distr_nodes(distr, graph)
    total_pop = 0
    for node in nodes_in_distr:
        total_pop += graph.nodes[node]["pop"]

    return total_pop
```

```
def contig_distr(distr, graph):
    """ Deploys breadth-first search algorithm to determine
    if supplied district is contiguous. Returns boolean
    result """

    # Produce list of nodes in the district in question
    d_nodes = distr_nodes(distr, graph)

    # A proposal plan which eliminates a district will get
    # rejected 100% of the time due to transition
    # probabilities, but this function is called before that
    # rejection would occur. As a result, it errors out in
    # this scenario because the d_nodes list above is empty
    # no nodes. We fix this with a quick conditional
    if not d_nodes:
        return False
    else:
        init_node = d_nodes[0]

        # Send required arguments to breadth-first search
        # (BFS) algorithm to get a list of contiguous nodes
        # in the district
        contig_nodes = bfs(init_node, graph)

        # While the code above initialises with the first
        # node in the district, it doesn't really matter. If
        # the BFS returns any list of contiguous nodes that
        # is not exactly the same as the full list of nodes
        # in the district, the district cannot be contiguous
        if set(d_nodes) == set(contig_nodes):
            return True
        else:
            return False


def bfs(node, graph):
    """ Breadth-first search (BFS) for traversing the number
    of connected nodes which share the district assignment
    of the supplied node. This is used to
    determine if the district is contiguous. """

    # BFS is implemented using a queue and visited pair of
    # lists. The queue tracks nodes that need to be
    # explored. Once explored, they are removed from the
    # queue and appended to the visited list. A while loop
    # keeps the search going until the queue is empty.
    queue = deque([node])
    visited = [node]
```

```
    # Get district of supplied node
    distr = graph.nodes[node]["distr"]

    while queue:
        curr_node = queue.popleft()
        nbors = list(graph.neighbors(curr_node))
        d_nbors = [
                item for item in
                nbors if
                graph.nodes[item]["distr"] == distr
        ]
        for n in d_nbors:
            if n not in visited:
                queue.append(n)
                visited.append(n)

    return visited


def graph_sig(graph):
    """ Function which takes in a districting graph and
    returns a numerical signature of that graph. This
    signature represents the node-to-district
    mapping of all nodes in the graph """

    # Start with signature as a string for easy
    # concatenation
    sig = ""

    for n in list(graph.nodes):
        sig += str(n)
        sig += str(graph.nodes[n]["distr"])

    return int(sig)
```

```
def remove_dups(graph_list):
    """ Accepts a list of districting graphs and returns a
    new list without duplicates, where a duplicate is graph
    with the same district assignments
    for every node"""

    uniq_sigs = []
    uniq_graphs = []
    for graph in graph_list:
        sig = graph_sig(graph)
        if sig not in uniq_sigs:
            uniq_sigs.append(sig)
            uniq_graphs.append(graph)

    print("{} duplicate plans removed".format(
                                        len(graph_list) -
                                        len(uniq_graphs)
                                    )
    )

    return uniq_graphs
```