Quantifying Gerrymandering using Markov Chain Monte Carlo Algorithms

Samarth Wahal

Contents

1 Introduction

In the United States, every state is partitioned into constituencies known as districts. People living in these districts vote and elect representatives to the Congress or the state legislature. Redistricting is the process of redrawing district lines within states following the decennial census. While redistricting is crucial to the functioning of representative democracy for a variety of reasons, the vague laws surrounding the process provide political actors an opening to exploit for electoral gain through the practice of gerrymandering. Gerrymandering is the manipulation of district lines to promote certain voter interests and to restrict others. It is usually characterized by oddly shaped districts and uneven district lines [18]. Even so, it is hard to prove that a particular district is gerrymandered.

Redistricting is required to follow certain rules and regulations. The equipopulation requirement — which demands that all districts in a state have similar population — is one of the major rules. A contiguity mandate is also imposed upon districts by most states. Another significant requirement is that districts must be compact [6]. The terms of the equipopulation and compactness requirement are highly unclear. It may be the case that having equal populations for every district in a state is unattainable. For the compactness requirement, there is no precise definition put forth by the law — it simply requires that the geography of a district should not be too spread out. In fact, in order to make sense of this requirement, there are several proposed definitions of compactness in the redistricting literature [9]. These requirements, along with other ambiguous ones such as competitiveness, preserving "communities of interest", and preserving existing political subdivisions [3], make the problem of detecting gerrymandering quite challenging.

A number of approaches for quantifying gerrymandering have been proposed. Some like partisan symmetry [13] and the Efficiency gap [19] deal with partisan gerrymandering in particular. Partisan symmetry prefers plans in which the election outcome would be the same for either party if the share of votes received by them was the same. An example would be if Democrats received 60% of the votes in a state and won 65% of the seats, then Republicans should have won 65% of the seats as well had they received 60% of the votes [8]. On the other hand, the Efficiency gap prefers plans with a smaller amount of "wasted votes." Any votes beyond the threshold needed to win a seat are considered wasted votes [19]. Besides only addressing a specific version of the problem, there are several known examples for which these metrics produce incorrect results. One of the reasons for this is that these metrics do not take into account the geometry of a state or its districts. There also exist many methods relying on some form of optimization for producing "good" redistrictings; however, most of these may not be used for judging the quality of a particular redistricting. Instead, we turn toward computational methods that are able to randomly sample a redistricting plan [8]. If these random samples are representative of the distribution of all valid redistrictings, then we can compare an existing redistricting plan against these samples in order to get a p-value. We can then use this p-value to test if the existing redistricting plan is gerrymandered or not. Note that most randomized algorithms in the literature for finding redistrictings will not work for this task unless they are guaranteed to sample from the space of all possible valid district maps. Therefore, we direct our attention to Markov Chain Monte Carlo (MCMC) algorithms.

MCMC algorithms are able to sample from complicated distributions or from a large state space such as that of all valid district maps. They have been shown to perform better than other algorithms for sampling in [11] and in [5]. However, while the algorithms are proven to be correct, these papers exhibit a lack of analysis surrounding mixing time and the inclusion of legal constraints. In this paper, we implement these algorithms and empirically study their behavior on a graph of the state of Georgia.

2 Literature Review

A major portion of literature in the field of redistricting focuses on algorithms for producing redistrictings subject to certain constraints. A common thread between such algorithms is that they represent a given US state as a graph G. The vertices $V(G)$ of this graph may be a small piece of land such a census tract or a precinct. We let $uv \in E(G)$ be an edge if for $u, v \in V(G)$, the precincts associated with u and v border one another. Let k be the number of districts in the state. Then the vertex set of G can be partitioned into k districts. For notational convenience, we write this as $V(G) = (V_1, \ldots, V_k)$ where each V_i represents the i-th district in a fixed ordering of the districts of the state. Note that for any vertex $v \in V(G)$, v must belong to exactly one V_i for $i \in [k]$. While it is true that a precinct may belong to more than one district, this scenario is rare and is usually ignored for simplicity. Several constraints are then placed on the structure of G. The contiguity constraint is the most essential one. It states that districts in a state must be contiguous, that is, for any two points x and y in a district, there must exist a way to travel on land from x to y that does not venture into another district. In terms of our graph formulation, we can write this constraint as follows: for any partition V_i of $V(G)$, if $u, v \in V_i$, then there exists a $u - v$ path P s.t. for any vertex $x \in P$, $x \in V_i$.

In order to better illustrate the aforementioned graph theoretic formulation, we present a precinct-level graph

created from the current districts of Georgia in Figure 1. Each vertex represents a precinct of Georgia and a pair of vertices have an edge connecting them if their corresponding precincts are adjacent to one another. The color of each vertex represents the district that its corresponding precinct lies in. So, different colored vertices imply that their corresponding precincts lie in different districts. Finally, notice that the set of vertices corresponding to a any single color form a contiguous partition. This partition represents a district of Georgia. Note that Georgia has 14 districts and 2690 precincts.

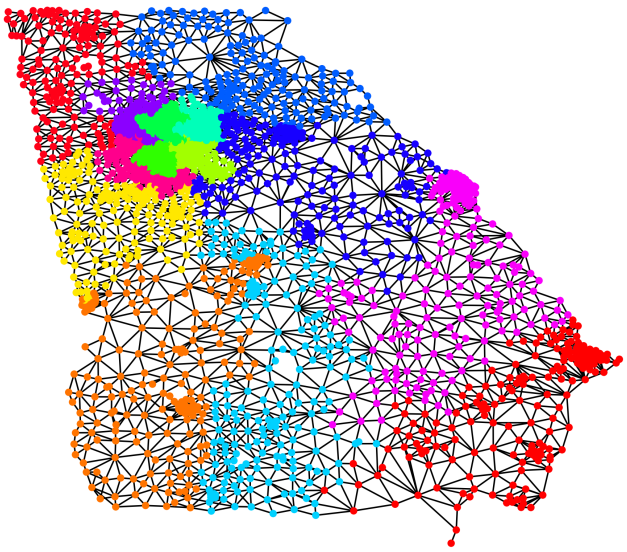


Figure 1: Precinct-level graph of the current districts of Georgia

Aside from contiguity, other constraints, including major ones such as compactness and equipopulation, may be incorporated in non-graph theoretic ways. Several different optimization techniques have been put forth to construct a graph satisfying the given constraints. A paper by Altman and McDonald [1] provides a software to generate redistrictings using methods such as random walks and weighted k-means. Jacobs and Walch [14] use partial differential equations to generate hundreds of reasonable redistrictings in a matter of seconds. Dube and Clark [7] solve the problem of redistricting using a balanced k-way cut algorithm on a vertex-weighted (by population, race, and partisanship data) graph. Dube and Clark also explore a strictly-graph based view of compactness by using the size of edge cuts between district partitions as their metric for compactness.

While the aforementioned papers contain techniques to produce "good" redistrictings, these may not be used to quantify gerrymandering. Proving that a district or state is gerrymandered rigorously is a hard problem. Existing metrics such as partisan symmetry and the efficiency gap are useful only for partisan gerrymandering. Even then, these metrics have failed to stand up in court [8]. However, it is known that sampling-based approaches — specifically those that sample from the distribution of all valid redistricting maps — are a solution to this problem. A large random sample using this approach will be representative of the mentioned distribution. Then comparing an existing redistricting to this sample under some metric lets us compute a p-value that we can use to determine if the redistricting is gerrymandered. Markov Chain Monte Carlo (MCMC) algorithms work especially well for the problem of sampling from an especially large space such as that of all valid redistricting maps. Note that these types of algorithms, by definition, would also accomplish the task of producing "good" redistrictings.

A paper by Fifield *et al.* [11] provides one of the first MCMC approaches to the problem of redistricting [11]. The authors view the problem in terms of a graph — similar to the formulation done above. The algorithm that they provide is a modification of the SWC-1 algorithm from a paper by Barbu and Zhu in 2005 [4]. The SWC-1 algorithm itself is a generalized version of the famous Swendsen-Wang algorithm [20] for the Ising and Potts models to arbitrary posterior distributions. A single step of the algorithm in the paper by Fifield

et al. works by randomly deleting edges from the adjacency graph of G (as described above), randomly selecting connected components along the partition boundaries of G, and then attempting to swap components between partitions to output a new graph G'. The swap is accepted by a ratio based on Swendsen-Wang cuts [4] that is given through the Metropolis criterion. Note that the algorithm is initialized using a graph constructed from an existing valid redistricting plan. The algorithm provably samples from the distribution of all valid redistricting maps; however, no formal bounds on mixing time are provided. Note that mixing time refers to how long it takes a Markov chain to converge to its stationary distribution i.e. the distribution we are sampling from.

There are other papers in the literature that propose similar MCMC algorithms. The algorithm in the paper by Barkstrom *et al.* [5] is strikingly similar to the one by Fifield *et al.* However, Barkstrom *et al.* choose to focus solely on equal size and contiguity constraints and disregard compactness entirely. Barkstrom *et al.* seem to underestimate the impact of a compactness constraint in preventing gerrymandering. In particular, a loose compactness constraint allows for creative ways to segregate voting populations into different districts and influence election outcome. A paper by Bangia *et al.* [2] proposes an MCMC algorithm that moves a single vertex in one iteration rather than a whole component like Swendsen-Wang. This is especially similar to the Glauber dyanamics algorithm for the Ising and Potts models. This approach seems to be slower than Fifield *et al.* but may be simpler in terms of incorporating constraints.

This study focuses on implementing and making improvements to the algorithm in the paper by Fifield *et al.* In particular, we wish to find a set of examples on which the algorithm performs poorly. Furthermore, the algorithm provides empirical evidence of its performance based on a purely geometric (Euclidean distance based) view of compactness. We explore the usage of other measures of compactness [9] along with this algorithm. Finally, we run this algorithm on the Georgia graph to find out if it is gerrymandered or not.

3 Background

3.1 Markov Chains

We will first define a *Markov chain* [16].

Definition 1 (Markov chain)**.** *Let Ω be a finite set. We call Ω the state space. Let $P : \Omega \times \Omega \mapsto [0, 1]$ be a function such that $\sum_{y \in \Omega} P(x, y) = 1$ for each $x \in \Omega$. A Markov chain over Ω and P is a sequence of random variables (X_t) for integer $t \geq 0$ such that $\Pr(X_{t+1} = x_{t+1} \mid X_t = x_t, \ldots, X_0 = x_0) = \Pr(X_{t+1} = x_{t+1} \mid X_t = x_t) = P(x_t, x_{t+1})$ where each $x_i \in \Omega$.*

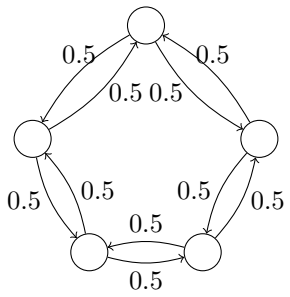
In the above definition, we refer to Ω as the *state space* and P as the *transition function* of the Markov chain. P may be represented as a $|\Omega| \times |\Omega|$ matrix. In this case, we refer to P as the *transition matrix* of the Markov chain. We will sometimes refer to the Markov chain solely by its transition matrix P for the sake of brevity.

A Markov chain may also be represented in a graph theoretic manner. In particular, we can represent a Markov chain P over Ω as a directed graph G where $V(G) = \Omega$ and for every $x, y \in \Omega$, we have $xy \in E(G)$ if and only if $P(x, y) > 0$. Now, we define an edge weight $w : E(G) \mapsto [0, 1]$ on G s.t. for any $x, y \in E(G)$, $w((x, y)) = P(x, y)$. To better illustrate this, consider a simple random walk over $\mathbb{Z}_5$. We can describe this walk as a Markov chain over the state space $\Omega = \{0, 1, 2, 3, 4\}$ where the transition matrix P is as follows:

$$P(x, y) = \begin{cases} 1/2 & y \equiv x + 1 \pmod 5 \\ 1/2 & y \equiv x - 1 \pmod 5 \\ 0 & \text{otherwise} \end{cases}$$

This Markov chain can be represented as the graph of a 5-cycle where if we are at a vertex v at time t, we may only move to either of the two neighbors of v with probability 0.5 at time $t + 1$. In Figure 2, we have the graph that describes this Markov chain.

Figure 2: Simple random walk on a 5-cycle



We now define two important properties of Markov chains.

Definition 2 (Irreducibility)**.** *A Markov chain is irreducible if for all $x, y \in \Omega$, $P^t(x, y) > 0$ for some integer $t \geq 0$.*

In terms of the graph theoretic representation, if a Markov chain P is irreducible, its graph must be *strongly connected*.

Definition 3 (Periodicity)**.** *Let $\mathcal{T}(x) = \{t \geq 1 : P^t(x, x) > 0\}$ for $x \in \Omega$. We refer to $\gcd \mathcal{T}(x)$ as the period of state x.*

If the period of each $x \in \Omega$ of a Markov chain is 1, then we say that the chain is *aperiodic*. We refer to an irreducible and aperiodic Markov chain as *ergodic*. The Markov chain in Figure 2 is aperiodic since if we start a walk from any vertex x, we can get back to x in an even number of steps, or in an odd number t of steps where $t \geq 5$. Note that in general, a random walk on an odd cycle is aperiodic whereas a random walk on an even cycle is periodic with a periodicity of 2.

Definition 4 (Stationary Distribution)**.** *A stationary distribution of a Markov chain is a probability distribution π over Ω such that for each $y \in \Omega$, $\pi(y) = \sum_{x \in \Omega} \pi(x) P(x, y)$.*

We can also write π as a row vector of length $|\Omega|$. In this case, we have that $\pi = \pi P$. Now, we state without proof one of the most useful theorems about Markov chains.

Theorem 1. *An ergodic Markov chain converges to a unique stationary distribution.*

An important consequence of the above theorem is that if run long enough, an ergodic Markov chain will converge to its stationary distribution irrespective of the starting state. The algorithm that we use in this paper employs an ergodic Markov chain that converges to a stationary distribution over the space of all possible redistricting maps.

While ergodicity gives us a guarantee that a Markov chain converges to a unique stationary distribution, it is not immediately useful for our goal of drawing samples from a particular distribution. For our task, we wish to design the transition matrix of a Markov chain that converges to a given distribution over the state space. The following definition will prove to be useful for this.

Definition 5 (Reversibility)**.** *A Markov chain is said to be reversible with respect to a probability distribution π over Ω if it satisfies the following for all $x, y \in \Omega$:*

$$\pi(x)P(x, y) = \pi(y)P(y, x)$$

These equations are known as the detailed balance equations.

Proposition 1. *If a Markov chain P is reversible with respect to a probability distribution π over Ω, then π is a stationary distribution of P.*

Proof. We can sum over the detailed balance equations for each $x \in \Omega$:

$$\begin{aligned}\sum_{x\in\Omega} \pi(x)P(x,y) &= \sum_{x\in\Omega} \pi(y)P(y,x) \\ &= \pi(y)\end{aligned}$$

So, π is a stationary distribution of P by Definition 4. □

3.2 The Ising Model

Consider an undirected graph G. The *Ising model* on G is a spin system where each vertex $v \in V(G)$ represents a particle. A *configuration* $\sigma : V(G) \mapsto \{-1, +1\}$ is a function that represents the spin of each particle in the system. We define the energy of a configuration σ by the *Hamiltonian* as follows:

$$H(\sigma) = -\sum_{uv\in E(G)} \sigma(u)\sigma(v)$$

Our state space is the set of all possible configurations over G. We denote this as $\Omega = \{-1, +1\}^{V(G)}$. For a particular configuration $\sigma \in \Omega$, the Gibbs distribution parameterized by $\beta > 0$ defines a probability over Ω as follows:

$$\mu(\sigma; \beta) = \frac{e^{-\beta H(\sigma)}}{Z(\beta)}$$

Here, $Z(\beta)$ is the *partition function* defined as follows:

$$Z(\beta) = \sum_{\sigma\in\Omega} e^{-\beta H(\sigma)}$$

$Z(\beta)$ can be viewed as a normalizing constant required to make μ a probability distribution. Our goal is to sample from the distribution μ. Note that the state space Ω is huge — there are $2^{|V(G)|}$ possible configurations. Therefore, we cannot simply compute the probability of each configuration.

Since we define $\beta > 0$, the neighboring vertices prefer the same spin. Hence, in this case, the model is called ferromagnetic or attractive. But when $\beta < 0$, then the model is antiferromagnetic or repulsive. Note that we only consider the ferromagnetic case in this paper.

3.3 Glauber Dynamics

We will introduce a Markov chain known as *Glauber dynamics* that will allow us to sample from μ. In each step, the chain chooses a vertex from G u.a.r and updates its spin to $+1$ with probability α or to -1 with probability $1 - \alpha$.

Algorithm 1 Glauber dynamics for the Ising Model

function GLAUBERDYNAMICS(G, β, T)
  $X_0 \leftarrow$ any configuration over G
  for $t \leftarrow 1, \ldots, T$ **do**
    $u \leftarrow$ vertex chosen u.a.r from $V(G)$
    for $v \in V(G)$ **do**
      if $u \neq v$ **then**
        $X_t(v) \leftarrow X_{t-1}(v)$
    $X_t(u) \leftarrow \begin{cases} +1 & \text{w.p. } \alpha(X_{t-1}, u) \\ -1 & \text{w.p. } 1 - \alpha(X_{t-1}, u) \end{cases}$
  return X_T

Let $S(\sigma, u) = \sum_{v \in N(u)} \sigma(v)$. We define the probability function α used in Algorithm 1 as follows:

$$\alpha(\sigma, u) = \frac{e^{\beta S(\sigma,u)}}{e^{\beta S(\sigma,u)} + e^{-\beta S(\sigma,u)}}$$

Claim 1. *Glauber dynamics on the Ising model is reversible with respect to μ.*

Proof. Let σ and λ be two configurations on the graph G s.t. there exists $w \in V(G)$ for which $\sigma(v) = \lambda(v)$ for all vertices $v \neq w$ and $\sigma(w) \neq \lambda(w)$. Note that for any other pair of configurations ρ and τ, the detailed balance equations hold since either $\rho = \tau$ or $P(\rho, \tau) = 0$ (because the spins differ at more than one vertex). We will make use of the following observation:

$$H(\sigma) + \sigma(w)S(\sigma, w) = H(\lambda) + \lambda(w)S(\lambda, w)$$

Let P be the transition matrix of our Markov chain. Then we have that:

$$P(\sigma, \lambda) = \frac{1}{|V(G)|} \frac{e^{\beta\lambda(w)S(\sigma,w)}}{e^{\beta S(\sigma,w)} + e^{-\beta S(\sigma,w)}}$$

This is because we have a $1/|V(G)|$ probability of picking w to flip. The second term is the probability of setting the spin of w to $\lambda(w)$.

Notice that $S(\mu, w) = S(\lambda, w)$ due to our choice of σ and λ. Finally, we have the following:

$$\begin{aligned}
\mu(\sigma)P(\sigma, \lambda) &= \frac{1}{|V(G)|} \frac{e^{-\beta H(\sigma)}}{Z(\beta)} \frac{e^{\beta\lambda(w)S(\sigma,w)}}{e^{\beta S(\sigma,w)} + e^{-\beta S(\sigma,w)}} \\
&= \frac{1}{|V(G)|} \frac{e^{-\beta[H(\lambda)+\lambda(w)S(\lambda,w)-\sigma(w)S(\sigma,w)]}}{Z(\beta)} \frac{e^{\beta\lambda(w)S(\sigma,w)}}{e^{\beta S(\sigma,w)} + e^{-\beta S(\sigma,w)}} \\
&= \frac{1}{|V(G)|} \frac{e^{-\beta H(\lambda)}}{Z(\beta)} \frac{e^{\beta\sigma(w)S(\sigma,w)}}{e^{\beta S(\sigma,w)} + e^{-\beta S(\sigma,w)}} \\
&= \mu(\lambda)P(\lambda, \sigma)
\end{aligned}$$

Therefore, by Definition 5, Glauber dyanamics is reversible. □

By Proposition 1, it follows that μ is a stationary distribution of the Glauber dynamics chain. Now, notice that for $\sigma \in \Omega$, it is always true that $\alpha(\sigma, u) \in (0, 1)$. By line 8 of Algorithm 1, this implies that $\Pr(X_{t+1} = \sigma \mid X_t = \sigma) > 0$. So, by Definition 3, Glauber dynamics is aperiodic. Now, consider any two configurations $\sigma, \lambda \in \Omega$. Let $S = \{v \in V(G) : \sigma(v) \neq \lambda(v)\}$. Since $\Pr(X_{t+1}(v) \neq X_t(v)) > 0$ for any $v \in V(G)$ for our Markov chain P, we have that $P^{|S|}(\sigma, \lambda) > 0$. So, by Definition 2, Glauber dynamics is irreducible as well. Thus, the chain is ergodic. Finally, we can use Theorem 1 to say that μ is the unique stationary distribution of our chain.

3.4 Swendsen-Wang

3.4.1 The Swendsen-Wang Algorithm

While Glauber dynamics attempts to flip a single vertex at each time step, we can achieve faster mixing times on the Ising model by flipping several vertices at once. The Swendsen-Wang algorithm does exactly this. We first define what a *monochromatic* edge is.

Definition 6. *An edge $uv \in E(G)$ is monochromatic with respect to some configuration σ if $\sigma(u) = \sigma(v)$. Otherwise, uv is non-monochromatic.*

The idea behind the Swendsen-Wang algorithm is to map the Ising model onto an edge based model called the *Random Cluster Model*. We make a probabilistic move in this new model. Then we map our state back to the Ising model.

Starting at configuration X_{t-1}, we first remove all non-monochromatic edges in our graph G. Then the remaining edges are monochromatic. This is the Random Cluster representation of our configuration X_{t-1}. Now, we remove each remaining edge with probability $e^{-\beta}$ and keep it with probability $1 - e^{-\beta}$. Next, we gather all the connected components in our current graph. Note that isolated vertices count as a connected component. For each connected component, we assign all its vertices the same spin chosen u.a.r. This forms our new configuration X_t in the Ising model.

Algorithm 2 Swendsen-Wang algorithm for the Ising Model

1: **function** SWENDSEN-WANG(G, β, T)
2: $X_0 \leftarrow$ any configuration over G
3: **for** $t \leftarrow 1, \ldots, T$ **do**
4: $G^{(0)} \leftarrow G$ with non-monochromatic edges w.r.t. X_{t-1} removed ▷ $G^{(0)}$ only has monochromatic edges now
5: $G^{(1)} \leftarrow G^{(0)}$ where each edge is kept w.p. $1 - e^{-\beta}$ and removed w.p. $e^{-\beta}$
6: $C \leftarrow$ set of connected components of $G^{(1)}$
7: **for** $C_i \in C$ **do**
8: $\texttt{spin} \leftarrow \begin{cases} +1 & \text{w.p. } 1/2 \\ -1 & \text{w.p. } 1/2 \end{cases}$
9: **for** $v \in V(C_i)$ **do**
10: $X_t(v) = \texttt{spin}$
11: **return** X_T

3.4.2 Random Cluster Model

To prove the correctness of this algorithm, we introduce the *Random Cluster Model* [12]. Let G be an undirected graph. Let $S \subseteq E(G)$ formed by keeping each edge with probability p and removing it with probability $1 - p$. Let $C(S)$ denote the number of connected components in the graph $(V(G), S)$. Note that isolated vertices count as a single component. Let $\omega(S) = 2^{C(S)} p^{|S|} (1-p)^{|E(G) \setminus S|}$. The probability of obtaining S is then defined as follows:

$$\phi(S) = \frac{\omega(S)}{Z_{RC}}$$

Here, Z_{RC} is the partition function defined as follows:

$$Z_{RC} = \sum_{S \subseteq E(G)} \omega(S)$$

Essentially, in Algorithm 2, line 4 can be viewed as mapping the Ising model to the Random Cluster Model, and the block starting line 7 maps the Random Cluster Model back to the Ising model. Let $\sigma \in \Omega$ and $S \subseteq E(G^{(0)})$. We define $Pr(\sigma \to S)$ as the probability of moving from configuration σ to S in lines 4 and 5 of Algorithm 2. Similarly, for $T \subseteq E(G)$ and $\lambda \in \Omega$, we let $\Pr(T \to \lambda)$ denote the probability of moving from T to configuration λ in the block starting line 7. Lastly, we denote by $m(\sigma)$ the number of monochromatic edges in G with respect to configuration σ.

Before we show that μ is a stationary distribution of the Swendsen-Wang algorithm, we perform a few adjustments to our definitions to make the proof easier. First, notice that $m(\sigma)$ can be written in terms of the Hamiltonian as follows:

$$m(\sigma) = \frac{1}{2}(|E(G)| - H(\sigma))$$

Now, we redefine the distribution μ of the Ising model as follows:

$$\begin{aligned}\mu(\sigma;\beta) &= \frac{e^{-\beta H(\sigma)}}{\sum_{\sigma\in\Omega} e^{-\beta H(\sigma)}} \\ &= \frac{e^{2\beta m(\sigma)-\beta|E(G)|}}{\sum_{\sigma\in\Omega} e^{2\beta m(\sigma)-\beta|E(G)|}} \\ &= \frac{e^{2\beta m(\sigma)-\beta|E(G)|}}{\sum_{\sigma\in\Omega} e^{2\beta m(\sigma)-\beta|E(G)|}} \\ &= \frac{e^{2\beta m(\sigma)}}{\sum_{\sigma\in\Omega} e^{2\beta m(\sigma)}}\end{aligned}$$

Rescaling the parameter β to 2β, we get that:

$$\mu(\sigma;\beta) = \frac{e^{\beta m(\sigma)}}{Z_I(\beta)}$$

Here, the partition function is $Z_I = \sum_{\sigma\in\Omega} e^{\beta m(\sigma)}$. Similarly, for the Random Cluster Model, we can rewrite ϕ as such:

$$\begin{aligned}\phi(S) &= \frac{2^{C(S)}p^{|S|}(1-p)^{|E(G)\setminus S|}}{\sum_{S\subseteq E(G)} 2^{C(S)}p^{|S|}(1-p)^{|E(G)\setminus S|}} \\ &= \frac{(1-p)^{|E(G)|}2^{C(S)}p^{|S|}(1-p)^{-|S|}}{(1-p)^{|E(G)|}\sum_{S\subseteq E(G)} 2^{C(S)}p^{|S|}(1-p)^{-|S|}} \\ &= \frac{1}{Z_{RC}}2^{C(S)}p^{|S|}(1-p)^{-|S|}\end{aligned}$$

Where the partition function $Z_{RC} = \sum_{S\subseteq E(G)} 2^{C(S)}p^{|S|}(1-p)^{-|S|}$. Now, we are ready to prove the following two claims. For both proofs, we let $p = 1 - e^{-\beta}$ as stated in Algorithm 2. These proofs are based on the ones in a paper by Edwards and Sokal [10].

Claim 2. *If $\sigma \sim \mu$, then $E(G^{(1)})$ in Algorithm 2 has distribution ϕ.*

Proof. Let $S = E(G^{(1)})$. Then we have the following:

$$\begin{aligned}\Pr(S) &= \sum_{\sigma\in\Omega} \mu(\sigma)\Pr(\sigma\to S) \\ &= \sum_{\sigma:S\subseteq m(\sigma)} \frac{1}{Z_I}e^{\beta m(\sigma)}p^{|S|}(1-p)^{m(\sigma)|S|} \\ &= \frac{1}{Z_I}\sum_{\sigma:S\subseteq m(\sigma)} (1-p)^{-m(\sigma)}p^{|S|}(1-p)^{m(\sigma)-|S|} \\ &= \frac{1}{Z_I}\sum_{\sigma:S\subseteq m(\sigma)} p^{|S|}(1-p)^{-|S|} \\ &= \frac{1}{Z_I}2^{C(S)}p^{|S|}(1-p)^{-|S|}\end{aligned}$$

Now, we show that $Z_I = Z_{RC}$.

$$\begin{aligned}1 &= \sum_{T\subseteq E(G)} \Pr(T) \\ &= \frac{1}{Z_I}\sum_{T\subseteq E(G)} 2^{C(T)}p^{|T|}(1-p)^{-|T|} \\ &= \frac{Z_{RC}}{Z_I}\end{aligned}$$

Therefore, we have that $\Pr(S) = \phi(S)$. □

Claim 3. *If $E(G^{(1)}) \sim \phi$, then X_t in Algorithm 2 has distribution μ.*

Proof. Let $\sigma = X_t$. Then we have the following:

$$\begin{aligned}
\Pr(\sigma) &= \sum_{T \in \Omega_{RC}} \phi(T) \Pr(T \to \sigma) \\
&= \sum_{T : T \subseteq m(\sigma)} \frac{1}{Z_{RC}} 2^{C(T)} p^{|T|} (1-p)^{-|T|} \frac{1}{2^{C(T)}} \\
&= \frac{1}{Z_I} \sum_{T : T \subseteq m(\sigma)} p^{|T|} (1-p)^{-|T|} \\
&= \frac{1}{Z_I} \sum_{k=1}^{m(\sigma)} \sum_{T : |T| = k} p^k (1-p)^{-k} \\
&= \frac{1}{Z_I} \sum_{k=1}^{m(\sigma)} \binom{m(\sigma)}{k} \left(\frac{p}{1-p} \right)^k \\
&= \frac{1}{Z_I} \left(\frac{p}{1-p} + 1 \right)^{m(\sigma)} \\
&= \frac{1}{Z_I} \left(\frac{1}{1-p} \right)^{m(\sigma)} \\
&= \frac{1}{Z_I} e^{\beta m(\sigma)} \\
&= \mu(\sigma)
\end{aligned}$$

□

By Claims 2 and 3, we have that μ is a stationary distribution of the Swendsen-Wang Markov chain. Note that this chain P is aperiodic by Definition 3 since $P(\sigma, \sigma) > 0$ for any configuration σ. This is because the algorithm allows for the case where we do not flip any component.

Claim 4. *The Swendsen-Wang Markov chain P is irreducible.*

Proof. Let $\mu, \lambda \in \Omega$. Notice that the probability of $|C| = |V(G)|$ is non-zero. That is, there is a non-zero probability that after line 5 we end up with a graph $G^{(1)}$ that contains no edges. Thus, each vertex is in a connected component on its own. Now, for each $v \in V(G)$, the probability of changing the spin from $\sigma(v)$ to $\lambda(v) = (1/2)^{|V(G)|} > 0$. Thus, $P(\mu, \lambda) \geq \Pr(\mu \to E(G^{(1)})) \cdot (1/2)^{|V(G)|} > 0$. So by Definition 2, we have that P is irreducible. □

So, we have that the chain is ergodic. Finally, by Theorem 1, μ is the unique stationary distribution of the Swendsen-Wang algorithm.

3.5 Metropolis Algorithm

The idea behind the Metropolis algorithm [17] is to break the transition probability $P(\sigma, \lambda)$ of a Markov chain P into a proposal probability $\kappa(\sigma, \lambda)$ and an acceptance probability $\alpha(\sigma, \lambda)$ for $\sigma, \lambda \in \Omega$. That is, instead of simply transitioning to a new state with some probability, we first propose a new state and then probabilistically accept or reject this proposal. This means that if our chain is at state X_{t-1} and the algorithm proposes a new state X', then we set X_t to X' w.p. $\alpha(X_{t-1}, X')$ and to X_{t-1} w.p. $1 - \alpha(X_{t-1}, X')$. We choose κ and α s.t. they satisfy the detailed balance equations w.r.t the distribution μ.

$$\begin{aligned}
\mu(\sigma) P(\sigma, \lambda) &= \mu(\lambda) P(\lambda, \sigma) \\
\mu(\sigma) \kappa(\sigma, \lambda) \alpha(\sigma, \lambda) &= \mu(\lambda) \kappa(\lambda, \sigma) \alpha(\lambda, \sigma) \\
\frac{\alpha(\sigma, \lambda)}{\alpha(\lambda, \sigma)} &= \frac{\mu(\lambda) \kappa(\lambda, \sigma)}{\mu(\sigma) \kappa(\sigma, \lambda)}
\end{aligned}$$

Note that setting $\alpha(\sigma, \lambda) = \min\left(1, \frac{\mu(\lambda)\kappa(\lambda, \sigma)}{\mu(\sigma)\kappa(\sigma, \lambda)}\right)$ satisfies the above expression. This setting is known as the Metropolis criterion. By definition 5, μ is a stationary distribution of P.

We now tailor this algorithm to the Ising model. We first need to come up with a good proposal probability κ. A natural proposal for a configuration σ over G is to arrive at a new configuration λ by flipping the spin of one of the vertices of G. Following this plan, we can arrive at one of exactly $|V(G)|$ configurations from σ. Thus, we have:

$$\kappa(\sigma, \lambda) = \begin{cases} 1/|V(G)| & \text{if } \sigma(v) \neq \lambda(v) \text{ for exactly one } v \in V(G) \\ 0 & \text{otherwise} \end{cases}$$

Note that $\kappa(\sigma, \lambda) = \kappa(\lambda, \sigma)$ by our definition. Thus, we set the acceptance probability $\alpha(\sigma, \lambda)$ to $\min\left(1, \frac{\mu(\lambda; \beta)}{\mu(\sigma; \beta)}\right)$. In concrete terms, this probability is equivalent to:

$$\begin{aligned} \alpha(\sigma, \lambda) &= \min\left(1, \frac{\mu(\lambda; \beta)}{\mu(\sigma; \beta)}\right) \\ &= \min\left(1, \left(\frac{e^{-\beta H(\lambda)}}{Z(\beta)}\right)\left(\frac{e^{-\beta H(\sigma)}}{Z(\beta)}\right)^{-1}\right) \\ &= \min\left(1, \frac{e^{-\beta H(\lambda)}}{e^{-\beta H(\sigma)}}\right) \end{aligned}$$

Notice that we get rid of the partition function $Z(\beta)$ which is usually intractable to compute.

Algorithm 3 Metropolis algorithm for the Ising Model

function METROPOLIS(G, β, T)
  $X_0 \leftarrow$ any configuration over G
  for $t \leftarrow 1, \ldots, T$ **do**
    $u \leftarrow$ vertex chosen u.a.r. from $V(G)$
    $X'(u) = (-1) \cdot X_{t-1}(u)$ ▷ Flip the spin of the randomly chosen vertex
    for $v \in V(G)$ **do**
      if $u \neq v$ **then**
        $X'(v) = X_{t-1}(v)$ ▷ Every other vertex has the same spin as before
    $X_t \leftarrow \begin{cases} X' & \text{w.p. } \alpha(X_{t-1}, X') \\ X_{t-1} & \text{w.p. } 1 - \alpha(X_{t-1}, X') \end{cases}$ ▷ Accept or reject the proposal based on α
  return X_T

At each time step X_{t-1} of Algorithm 3, we pick a single vertex from $V(G)$ uniformly at random and flip its spin to obtain our proposed state X'. Note that this probability of obtaining X' as our proposed state is exactly $\kappa(X_{t-1}, X') = 1/|V(G)|$. Then with probability $\alpha(X_{t-1}, X')$, as previously defined, we accept the proposal and set the new state X_t to X'. With probability $1 - \alpha(X_{t-1}, X')$, we reject the proposal and X_t is the same as our old state X_{t-1}.

Note that this chain is ergodic by the same arguments as those for the Glauber Dynamics chain. We have also shown above that this chain is reversible w.r.t μ. Therefore, μ is the unique stationary distribution of the Metropolis algorithm.

3.6 The Potts Model

We briefly mention the *Potts Model* — a generalization of the Ising model to any finite number of spins q over a graph G. All of the algorithms discussed in this section can be generalized to the Potts model as

well. We note that the space of all valid redistricting maps is more similar to the Potts model than the Ising model since each precinct is allowed to be in one of k districts.

4 The Algorithm

In this section, we will define the Markov chain that we use for sampling redistricting plans. We will also show how to ensure that our plans adhere to equipopulation and compactness requirements.

4.1 Overview

We use the algorithm proposed by Fifield *et al.* [11] with some changes. The input to this algorithm is an undirected connected graph G, number of partitions k, number of timesteps T, number of components to be swapped R, and the probability of keeping an edge q. Let Ω_R be the state space of all valid redistricting maps on G. We will run this algorithm N times with the same parameters to produce N samples from a distribution over Ω_R. We will define the exact distribution later in this section. We first give an overview of what each function used in Algorithm 4 does.

- SeedPartition(G, k)
  - Divides graph G into k random contiguous partitions
- PartitionGraph(G)
  - Removes all non-monochromatic edges from graph G
- RemoveEdges(G, q)
  - Removes each monochromatic edge from G with probability $1 - q$ and keeps it with probability q
- ConnectedComponents(G)
  - Returns all connected components of graph G
- BoundaryComponents(G, CP)
  - Input CP is a set of connected components of G
  - Returns all components in CP that lie along a partition boundary of G
- NonAdjacentBoundaryComponents(G, BCP, R)
  - Input BCP is a set of components that lie along a partition boundary of G
  - Returns R components from BCP s.t. none of these components are adjacent to one another in G
- ProposeSwap(VCP)
  - Input VCP is a set of non adjacent boundary components in G
  - Proposes a new configuration over G where each component in VCP is swapped into one of its adjacent partitions in G chosen u.a.r.
- AcceptRejectProposal(X_{t-1}, X')
  - (Accept) Returns the new configuration X' with probability $\alpha(X_{t-1}, X')$
  - (Reject) Returns the old configuration X_{t-1} with probability $1 - \alpha(X_{t-1}, X')$

Algorithm 4 Redistricting algorithm

function RedistrictingChain(G, k, T, q)
  $X_0 =$ SeedPartition(G, k)
  for $t \leftarrow 1, \ldots, T$ **do**
    $G^{(0)} \leftarrow$ PartitionGraph(X_{t-1})
    $G^{(1)} \leftarrow$ RemoveEdges($G^{(0)}, q$)
    $CP \leftarrow$ ConnectedComponents($G^{(1)}$)
    $BCP \leftarrow$ BoundaryComponents(G, CP)
    $VCP \leftarrow$ NonAdjacentBoundaryComponents(G, BCP, R)
    $X' \leftarrow$ ProposeSwap(G, VCP)
    $X_t \leftarrow$ AcceptRejectProposal(X_{t-1}, X')
  return X_T

Note that at each step t of the algorithm, X_t is a contiguous graph consisting of k partitions. At each step $t \in [T]$, the algorithm requires X_{t-1} to be a contiguous graph as well. Therefore, we need to set the initial state X_0 of the algorithm. The SeedPartition function takes an connected input graph G and a number of partitions k to divide this graph into. The output of this function is a contiguous partitioned graph which suffices as X_0. The primary reason we use such as function rather than a previously known contiguous partition (as is in the case of redistricting maps) is so that our samples are not biased by starting at the same initial state.

Algorithm 5 Seed partition algorithm

function SeedPartition(G, k)
  $V_1, \ldots, V_k \leftarrow$ sets consisting of 1 vertex each of G chosen u.a.r without replacement
  $V \leftarrow$ remaining vertices of G
  while V has vertices left **do**
    for $i \leftarrow 1, \ldots, k$ **do**
      $C \leftarrow$ set of neighbors of the node boundary of V_i s.t. no neighbor is in any V_j for $j \neq i$
      $C' \leftarrow$ vertices from C that are in V
      $v \leftarrow$ vertex chosen u.a.r. from C'
      $V_i \leftarrow V_i \cup \{v\}$
      $V \leftarrow V - v$
  return $V_1, \ldots, V_k$

At a high level, Algorithm 5 is similar to a flood fill algorithm. We start off by choosing k vertices from $V(G)$ u.a.r. The idea is to grow a random tree around each of these vertices in G. At each step, for each tree T_i, we u.a.r pick a vertex v from $N_G(V(T))$ s.t. v does not belong to another tree. Then we add v to T_i and repeat the process until each vertex in $V(G)$ belongs to some tree. Finally, on termination, $V(T_1), \ldots, V(T_k)$ form our contiguous seed partitions. Figure 3 shows an example output of SeedPartition on the Georgia graph with $k = 14$. Each color represents a partition, and we omit edges between partitions for clarity.

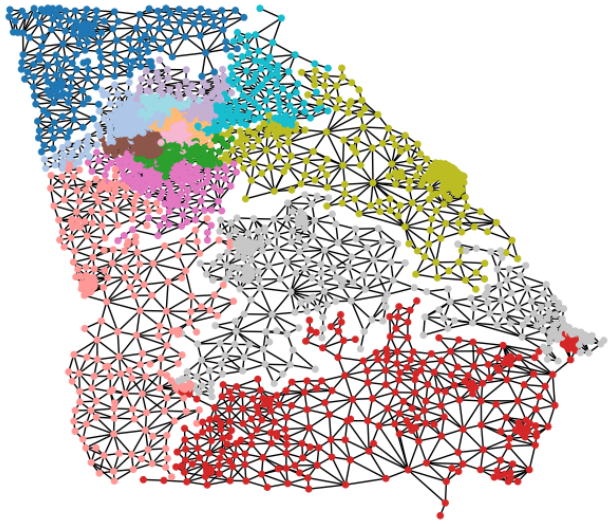


Figure 3: Partitions generated by Algorithm 5 on the Georgia graph

While we have previously stated that each X_t is a contiguous graph, it is also useful to look at it as a configuration over the graph G. Specifically, we have that $X_t : |V(G)| \to [k]$. That is, X_t maps each vertex to one of our k partitions. To impose the contiguity requirement, we have that for any partition $c \in [k]$ and $S = \{v \in V(G) : X_t(v) = c\}$ that $G[S]$ is connected. Now, recall Definition 6 from the Swendsen-Wang algorithm. We slightly modify this definition for Algorithm 4 as follows:

Definition 7. *An edge $uv \in E(G)$ is monochromatic with respect to X_t if $X_t(u) = X_t(v)$. Otherwise, it is non-monochromatic. Similarly, a subgraph G' of G is monochromatic if there exists $c \in [k]$ s.t. for each $v \in V(G')$, $X_t(v) = c$.*

The PartitionGraph function removes all non-monochromatic edges with respect to X_{t-1} from G to form the graph $G^{(0)}$. $G^{(0)}$ only contains monochromatic edges and will look similar to the graph in Figure 3. The RemoveEdges function simply perform the following task on its inputs (graph $G^{(0)}$ and $q \in [0, 1]$): construct a graph $G^{(1)}$ s.t. $V(G^{(1)}) = V(G^{(0)})$ and for each edge $e \in E(G^{(0)})$, we let $e \in E(G^{(1)})$ with probability q. Thus we are throwing away edge e with probability $1 - q$. Next, ConnectedComponents returns the connected components CP of its input graph $G^{(1)}$. Notice that each component in CP must be monochromatic. After that, BoundaryComponents(CP) produces the set of components BCP from CP that lie along the boundary of some partition in G. In other words, for every $C \in BCP$, there exists a monochromatic edge w.r.t X_{t-1} in the cut-set $[V(C), V(G) \setminus V(C)]$. Then NonAdjacentBoundaryComponents picks a subset of R components VCP from BCP s.t. for each pair of components $C_1, C_2 \in VCP$ where $C_1 \neq C_2$, we must have that for any pair of vertices $x \in C_1$ and $y \in C_2$, $xy \notin G$. That is, no two components should be adjacent to one another in G. If this condition is not satisfied, we keep performing step 8 of the algorithm again until it is. Note that for some choice of parameters to our algorithm, this step may loop for several iterations. Next, in ProposeSwap, for each $C \in VCP$, we assign as new partition to the vertices of C. This partition is chosen u.a.r from the partitions neighboring the vertices of C. Let this proposed swap be X'. AcceptRejectProposal sets X_t to X' with probability α. Otherwise, it sets X_t to X_{t-1}. For each step, α is chosen based on the Metropolis criterion. Note that Fifield *et al.* claim that for one of their chosen values of α, the stationary distribution of their Markov chain is uniform over the set of all valid configurations over G.

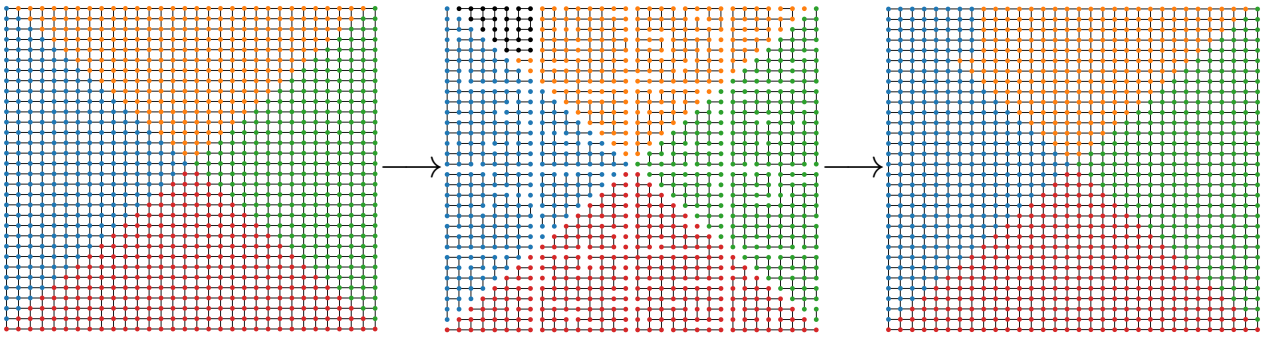


Figure 4: A successful swap taking place after one iteration of Algorithm 4

In Figure 4, we provide an example of a single iteration of Algorithm 4 on a square lattice with four partitions/districts with $R = 1$. Notice that the middle image does not contain any non-monochromatic edges. This is due to the call to PartitionGraph. Several monochromatic edges are also missing because of RemoveEdges. The black highlighted component on the boundary is the candidate for our proposed swap. The swap is successful and the component is part of the adjacent partition now.

4.2 Incorporating Constraints

4.2.1 Why do we need constraints?

While the Markov chain described above can sample uniformly from the distribution all valid redistricting maps, our primary goal is to see if a given map is gerrymandered or not. Thus, we would like to have a batch of samples that mostly consists of "good" redistricting maps so that we can see how our given map measures up against the ideal ones. Legal literature specifies that a "good" redistricting map should at least satisfy the equipopulation and compactness requirements. So, we will need to define a distribution that weighs maps adhering to these requirements more than those that do not. Before doing that, we will discuss the formulation of the aforementioned requirements as constraints that will be computed at each iteration of Algorithm 4.

4.2.2 Compactness

The geographical compactness requirement is fairly vague. In broad terms, it requires districts to look reasonable or compact. Due to this lack of clarity, there are several proposed definitions of compactness in the literature. Fifield *et al.* use the definition proposed by Fryer and Holden [15]. Let V_i be a partition of $V(G)$ imposed by a configuration X_t. Then their measure of compactness can be defined as follows:

$$\psi(V_i) = \sum_{u,v \in V_i} e_u e_v d_{uv}^2 \tag{1}$$

Here, e_u and e_v represent the population densities of vertices u and v. d_{uv} is the distance between the centroids of the precincts corresponding to u and v. The compactness for the entire graph is then $\sum_{i=1}^{k} \psi(V_i)$ where $V_1, \ldots, V_k$ are the partitions of G imposed by X_t. According to this definition, the higher the compactness score, the less compact a district or state is.

Note that this constraint will need to be computed at every iteration of our algorithm. We can perform this computation in a naive way — simply calculate the sum of $e_u e_v d_{uv}^2$ for every pair of vertices in each partition. However, this results in an $O(|V(G)|^2)$ cost at each time step. We propose a faster way of computing this compactness score.

Algorithm 6 Compactness algorithm

```
1: function ComputeCompactness(G, X', X_{t-1}, VCP, prev_score)
2:     curr_score ← prev_score
3:     ignore ← map of each v ∈ V(G) to an empty set
4:     for C ∈ VCP do
5:         for u ∈ C do
6:             for v ∈ V(G) do
7:                 if v ∈ ignore[u] then
8:                     continue
9:                 wasSameColor ← X_{t-1}(u) = X_{t-1}(v)
10:                isSameColor ← X'(u) = X'(v)
11:                if ¬wasSameColor & isSameColor then
12:                    curr_score ← curr_score + Score(u, v)
13:                    if X'(v) = X_{t-1}(v) then
14:                        ignore[v].add(u)
15:                if wasSameColor & ¬isSameColor then
16:                    curr_score ← curr_score − Score(u, v)
17:                    if X'(v) = X_{t-1}(v) then
18:                        ignore[v].add(u)
19:    return curr_score
```

Given a previous compactness score, Algorithm 6 computes a new score by only considering changes in compactness caused by the components VCP that are swapped. Notice that for each $v \in V(C)$ for any $C \in VCP$, we only need to compute compactness between v and the vertices of the partitions that it lies in before and after the swap. This works well when $R = |VCP| = 1$. However, when $|VCP| > 1$, we need to consider two edge cases. Let $C_1, C_2 \in VCP$ s.t. $C_1 \neq C_2$. The first case is when C_1 and C_2 lie in the same partition and swap into different partitions. The second is when C_1 and C_2 lie in different partitions and swap into the same partition. In both these cases, we only need to add/subtract Score(u, v) exactly once for a pair of vertices $x \in C_1$ and $y \in C_2$. Algorithm 6 prevents double counting Score(u, v) for these edge cases by storing information about if u and v have already been considered in a map. Note that now we only need to naively compute compactness for the initial state X_0. For every other time step, we can use Algorithm 6.

As an aside, notice that Equation 1 uses Euclidean distance between vertices in the same partition to compute the compactness score. A better idea would be to set d_{uv} to the length of the shortest path p between u and v s.t. p lies entirely inside the partition V_i. Note that in this case, the cost of a single edge uv would be our original definition of d_{uv} — the distance between the centroids of the precincts corresponding to u and v. Our redefinition provides accurate distances between precincts and may result in a more meaningful compactness score. However, computing this score is expensive. Computing all pairs shortest path with the Floyd-Warshall algorithm takes $O(|V(G)|)^3$ time. Given the compactness score from the previous time step, we can compute $\psi(V_i)$ efficiently for a partition V_i that only has vertices being swapped into it. We can do this by using Djikstra's algorithm to compute distances between the newly swapped vertices and the already existing ones in V_i. However, for a partition V_j that has vertices being swapped out of, there does not seem to be an easy method to compute $\psi(V_j)$. In particular, for $x, y \in V_j$ and a vertex z being swapped out of V_j, we will need to recompute d_{xy} since the shortest path between x and y may have included z. Thus, we will have to compute all pairs shortest path at each step of our algorithm. This is too slow for our use case, so we stick with the original definition of compactness.

4.2.3 Equipopulation

The equipopulation requirement mandates that the districts of a state have similar sizes. Suppose p_v represents the population of the precinct corresponding to a vertex $v \in V(G)$. The *population parity* $\hat{p}$ of G is

then:

$$\hat{p} = \frac{\sum_{v \in V(G)} p_v}{k}$$

That is, $\hat{p}$ is the mean population of the districts of a state. We would like for the population of each state in our sampled redistricting map to be close to this mean. Thus, the equipopulation constraint is defined as follows:

$$\rho(V_i) = \left| \frac{\sum_{v \in V_i} p_v}{\hat{p}} - 1 \right|$$

The equipopulation constraint for the entire graph is then $\sum_{i=1}^{k} \rho(V_i)$. Note that we only need to compute population parity once at the start of Algorithm 4. We also need to compute the initial equipopulation for X_0 in $O(|V(G)|)$ time by iterating over each vertex of G. For every other X_t, we can employ a strategy similar to Algorithm 6. We simply keep track of the total population for each partition from X_{t-1} in a list. Then for each swapped vertex v, we increase/decrease the populations for partitions $X_{t-1}(v)$ and $X'(v)$ appropriately.

4.3 The Distribution

For the stationary distribution of their chain, Fifield *et al.* choose the same distribution as the one we saw for the Ising model — the Gibbs distribution parameterized by $\beta > 0$ and weights w_c, w_e:

$$g(\sigma; \beta, w_c, w_e) = \frac{\exp\left(-\beta \sum_{i=1}^{k} (w_c \cdot \psi(V_i) + w_e \cdot \rho(V_i))\right)}{Z_R(\beta)} \tag{2}$$

Here, $Z_R(\beta)$ is the partition function defined as follows:

$$Z_R(\beta, w_c, w_e) = \sum_{\sigma \in \Omega_R} \exp\left(-\beta \sum_{i=1}^{k} (w_c \cdot \psi(V_i) + w_e \cdot \rho(V_i))\right)$$

Distribution g gives higher weight to maps with lower compactness and equipopulation scores. Note that this distribution can be easily extended to incorporate more constraints than the ones we have defined. To converge to the distribution g, the only thing we need to change in Algorithm 4 is the acceptance probability α. Again, α is based on the Metropolis criterion. Fifield *et al.* define it as:

$$\alpha(\sigma \to \lambda) = \min\left(1, \frac{g(\lambda; \beta, w_c, w_e)}{g(\sigma; \beta, w_c, w_e)} \left(\frac{|B(CP, \sigma)|}{|B(CP, \lambda)|}\right)^R \frac{(1-q)^{|C(\lambda, VCP)|}}{(1-q)^{|C(\sigma, VCP)|}}\right)$$

Where $B(CP, \sigma)$ is defined as the set of boundary components BCP as obtained in line 7 of Algorithm 4. Note that $C(\sigma, VCP)$ is called a Swendsen-Wang cut [4]. It is defined as follows:

$$C(\sigma, VCP) = \{uv \in E(G) : uv \text{ monochromatic w.r.t. } \sigma, \exists C \in VCP \text{ s.t. } u \in C_i, v \notin C_i\}$$

So, $C(\sigma, VCP)$ is the set of monochromatic edges w.r.t. σ that need to be cut to form VCP.

5 Methodology

To construct the graph of Georgia, we begin with the state's precinct-level Shapefiles provided by the US government. A Shapefile is a file format that stores geographical information. We convert each precinct to a vertex and store the position of its centroid. We add an edge to our graph for every pair of adjacent precincts. To find these neighboring precincts, we pad each precinct's polygonal shape by a small amount and find the precinct polygons that intersect with our current one. Note that some precincts may be disconnected from land. In this case, we connect them to their nearest neighboring precinct. We use the modified algorithm to sample redistricting plans from the graph of Georgia. For each sample, we run the algorithm for $T = 10000$

time steps. To improve the mixing time of our Markov chain, we employ a linear scaling of the β parameter. We start with $\beta = 5 \times 10^{-9}$ and end at $\beta = 5 \times 10^{-2}$. We set the probability of keeping an edge $q = 0.5$. We move $R = 2$ components in one time step. We let the equipopulation weight $w_e = 5 \times 10^{10}$ and the compactness weight $w_c = 1$. The huge difference in the weight is to account for the vastly different scales of our equipopulation and compactness scores. Our null hypothesis is that Georgia is not gerrymandered. We set the significance level as 0.05. To test this hypothesis, we draw a 100 samples of redistricting maps. For each sample, we tally the number of votes received by each candidate in the Democrat and Republican parties. To compare the current redistricting plan for Georgia with the samples, we use the following metric: number of Republican seats won in the house. Since we are working with a two-party system, it does not matter which party's seats we choose to measure. This measure will let us compute a p-value that will provide an indication for whether the current redistricting plan is gerrymandered or not: that is, if the plan is an outlier compared to our samples with regard to this measure, it may be gerrymandered. Note that the current redistricting plan results in 10 Republican seats for the 2016 Georgia elections. Thus, we will use the proportion of samples that result in *at least* 10 Republican seats won to compute the p-value.

6 Data Cleaning

In order to run our algorithm over the graph of Georgia, we require geographical data on the precinct-level. We obtain this data in the form of Shapefiles from the MGGG group. The MGGG group obtained its underlying data from the Georgia General Assembly Legislative and Congressional Reapportionment Office. After drawing samples of redistrictings, we need to examine the number of Democratic/Republican seats won for each sample. To do this, we require precinct-level election results for Georgia. We obtain the 2016 Georgia election results from the MIT Election Data and Science Lab.

An issue that arises here is that for many precinct names in the Shapefile, there does not exist an exact match in the election dataset. That is, precinct names across the two datasets are similar but not the same. So, we need to construct a good one-to-one mapping from the precinct names in the Shapefile to those in the election dataset. We construct this mapping by phrasing this problem as a *stable marriage* problem.

Algorithm 7 Data matching algorithm

```
1: function MatchPrecinctNames(counties, shape, election)
2:     M ← county-wise mapping between shape and election precincts
3:     for county in counties do
4:         shape_precincts ← shape[county]
5:         election_precincts ← election[county]
6:         prefs ← PrefList(shape_precincts, election_precincts)
7:         M[county] ← Gale-Shapely(shape_precincts, election_precincts, prefs)
8:     return M
```

The inputs to MatchPrecinctNames are a list of Georgia's county names, precinct names in the Shapefile indexed by county name, and precinct names in the election dataset indexed by county name. We treat precincts names within a single county as one stable marriage problem. The PrefList function generates preference lists for each precinct in either dataset with respect to the precinct names in the other dataset. It does so by constructing an ordering through approximate string matching using the Levenshtein distance. Finally we obtain a matching for this county by feeding the precinct names from both datasets along with these preference lists to the Gale-Shapely algorithm which produces a solution to our stable marriage problem. Note that since `shape_precincts` and `election_precincts` may be lists of different lengths, and the traditional Gale-Shapely algorithm expects equal size inputs, we add "dummy" precincts to the smaller list. We pad all relevant preference lists by placing these dummy precincts at the end of the list — that is, the dummy precincts are always least preferred. Note that for our datasets, it is always true that $length(\texttt{shape\_precincts}) \leq length(\texttt{election\_precincts})$. Thus, M is always a valid one-to-one mapping.

7 Results and Discussion

As mentioned in Section 5, we sample an ensemble of 100 redistricting plans using the algorithm. 6 plans from this ensemble are shown in Figure 5. We omit the edges between district partitions for clarity.

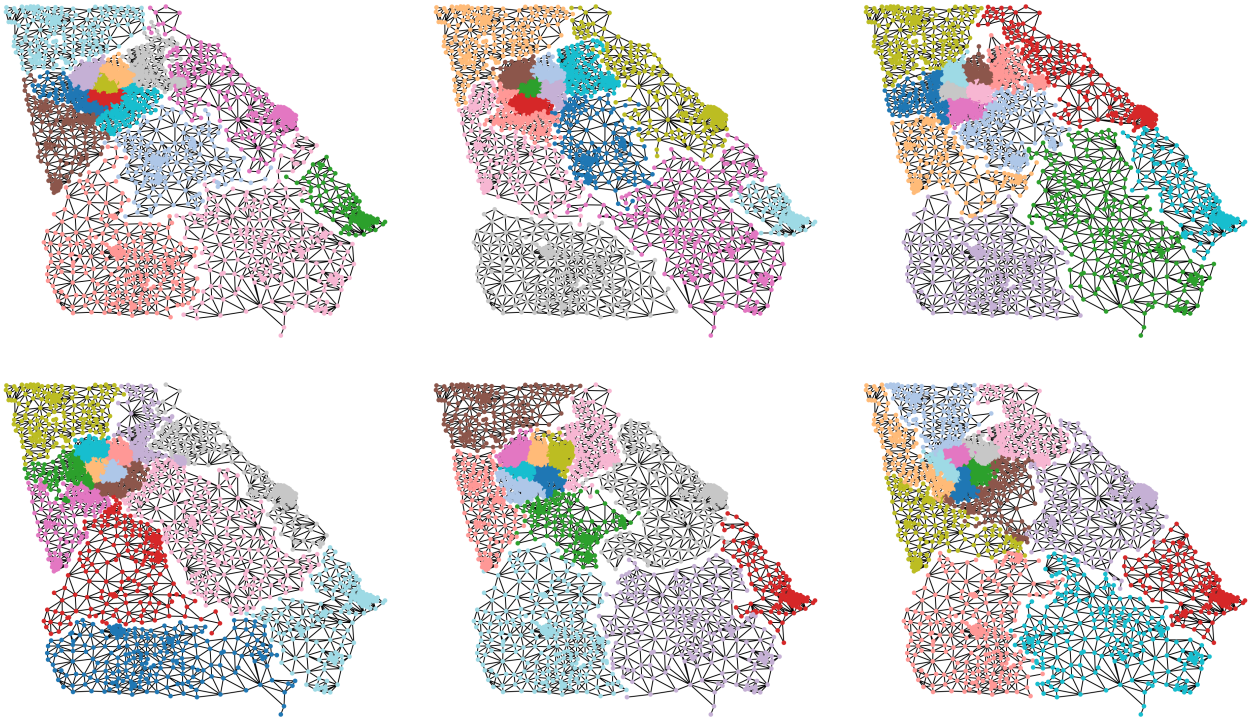
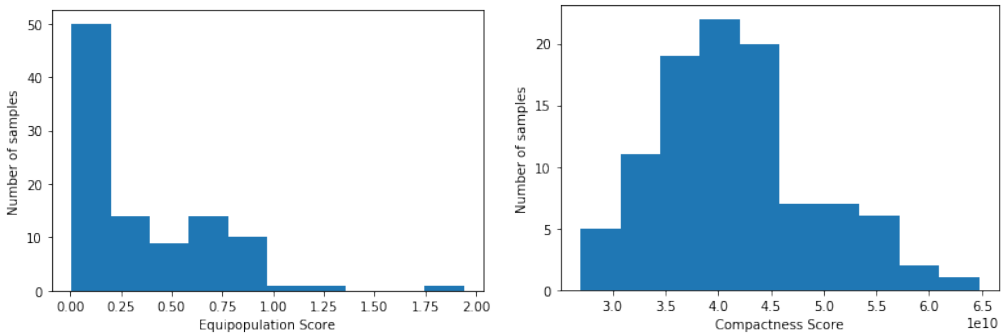


Figure 5: Samples generated by Algorithm 4

In Figure 6a, we have the distribution of equipopulation scores computed for our ensemble. The mean of this distribution is 3.353×10^{-1}. The median is 2.003×10^{-1}. Georgia's current district plan has an equipopulation score of 1.279×10^{-1}. Figure 6b is the distribution of compactness scores computed for our ensemble. The mean of this distribution is 4.192×10^{10}. The median is 4.122×10^{10}. Georgia's current district plan has a compactness score of 4.838×10^{10}. Note that lower scores are better for both of the constraints.



(a) Distribution of equipopulation scores

(b) Distribution of compactness scores

Figure 6: Distribution of constraint scores for 100 samples

In Figure 7, we have the distribution of maximum equipopulation scores over the districts of each sampled

plan. A max. equipopulation score δ for a sample can be interpreted as saying that the population of each district for the sample is within $(100 \cdot \delta)\%$ of the population parity. The mean of this distribution is 1.559×10^{-1}. The median is 9.75×10^{-2}. For the current redistricting plan of Georgia, this value is 3.9×10^{-2}.

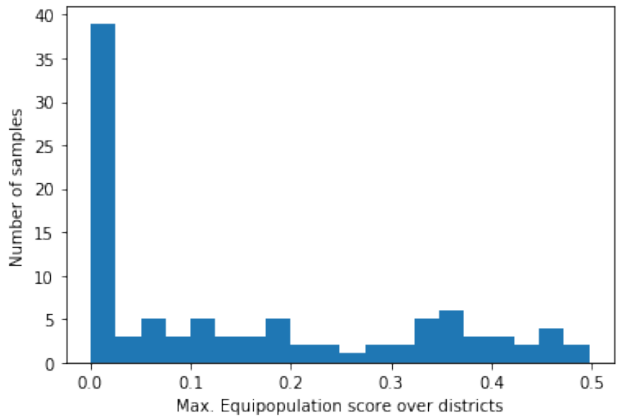
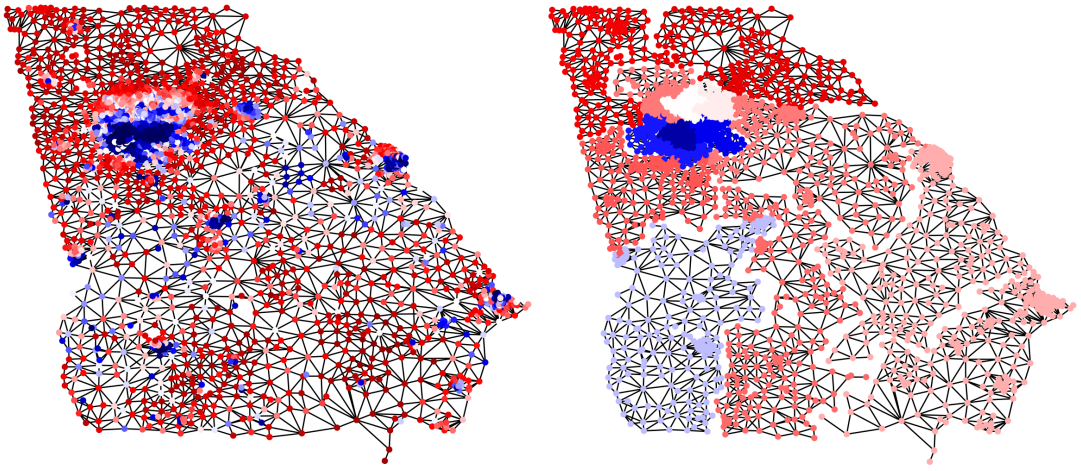


Figure 7: Distribution of maximum equipopulation scores over districts

Since the algorithm moves precincts around to come up with a new redistricting plan, we require Georgia's precinct-level voting data to see how many votes each district in our plan receives. We note that in the 2016 House of Representatives election in Georgia, 4 districts were uncontested in favor of Republicans and 1 was uncontested in favor of Democrats. So, it is likely that this data may not be representative of Georgia's precinct-level voting patterns. Thus, we perform our analysis based on the 2016 Presidential election precinct-level voting patterns. In doing so, we rely on the assumption that people vote for candidates from the same party in both House and Presidential elections. Figure 8 depicts Georgia's voting patterns based on the 2016 Presidential election data. The red and blue colors imply Republican and Democratic majority votes respectively. The intensity of the color denotes the percentage vote. For example, a darker red denotes that Republicans received most of the votes. In figure 8a, we have these voting patterns at a precinct-level. In figure 8b, we compute the votes for each district and color the entire district based on which party received more votes. Note that based on these votes, Republicans win 10 seats — the same as they did in the 2016 House of Representatives election.



(a) Precinct-level voting patterns

(b) District-level voting patterns

Figure 8: Voting patterns based on 2016 Georgia Presidential election voting data

Figure 9 shows the votes per district for the 6 plans depicted in Figure 5.

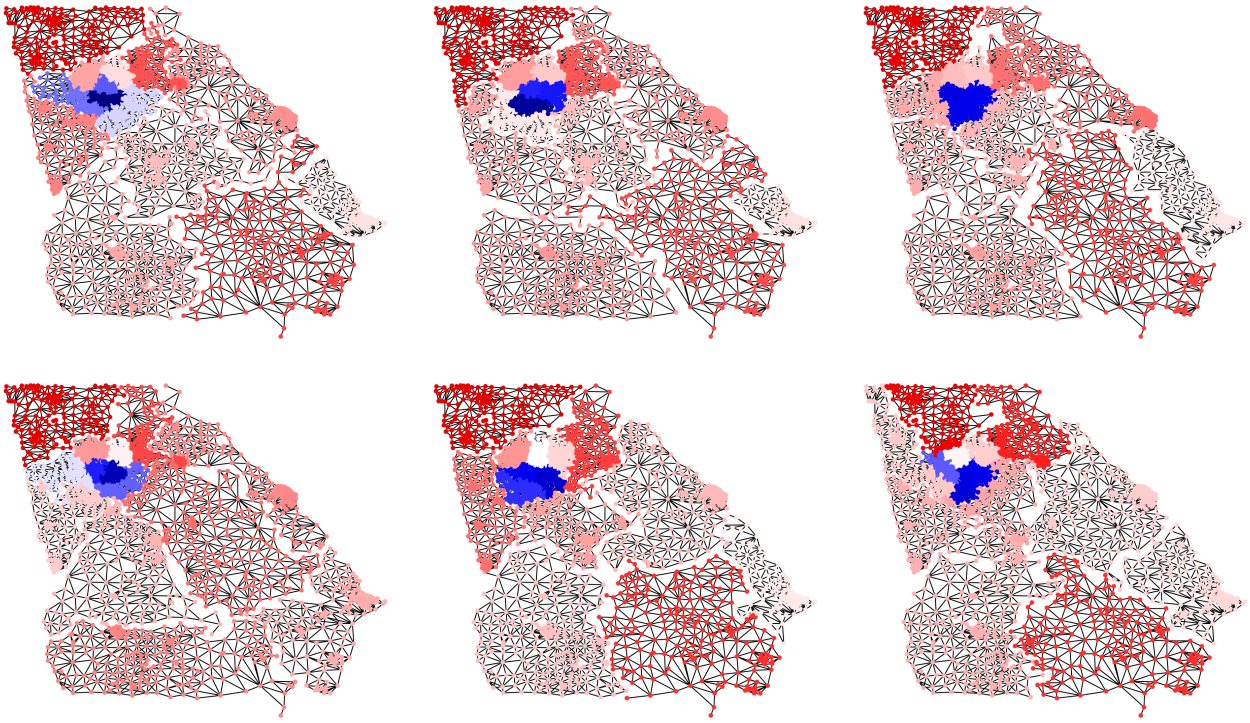


Figure 9: District-level votes for samples generated by Algorithm 4

Figure 10 is the distribution of Republican seats won for our ensemble of plans. 3 plans win 8 seats, 23 win 9 seats, 57 win 10 seats, and 17 win 11 seats. As mentioned in section 5, our null hypothesis is that Georgia is not gerrymandered. According to this distribution, the p-value of this null hypothesis is 0.74. This is greater than our significance level of 0.05. Thus, we fail to reject the null hypothesis.

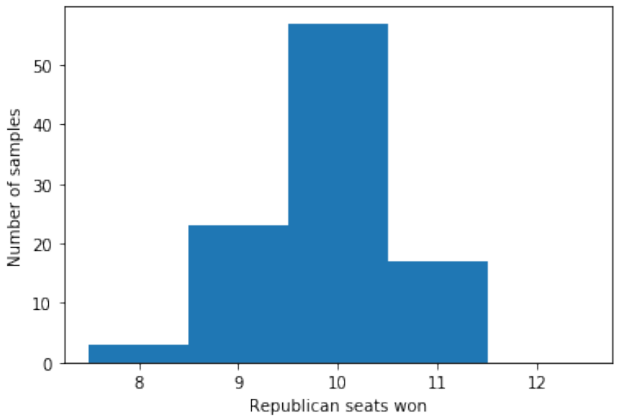


Figure 10: Histogram of number of Republican House seats won based on 2016 Georgia Presidential elections voting data

8 Conclusion

In this work, we implemented and modified the MCMC algorithm designed by Fifield *et al.* We used this algorithm to sample redistricting plans over the graph of Georgia. We tested our null hypothesis that Georgia is not gerrymandered against these samples. According to our results, we fail to reject this null hypothesis. It is important to note that while the Markov chain eventually converges to the distribution specified by equation 2, we do not have any bounds on the mixing time. That is, we cannot guarantee that $T = 10000$ time steps are enough to produce a sample from the aforementioned distribution. Finally, we point out that our implementation of the algorithm and the data pipeline is flexible enough to be tailored to Shapefiles and election datasets of different US states.

References

[1] M. Altman and M. P. McDonald, "BARD: Better Automated Redistricting," *Journal of Statistical Software*, vol. 42, no. 4, pp. 1–28, 2011, ISSN: 1548-7660. DOI: 10.18637/jss.v042.i04. [Online]. Available: https://www.jstatsoft.org/v042/i04.

[2] S. Bangia, C. V. Graves, G. Herschlag, H. S. Kang, J. Luo, J. C. Mattingly, and R. Ravier, "Redistricting: Drawing The Line," *Preprint*, 2017. [Online]. Available: https://arxiv.org/abs/1704.03360.

[3] J. Barabas and J. Jerit, "Redistricting Principles and Racial Representation," *State Politics & Policy Quarterly*, vol. 4, no. 4, pp. 415–435, 2004. DOI: 10.1177/153244000400400404. [Online]. Available: https://doi.org/10.1177/153244000400400404.

[4] A. Barbu and S.-C. Zhu, "Generalizing Swendsen-Wang to Sampling Arbitrary Posterior Probabilities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1239–1253, 2005.

[5] J. Barkstrom, R. Dalvi, and C. Wolfram, "Detecting Gerrymandering with Probability: A Markov Chain Monte Carlo Model," *Preprint*, 2018. [Online]. Available: http://www.dam.brown.edu/siam/2017/jrc.pdf.

[6] A. Cox, "Partisan Fairness and Redistricting Politics," *New York University Law Review*, vol. 79, no. 3, pp. 751–802, Jun. 2004.

[7] M. P. Dube and J. T. Clark, "Beyond the Circle: Measuring District Compactness Using Graph Theory," (Northeastern Political Science Association, Boston, MA, November, 2016), 2016. [Online]. Available: https://www.researchgate.net/publication/311557290_Beyond_the_Circle_Measuring_District_Compac

[8] M. Duchin, "Gerrymandering metrics: How to measure? What's the baseline?" *Bulletin of the American Academy for Arts and Sciences*, vol. Winter 2018, 2018. [Online]. Available: https://arxiv.org/abs/1801.02064.

[9] M. Duchin and B. E. Tenner, "Discrete geometry for electoral geography," *Preprint*, 2018. [Online]. Available: https://arxiv.org/abs/1808.05860.

[10] R. G. Edwards and A. D. Sokal, "Generalization of the Fortuin-Kasteleyn-Swendsen-Wang representation and Monte Carlo algorithm," *Phys. Rev. D*, vol. 38, pp. 2009–2012, 6 Sep. 1988. DOI: 10.1103/PhysRevD.38.2009. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.38.2009.

[11] B. Fifield, M. Higgins, K. Imai, and A. Tarr, "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo," *Princeton University Working Paper*, 2018. [Online]. Available: https://imai.fas.harvard.edu/

[12] G. Grimmett, *The Random-Cluster Model*, ser. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin, 2006, vol. 333. DOI: 10.1007/978-3-540-32891- [Online]. Available: https://doi.org/10.1007/978-3-540-32891-9.

[13] B. Grofman and G. King, "The Future of Partisan Symmetry as a Judicial Test for Partisan Gerrymandering after LULAC v. Perry," *Election Law Journal*, vol. 6, no. 1, pp. 2–25, 2008. [Online]. Available: https://gking.harvard.edu/files/abs/jp-abs.shtml.

[14] M. Jacobs and O. Walch, "A partial differential equations approach to defeating partisan gerrymandering," *Preprint*, 2018. [Online]. Available: https://arxiv.org/abs/1806.07725.

[15] R. G. F. Jr. and R. Holden, "Measuring the Compactness of Political Districting Plans," *Journal of Law and Economics*, vol. 54, no. 3, pp. 493–535, 2011. [Online]. Available: https://ideas.repec.org/a/ucp/jlawec/doi10

[16] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov Chains and Mixing Times*. American Mathematical Society, 2006.

[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953. DOI: 10.1063/1.1699114. [Online]. Available: https://doi.org/10.1063/1.1699114.

[18] R. Morrill, "Gerrymandering," *Focus*, vol. 41, no. 3, pp. 23–27, 1991, ISSN: 00155004. [Online]. Available: http://search.proquest.com/docview/198458548/.

[19] N. O. Stephanopoulos and E. M. McGhee, "Partisan Gerrymandering and the Efficiency Gap," *University of Chicago Law Review*, vol. 82, no. 2, pp. 831–900, 2015. [Online]. Available: https://heinonline.org/HOL/P?h=he

[20] R. H. Swendsen and J.-S. Wang, "Nonuniversal critical dynamics in Monte Carlo simulations," *Phys. Rev. Lett.*, vol. 58, pp. 86–88, 2 1986. DOI: 10.1103/PhysRevLett.58.86. [Online]. Available: https://link.aps.org/d