Detecting Gerrymandering with Probability: A Markov Chain Monte Carlo Model

Jackson Barkstrom, Rohan Dalvi, Christopher Wolfram

Revised June 28, 2019

Contents

1 Nontechnical Summary

Gerrymandering–the drawing of electoral districts in order to favor one political group–can dramatically bias the outcomes of elections. However, Gerrymandering is difficult to combat because it is hard to quantify. Recently, a metric known as the efficiency gap has come to prominence as a simple formula for measuring Gerrymandering by looking at wasted votes (votes that did not affect the outcome of an election). The efficiency gap formula, however, has a number of issues. These are explored more in section 6, but essentially efficiency gap measures more than just Gerrymandering.

We propose another, statistical approach. We compare the properties of random districtings to proposed districtings to see if the proposed districts would lead to a significantly different election outcome. That is, our random districtings give us a baseline with which we can compare proposals, and we can see if the proposals have been designed to land at the far tails of the distribution, optimizing for one party or the other. To put things in layman's terms, if the current districting scheme just so "happens" to elect more members of one party than 99.99% of all possible districting schemes (which we will approximate with a random sample), we can infer that it is the result of party-driven Gerrymandering.

We start with simple random districtings in the hypothetical square shaped state of North Squarolina, but we add more and more sophistication until we can generate valid random districtings for not just North Squarolina, but also arbitrary real-world states.

We find that proposed districting (C) in North Squarolina produces an extremely unlikely election outcome, suggesting that it is Gerrymandered, and we find that the 2012 districting in North Carolina produces the most likely election outcome, suggesting that it is not Gerrymandered.

2 Introduction

Our job is to detect the occurrences of Gerrymandering. According to the Cornell Legal Information Institute, Gerrymandering describes "when political or electoral districts are drawn with the purpose of giving one political group an advantage over another, a practice which often results in districts with bizarre or strange shapes" [1]. Historical evidence shows that the presence of Gerrymandering can significantly change outcomes of public policy [2]. Currently three main constraints exist in the United States prohibiting Gerrymandering [3], namely:

1. **Contiguity.** Every voting district must have a connected interior whenever possible.
2. **One Person, One Vote.** Voting districts must contain populations of nearly equal size.
3. **Voting Rights Act.** Voting districts must not dilute the votes of protected minorities.

Despite these constraints, what most people might think of as partisan Gerrymandering often occurs in the U.S. because there is no judicially managable standard to enforce "fair" partisan districting. The Supreme Court has indicated that extreme partisan Gerrymandering–districting that dramatically favors one party over the other–is unconstitutional [4], but courts have no method to even define extreme partisan Gerrymandering. Recently in the case *Gill v. Whitford* (2018), a metric known as efficiency gap that measures "wasted votes" by party was used to try to prove unfair districting in Wisconsin. Efficiency gap generally correlates with

Gerrymandering. In a hypothetical unfair districting scheme using four districts of equal size, for example, a party that has 60% of the popular vote might only win a quarter of all district elections. The majority party will have "wasted" nearly all of its votes by barely losing three elections and winning one in a landslide. In the losing cases, every vote is wasted, and in the landslide case all but the votes needed to win are wasted. Since one party wasted nearly all of its votes and the other party wasted almost none, the efficiency gap metric for this election will be extremely high. And obviously, this could easily be a Gerrymandered election: a party with 60% majority won only 25% of the districts! However, although this obvious example shows the success of the efficiency gap, the efficiency gap is absolutely not a reliable, one-size-fits all method to detect Gerrymandering. During *Gill v. Whitford* the reliability of the metric itself was called into question, and a high efficiency gap was not accepted by the Supreme Court as proof of Gerrymandering in Wisconsin. Later in the paper, we will look more at the efficiency gap and explain some of its flaws.

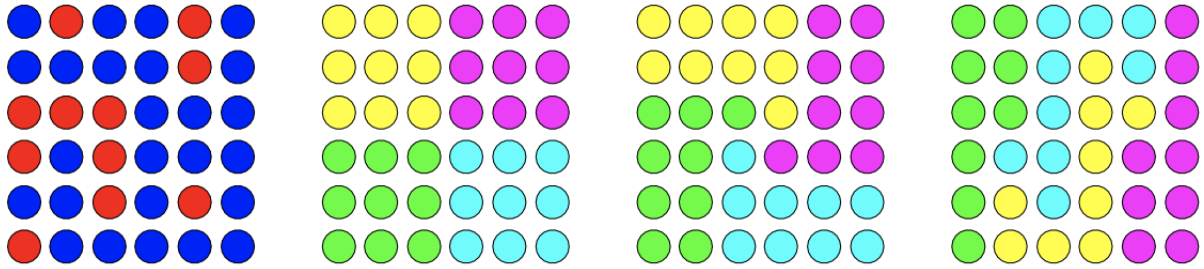
2.1 Our Method

Our method, unlike the efficiency gap, is not a metric. It is a method based on the simple idea of probability. We try to produce a randomly generated sample of all reasonable districtings–a sample of all ways to reasonably draw district boundaries–and we use the sample to test for Gerrymandering. If we are able to produce a sample that approximates well enough a random sample, we can use this generated sample to test for unfair districting. We can mostly figure out if a certain arrangement is Gerrymandered by comparing it with a random sample all possible district arrangements. If the arrangement we are testing elects more members of one party than almost any other arrangement randomly generated, if it is abnormal enough, we can say that it is probably Gerrymandered. Statistically, if the chances of randomly producing a district that happens to favor one party by at least a certain amount is .001, we can be almost certain that the district was Gerrymandered.

True random generation of a sample of reasonable districtings is almost impossible to achieve–the computational power required to generate a random sample of contiguous districts is absurdly large as the size of our space increases. This is because most randomly generated district mappings are not contiguous–thus if 1 in 10000, 10000000, etc. of our randomly generated district mappings are contiguous our generation will take a very, very long time. Thus, instead of a truly random model, we use a Markov Chain Monte Carlo method to approximate random contiguous districting samples of equal size (or at least as equal as possible). We start with n districts, and we randomly change out pieces of each district with pieces of other districts as the districts appear to walk around each other when visualized. We do this many, many times and then take all the districts that the chain generated as our random sampling of all possible electoral district drawings. The strength of this approach, as we will show later, is that as the number of steps in the Markov Chain increases our sampling gets closer and closer to a random sample of all contiguous districts of the same size. This approach was introduced in 2014 by Fifield, Higgins, Imai, and Tarr [6]. However, their approach uses geographical compactness and other subjective factors. Our approach is far simpler because the only constraints on our model are the objective factors of equal size and contiguity. In our model, districts only must have roughly equal populations and connected interiors.

2.2 North Squarolina

We have been tasked with evaluating the redistricting proposals of a hypothetical state, North Squarolina. North Squarolina consists of 36 voting blocks, arranged in a 6x6 grid, with each voting block containing the same number of voting citizens and consistently voting unanimously for the same party. The voting behavior is illustrated in (V). Because of this consistent behavior, we can assume that Gerrymandering is possible: one must be able to predict voting behavior in order to Gerrymander. We have been asked to evaluate three different redistricting proposals labeled (A), (B), and (C) below. We investigate in the results section of this paper if any of them appear to be Gerrymandered in favor of either Redublicans or Bluemocrats. Henceforth, we refer to Redublicans as "Red" and Bluemocrats as "Blue."



(V) (A) (B) (C)

2.3 What About North Carolina?

Later in the paper we will also apply our method to the real state of North Carolina to see whether or not it was Gerrymandered in 2012.

3 Model

3.1 Model Introduction

The goal of our model is to produce a large number (on the order of hundreds of thousands) of districtings, as randomly as possible, in order to obtain a highly representative sample of the space of all possible districts. By examining this very large sample, we can determine the probability of a given districting's voting outcome in the space of all voting outcomes for a particular voting arrangement. In other words, we can determine the probability of a given party winning x seats based on the arrangement of voting blocks. If a particular districting is a statistically significant outlier and produces voting outcome with a low probability, then the districting is likely biased. In the case of North Squarolina, if a districting scheme just so happens – with a .00000001% chance – to elect more Redublicans than almost any other districting scheme, we can safely say that the district is Gerrymandered.

We begin by developing a basic model of random district generation and then improve iteratively. Our first iteration relies on random selection of voting blocks into districts in order to generate districts. Our second iteration of the model accounts for the fact that districts are

legally required to contain the same number of voters. Our third iteration ensures that the contiguity of districts is preserved through the use of a Markov Chain Monte Carlo (MCMC) random walk. We then demonstrate that the districtings obtained via the Markov Chain closely approximate randomly generated districtings. All algorithms implemented in Mathematica.

3.1.1 Mathematical Definitions

Here we formally define the terms we use in the rest of the paper. We represent a given voter arrangement in a grid graph as V (you can think of this as an $m \times n$ matrix, but computationally this is an array). Each node in V represents a voting block. V is populated by values describing the difference between votes for party A and votes for party B in historical voting data. If party A won unanimously in the voting block the value of the block would be 1, if B won unanimously the value of the block would be -1, and a tie would be 0. We represent districting in a grid graph as D (you can also think of this as an $m \times n$ matrix). Each districting includes g districts. The ith node (or ijth entry if you're thinking matrices) in D is a voting block corresponding to the ith node and voting block in A, so both arrays describe the same geography. Each node in D consists of a value taken from the list of all "grouping values" $L = \{l_1, l_2 ... l_g\}$, where each grouping value l_i corresponds to a particular district. There are g districts, and each district is a set of nodes in D with the same value l_i. The voting outcome for a district with grouping value l_i is then obtained by summing all nodes in V corresponding to grouping value l_i in D. For each district, if the sum of V node values is greater than 0, then Party A won the seat. If it is less than 0, then Party B won the seat. This procedure can be performed across all districts to obtain a total number of seats won for Party and Party B respectively.

In the case of North Squarolina, the outcome of voting in any voting block is assumed to be unanimous and to have equal populations. Hence, all nodes in V can be normalized and are equal to either either -1 or 1.

3.2 Random Districts

The first method we develop is random generation of districtings for North Squarolina. In this method we independently assign each voting block to a random district. Mathematically, this means that each node in D (representing a particular voting block) is randomly assigned a grouping value, as shown in Algorithm 1. The district assigned to a given voting block is random, and thus district sizes are generally far from equal. We term this district generation algorithm the Random Districts (RD) Algorithm. Note that functions not defined here are pre-defined in Mathematica, the scientific computing language we used, or are implemented trivially. Implementation of this algorithm (and all other algorithms mentioned) are in the code appendix.

The *outcomes* function determines the voting outcome for a district whose voting blocks have grouping value l_i through the method outlined in the mathematical definitions section. We calculate the outcome of the election as a function of both the district arrangements and the voting block values.

Algorithm 1 Random Districts Algorithm

procedure RD(V)
  for $d_i \in D$ **do**
    $d_i \leftarrow rand(L)$ ▷ Randomly assign each block one of the g grouping values in L
  for $l_i \in L$ **do**
    outcome for district $i \leftarrow outcomes(l_i, V)$ ▷ Get voting outcome for each district
  Finally, we sum up the outcomes in every district to get the full election result

3.3 Random Districts of Equal Size

An issue with the randomly generated districts obtained via Algorithm 1 is that they are not all the same size. When using the algorithm, it is likely that districtings will be generated where each district has a different number of voting blocks, and this violates the "equal size" constraint. The votes of of people in districts with fewer voting blocks will therefore have greater impact, as these individuals have more power in selecting which party wins the seat in their district than individuals in a district where there are more voting blocks. Consequently, we modify the RD algorithm to generate random districts of equal size. This is shown in algorithm 2. We term this algorithm the Equal-Size Random District (ESRD) algorithm.

Algorithm 2 Equal-Size Random Districts Algorithm

procedure ESRD(V)
  $\{D_1, D_2 ... D_g\} \leftarrow partition(D)$ ▷ Randomly partition D into g equal sized groups
  for $D_i \in \{D_1, D_2 ... D_g\}$ **do**
    for $d_j \in D_i$ **do** ▷ Assign every voting block in the D_i group to district l_i
      $d_j \leftarrow l_i$
  for $l_i \in L$ **do**
    outcome for district $i \leftarrow outcomes(l_i, V)$ ▷ Get voting outcome for each district
  Finally, we sum up the outcomes in every district to get the full election result

The fundamental flaw in the ESRD algorithm is that it is not able to intentionally generate contiguous districts. Though consistent sizes, the districts generated through this algorithm are spatially random. Therefore, voting blocks in the same district will likely be disconnected from each other. One solution to this is to repeatedly generate districtings with the ESRD algorithm until a districting with contiguous districts is obtained. But this is highly computationally inefficient. For a real-life sized state it takes an impossible number of trials for ESRD to generate even one contiguous districting scheme. Evidence of this inefficiency is included in the code appendix.

3.4 Contiguous Districts with Markov Chain Monte Carlo

In order to generate a near-random distribution of districtings, we use a Markov Chain Monte Carlo (MCMC) algorithm. However, in the random walk, we limit possible changes in the districting to ones that 1) preserve contiguity and 2) make the smaller districts larger (thus letting us converge to districts of equal size). We use n iterations and take an initial contiguous

districting D_0 represented as a graph. Our algorithm will eventually generate districts of equal size regardless of whether districts in our initial condition D_0 are equal sized.

Algorithm 3 Markov Chain Monte Carlo Algorithm

procedure MCMC(D_0, V, n)
  $D \leftarrow D_0$ ▷ Initialize district map with initial seed
  while $i < n$ **do**
    $D_s \leftarrow sizemin(D_1...D_g)$ ▷ Pick out a smallest district
    $n \leftarrow random\_neighbor(D_s)$ ▷ Find location of a neighboring voting block
    if $keeps\_contiguity(d_n \leftarrow l_s)$ **then** ▷ If taking voting block preserves contiguity
      $d_n \leftarrow l_s$ ▷ Then take it into the smallest district
    $i \leftarrow i + 1$
    for $l_j \in L$ **do**
      outcome for district $i \leftarrow outcomes(l_i, V)$ ▷ Get voting outcome for each district
    Finally, we sum up the outcomes in every district to get the full election result

In the algorithm, we choose the smallest district to be D_s (randomly chosen if there are multiple smallest districts) and then randomly give it another voting block that touches it, on the terms that taking the block won't violate contiguity of another region. *Random_neighbor* obtains a random voting block adjacent to smallest district D_s. Additionally, *keeps_contiguity* checks if making a change maintains the continuity of the graph–if assigning the randomly chosen neighboring voting block to the smallest group results in a disconnected district, the change will not be made and the algorithm will go through another iteration. With a large n our algorithm will produce an approximately random sample of election results, n times, and we can sample from these results to detect Garrymandering.

In reality, the MCMC is not truly random. The initial "seed" districting will bias districtings obtained from higher iterations towards the original position. This is reflected in the dependence of the algorithm on D_0. However, the results of the MCMC become very close to random as we increase the count of iterations n. This issue is explored more below.

3.4.1 Approximation of Randomness

Figure 1 depicts the state of the Markov chain at increasingly high values of i in algorithm 3. The figure depicts the frequency that voting blocks were assigned to a specific district (i.e., the frequency with which they had a certain grouping value) in a single run of the Markov Chain. Black is O (never passed over), white is 1 (always passed over), and grey is somewhere in the middle. If we see that over time the frequency of each voter block in the Markov Chain becomes roughly equal, then this means that each voter block has been assigned to the specified district (passed over by the Markov Chain) a roughly equal number of times. A truly random sample would have the same property–a truly random sample would be the same shade of grey in all squares–and this is the key property of randomness that our Markov Chain can approximate.

As shown in the $i = 0$ case, one district begins in the top left, as the Markov chain has not begun yet. We can see in the $i = 10,000$ case, we obtain a visibly biased distribution of frequency with which the Markov chain has passed over different voting blocks, with blocks in the upper right being significantly more likely to be passed over. However, by the $i = 90,000$ case, the amount that each voting block has been passed over by the chain has gotten much

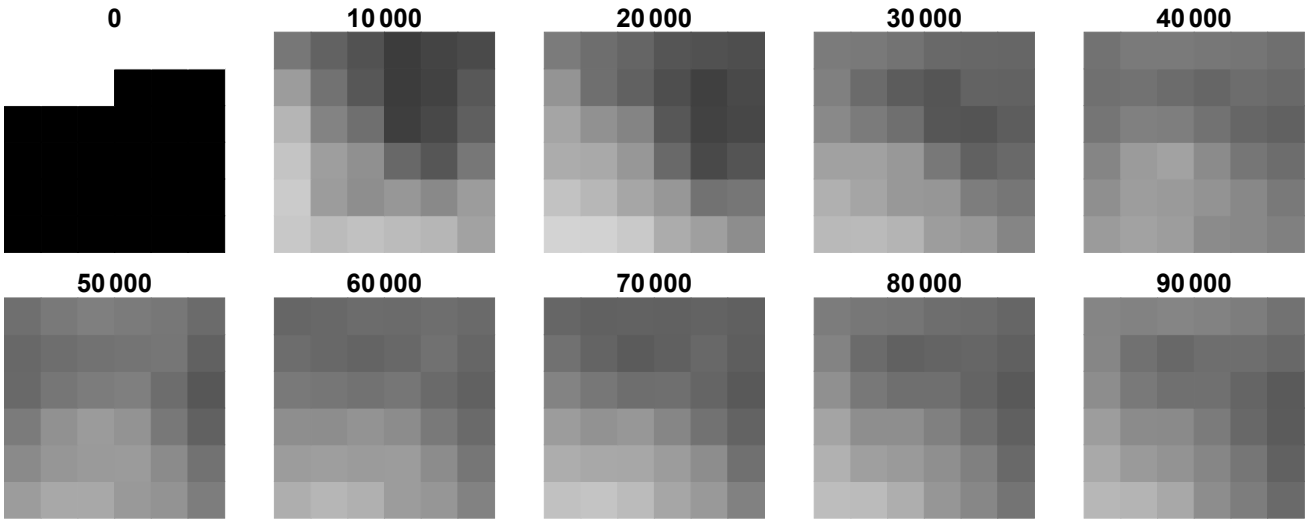


Figure 1: MCMC Convergence - 100,000 Runs

more even. This suggests that our 100000 iterations is enough to ensure that our districts are effectively random, and that the initial condition will not have too strong an effect.

3.4.2 Random Seeding to for A Better Approximation

Further improvement in the pseudo-random behavior of the Markov Chain can be obtained by seeding the initial position of the chain randomly. Doing this is equivalent to the procedure described above, except for the first m iterations (where $m < n$) voter outcomes are not recorded. See the Code Appendix for further details.

3.4.3 Better Random Seeding to for An Even Better Approximation

To seed our MCMC algorithm, we just need to give it any districting where all districts are contiguous. By repeatedly applying MCMC, any deviations in the sizes of the districts will be evened out. However, if we generate a seed graph with districts of wildly different sizes, it will take longer for it to converge on an equal-area districting.

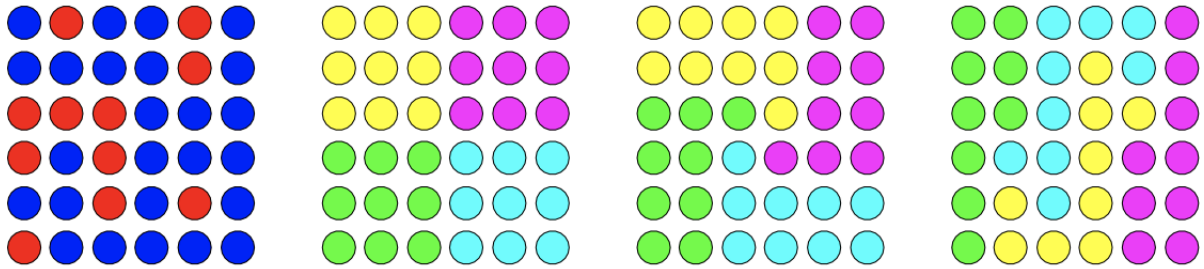
We developed an algorithm for getting around this. We take the graph that we used to represent the adjacency of voting blocks (a grid graph in the case of North Squarolina), we remove all of the edges, and then we repeatedly add edges back to the smallest connected component until we have the desired number of connected components (see Code Appendix for details). This leaves us with a districts of similar size that can be fed to MCMC.

4 Results

4.1 Do Any of the Proposals Gerrymander North Squarolina?

We look at the number of districts that will be won by red under each districting proposal. If this number is abnormally low, we might find evidence of a blue-driven Gerrymander, and if

the number is abnormally high, we might find evidence of a red-driven Gerrymander. Assuming that voting patterns stay the same in North Squarolina, like they have in the past, red will win 0 districts under (A), 1 district under (B), and 2 districts under (C).



4.1.1 Using Random Districts

Our Random Districts algorithm is not terribly accurate because it allows for districts that are wildly different sizes. An example districting obtained by the RD algorithm is shown below (Figure 2).

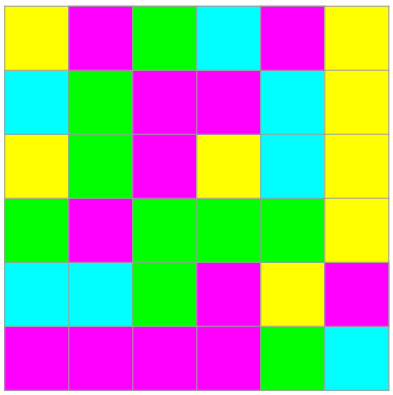


Figure 2: North Squarolina, Random Districts

It also allows for non-contiguous districts. In fact, when we generated 100000 completely random districtings, none of them contained only contiguous districts.

According to the sample given by the Random Districts generated histogram (Figure 3) for the number of voting districts won by red, neither 0, 1, nor 2 votes for red appears extremely unlikely. This model gives us almost nothing. Red seems to have a likelihood of winning 2 districts that is over 10%, which seems high given real-life constraints.

4.1.2 Using Random Districts of Equal Size

Our Random Districts of Equal Size algorithm is far more accurate because it restricts districting to maps with districts of the same sizes. An example districting obtained by the ESRD algorithm is shown in Figure 4.

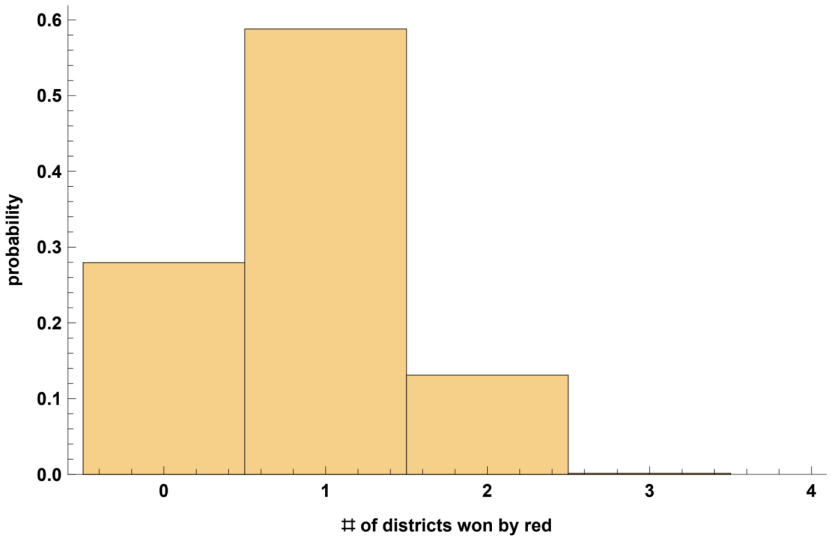


Figure 3: North Squarolina, Random Districts Histogram

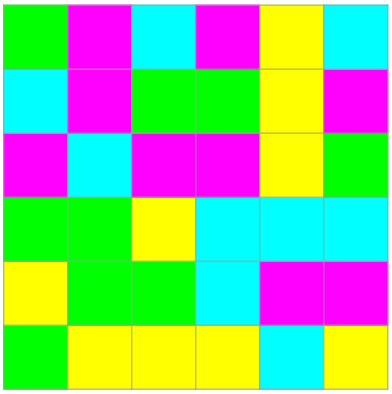


Figure 4: North Squarolina, Equal-Sized Random Districts

It is, however, not ideal because it still generates mostly noncontiguous (and therefore unrealistic) districting schemes. Out of the 100000 districtings we generated with random districts of equal size, only a few hundred managed to win 2 districts for red. According to this sample given by the Random Districts of Equal Size generated histogram the probability of red winning is 0.00296 (around 3 in 1000). This appears to suggest that voting patterns that give red 2 districts are extremely unlikely and that districting scheme (C) is Gerrymandered.

4.1.3 Using MCMC Contiguous Districts

The MCMC generated districts are contiguous, and so they could be valid as real voting districts. Figure 6 shows a random MCMC generated districting.

In Figure 7, we see that contiguous districts are more likely to give a seat to red than the less realistic non-contiguous random districts. Using this histogram, we can see that districting (C) still produces an extremely unlikely election outcome–reinforcing our belief that discricting (C) is probably the result of gerrymandering.

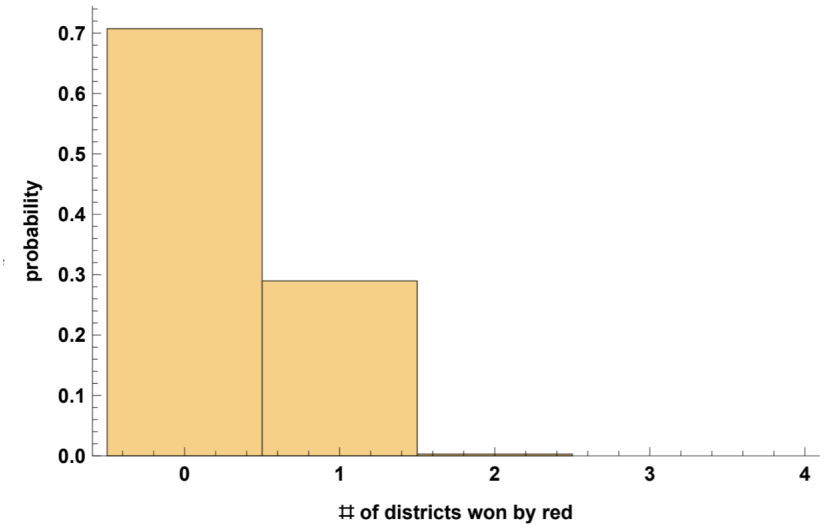


Figure 5: North Squarolina, Equal-Sized Random Districts Histogram

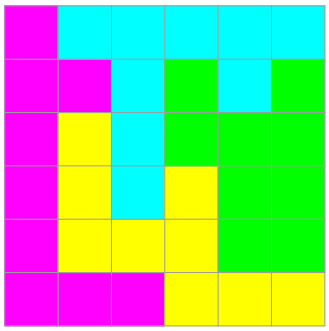


Figure 6: North Squarolina, MCMC Contiguous Districts

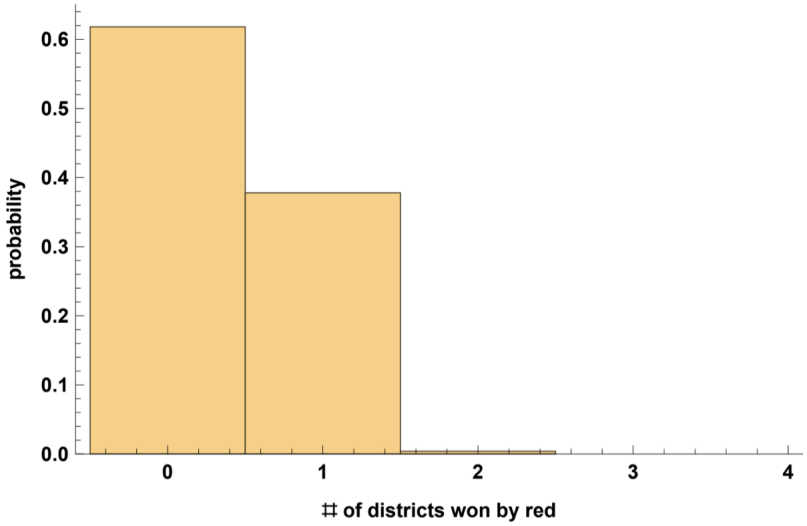


Figure 7: North Squarolina, MCMC Contiguous Districts Histogram

4.2 Is North Carolina Gerrymandered?

Fundamentally, these algorithms burn down to methods for randomly partitioning graphs. We have been partitioning grid graphs to model North Squarolina, but we can just as easily partition any other arbitrary graph. We use North Carolina as an example.

We start by generating the adjacency graph for counties in North Carolina (Figure 8). (See Code Appendix for details.)

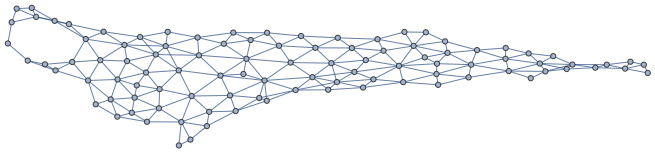


Figure 8: Adjacency graph of North Carolina counties.

There is a smaller unit of voting data than counties: VTDs, otherwise known as precincts. However, VTDs change between elections, and do not have fully unique identifiers, and so we cannot reliably align election results with an adjacency graph. However, we can abstract VTD level data to county level data, and perform the data alignment over counties very easily. We do this with voting data from OpenElections [5].

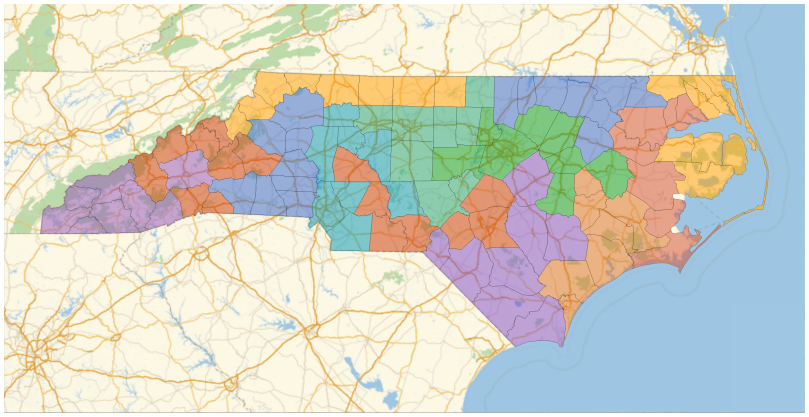


Figure 9: A random districting of North Carolina.

In Figure 9, we generate a random redistricting by applying our MCMC algorithm to the county graph. As before, we can create a large number of random districtings and see how many districts each party is expected to win. Figure 10 shows the distribution of the number of random districts that would be won by republicans. Our voting data is from 2012, when 9 of the 13 congressional districts on North Carolina were won by Republicans. Our random districtings show that this outcome of 9 Republicans winning is the most likely outcome regardless of Gerrymandering.

4.3 Potential Impact of Gerrymandering

Using the model we have developed, we can also assess the potential impact of Gerrymandering for a given voter arrangement. Algorithm 2 can be modified to generate random voter arrangements by substituting V for D in the procedure, substituting a set of party voting differences

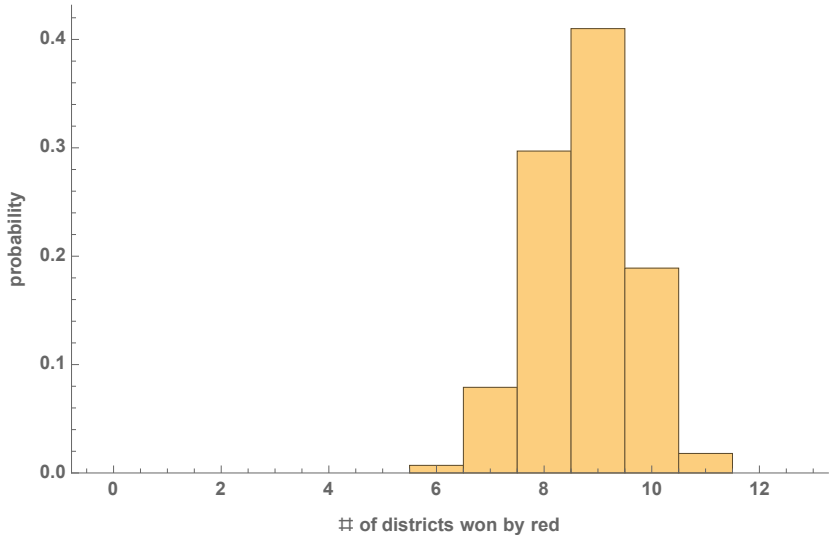


Figure 10: Distribution of the number of districts won by republicans in 10000 random districtings of North Carolina.

$-1, 1$ for L and omitting the part about recording voter outcomes. This modification lets randomly generate a voter arrangement, and it can be modified so that each party wins a specified fraction of the total number of voting blocks. See the Code Appendix for further information. The MCMC algorithm can be applied to all these random voter arrangements to assess the average number of voters won and the maximum number of possible voters won by a given party using a highly biased districting. The results of the procedure are below, applied to North Squarolina.

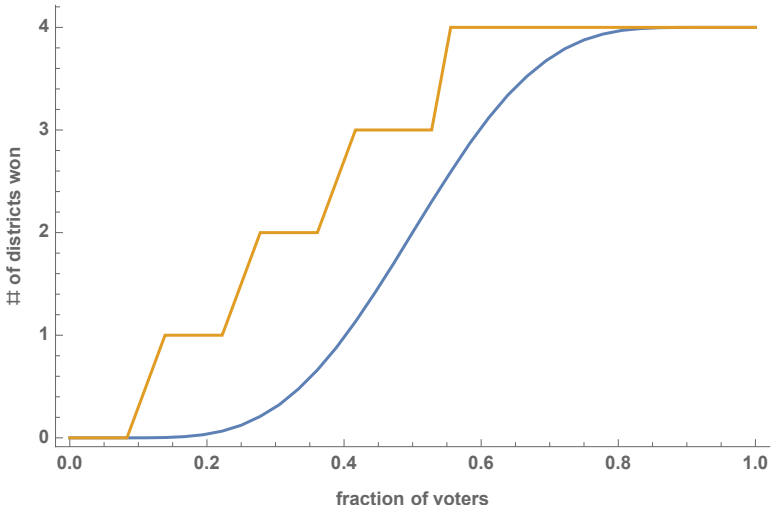


Figure 11: Seats Won vs. Fraction of Voters Won

Plotted in Figure 11 are both the average values for the fraction of voters by party A and the maximum number of districts that could possibly be won by party A depending on the districting. In other words, the gold line shows the extreme-case possibility of Gerrymandering in A's favor, whereas the blue line shows the average votes won by A across the the random

possible districtings generated by Algorithm 3. As expected, at extremely low values of vote-getting for A, Gerrymandering can't assure that A will obtain any seats. At extremely high values of A, Gerrymandering in A's won't yields more seats than average because the average number of seats won by A is the maximum number of seats won by A. (That said, it would be worth it to explore how Gerrymandering can improve Party A's confidence in securing those seats).

We can also apply this calculation to visualize how specific proposals deviate from the average number of seats won for a given voting arrangement. Applying this to the districting proposals and the voter arrangement in North Squarolina, we obtain Figure 12.

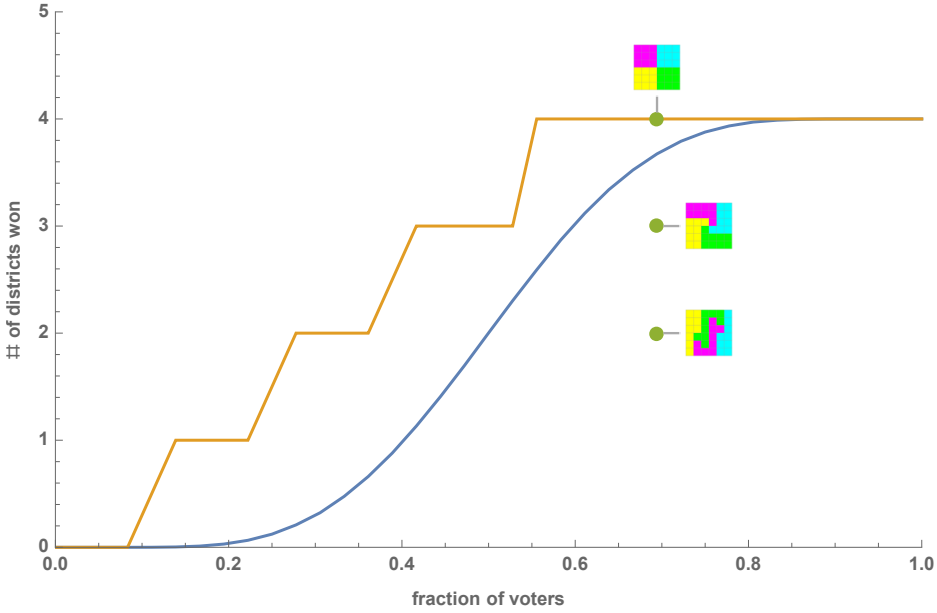


Figure 12: Seats Won vs. Fraction of Voters Won - Depicts Proposals

5 Strengths and Weaknesses

- Our Markov Chain Monte Carlo Method may not generate a good enough attempt at a perfectly random set of samples–our approximation may be flawed. Although the chain appears to eventually "flatten out" and distribute the probability of being in any district uniformly across all blocks, it may still unfairly favor some districting patterns over others. For example, our construction of our Markov Chain may be biased towards taking certain types of districting patterns over others–we can guess that it's not, but the truth is that we don't know. Maybe an outlier that occurs with 0.001 frequency in our MCMC sample–and looks like Gerrymandering–but it actually occurs with 0.01 frequency (a much less convincing number) in real life because it's a strange case that's hard for the chain to reach.

- In our simplified model we assume unanimously voting blocks. However, our model easily generalizes to any form of voting behavior, since our random district generation is meant to be unbiased and therefore independent of voting behavior. Just like in the simplified

process described in this paper, we generate our random districts with our MCMC method independently of voting behavior, and then use this generated sample to inform us of low-probability events that favor one party over the other.

- We assume that the people who draw district lines are able to (or at least believe they are able to) predict future voting patterns based on past patterns. Obviously, voting patterns often can be predicted based on past consistency, although not quite as easily as the case in North Squarolina. And when looking for most cases of Gerrymandering, we must assume relatively consistent voting patterns, and assume it is the districting map–not the election itself–that determines most of the outcome. Sometimes this is not necessarily the case. But since the act of Gerrymandering itself is flawed by the same assumption as our model, we don't see this as a flaw in our model.

5.1 Using $\sqrt{2\epsilon}$ to Bound Our Results for Mathematical Rigor

A recently published paper explains that as long as our Markov Chain is reversible and can reach its steady-state, it will well-approximate a random sample as given by the theorem below, which we quote verbatim from the paper [7]:

Theorem 5.1. *Let $M = X_0, X_1, ...$ be a reversible Markov chain with a stationary distribution π, and suppose the states of M have real-valued labels. If $X_0 \sim \pi$, then for any fixed k, the probability that the label of X_0 is an $\epsilon -$ outlier from among the list of labels observed in the trajectory $X_0, X_1, X_2, ..., X_k$ is at most $\sqrt{2\epsilon}$.*

In our justification for the Markov Chain Monte Carlo method as a good approximation for randomness we talk about how the chain flattens out into grey and begins to well-approximate a seemingly random sample, since the probability of any voting block being in any district, given n districts, gets closer and closer to $1/n$ as we take our chain further and further. This may not indicate a perfect approximation of randomness, but this does signal that our chain is consistently bouncing around all over and reaching what appears to be a dynamic steady state. Since we can see strong evidence of our chain reaching a dynamic steady state–we know for certain that it will eventually reach the state after a certain number of iterations–and we know that the probability of reaching any specific districting scheme with regions of equal size using our model is > 0, it then holds that we can use the theorem to more rigorously quantify outliers.

If, when using the sample for North Squarolina generated from MCMC Contiguous Districts given by the histogram in the last section, we find 189 out of 100k samples where red win two districts, this would seem to indicate that the probability of red winning 2 districts is 0.00189. Using the much more mathematically rigorous bound given by the theorem above, however, we can find that the upper bound for the true probability is $\sqrt{2 \times 0.00189} = 0.0615$, which is much higher than our actual result. Although we can believe that the probability that red wins two districts is almost definitely less than one half of one percent, we can only say with mathematical certainty that the probability is less than about 6%. In this example, North Squarolina is small enough where the the percentage produced by this rigorous bound is too large to mean anything of much significance, but applying our model to larger regions will yield small and inarguable upper bounds. Studies such as the one referenced above that used applied similar MCMC methods on larger areas such as states generated probabilities of 1×10^{-8} or smaller, which are small enough estimations to generate a significantly small, mathematically

defensible upper bound for the probability of an event happening. Many times we are able to say with mathematical certainty that the probability of creating a system that randomly favors one party over the other cannot be greater than 1 in 1000, which would be nearly irrefutable evidence of Gerrymandering.

6 Comparing Our Model to the Efficiency Gap

6.1 Defining the Efficiency Gap Method

The efficiency gap is a metric that tries to quantify the partisan bias of a districting scheme based on comparing historical "votes wasted" per party. For example, all votes from the losing party in a 49% to 51% loss would be "wasted" because they had no impact on winning the election–the party with 51% of the vote won–and in a landslide victory votes are wasted because they are more than what's needed to win. According to proponents of the efficiency gap, if one party "wastes" more votes than the other, the districting scheme is biased because one party's votes counted more than the other's. In a given district, given that party A secures a votes and party B secures b votes, we calculate the pair of values (w_a, w_b) to respectively be number of votes wasted by party A and number of votes wasted by party V. We can compute these values with the function $w(a, b)$:

$$w(a,b) = (w_a, w_b) = \begin{cases} (a - (\lfloor \frac{a+b}{2} \rfloor + 1), b) & a \geq b \\ (a, a - (\lfloor \frac{a+b}{2} \rfloor + 1)) & b < a \end{cases}$$

If party A loses a district (if $a < b$), then all a of its votes are wasted in that district. But in the case that party A wins the district, then the number of wasted votes is the difference between votes for A and the votes that A needed to win the district. The number of votes needed to win the district is equal to the floor of the total number of votes plus one. For instance, in a district where 100 people voted between party A and party B, the total votes are obtained through $a + b = 100$, and the number of votes either party needs to win is $\lfloor \frac{100}{2} \rfloor + 1 = 50 + 1 = 51$.

Examining an entire state composed of several districts, then (a, b) represents individual districts in the space of all districts d. Examining the entire state, the efficiency gap in favor of party A is defined to be the difference between B's wasted votes and A's wasted votes across every district, weighted by the inverse of the total number of votes:

$$e_b(w_1, w_2) = \frac{\sum_{(a,b)\in d} w_b - w_a}{\sum_{(a,b)\in d} a + b}$$

The efficiency gap in favor of party B is the negative of the efficiency gap in favor of party A. This formula measures the difference between votes wasted by party A and party B, then normalizes the difference by dividing by total votes.

6.2 Why the Efficiency Gap is Not Enough

The efficiency gap is simply an evaluation of how districting favors one party over another. By counting the difference in wasted votes divided by total votes it counts the "losses" incurred by the unfavored party. However, without any acknowledgement of probability the efficiency gap is

not useful as an indicator for Gerrymandering–naturally, given a certain voting arrangement, it might be more likely that one party gets favored over the other. For example, in the case where 30% of voters favor one party, but they are scattered uniformly about the other 70%, most districtings will largely favor the majority party because most districtings will contain around a 70% to 30% balance in each district. However, the resulting efficiency gap of 10% would, to an efficiency gap believer, seem to scream bloody murder. *Absolutely nothing abnormal happened in the drawing of the districts,* and yet according to the efficiency gap metric we have a case of Gerrymandering. Any state with an efficiency gap of $> 7\%$ has been condemned in the news.

7 Conclusion

A probability-driven approach is essential to quantifying Gerrymandering in a court of law. A metric like efficiency gap often can be the result of a coincidence such as obvious geographical boundaries or something like the natural 70% vs 30% example outlined above, as opposed to actual Gerrymandering. With our approach, defining situations that favor one party over the other in terms of probability, we can define Gerrymandering as a very low probability event that favors one party over the other. If we find an action that favors one party over the other with very low probability, we most likely found an action that favors one party over the other deliberately, and deliberately favoring one party over the other is Gerrymandering by definition.

References

[1] "Gerrymander." *LII / Legal Information Institute*, Cornell University Law School, www.law.cornell.edu/wex/Gerrymander.

[2] Jacobs, Tom. "The Policy Consequences of Partisan Gerrymandering." Pacific Standard, 4 Oct. 2017, psmag.com/news/the-policy-consequences-of-partisan-gerrymandering.

[3] Brown Division of Applied Math. "Brown Mathematical Contest for Modeling 2018 Problems." https://drive.google.com/file/d/1RhXsuI2IPCn4JJin54AOPg4Zs-JNeDZa/view.

[4] "Gill v. Whitford." *Wikipedia*, Wikimedia Foundation, 7 Aug. 2018, en.wikipedia.org/wiki/Gill_v._Whitford.

[5] "Home — OpenElections: Certified Election Results. For Everyone," www.openelections.net.

[6] Fifield, Benjamin, et al. *A New Automated Redistricting Simulator Using Markov Chain Monte Carlo.* Harvard University, 2017, https://imai.fas.harvard.edu/research/files/redist.pdf.

[7] Pegden, Wes. "Detecting Gerrymandering with Markov Chains." Carnegie Mellon University, 2018, www.cmu.edu/news/stories/archives/2018/january/images/MATH-18-114_Math%20Newsletter_Gerrymandering.pdf.

8 Code Appendix

Go here to really understand our model and dive into the algorithms.

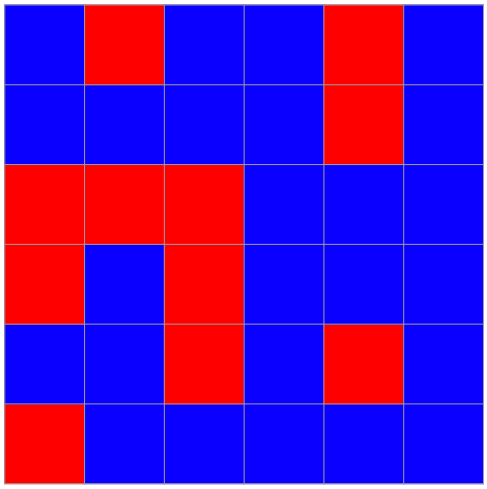
Initialization

Set the voting pattern of census blocks in North Squarolina:

In[1]:=
```
votes = Flatten@{{1, 0, 1, 1, 0, 1}, {1, 1, 1, 1, 0, 1},
    {0, 0, 0, 1, 1, 1}, {0, 1, 0, 1, 1, 1}, {1, 1, 0, 1, 0, 1}, {0, 1, 1, 1, 1, 1}};
```

In[ ]:=
```
ArrayPlot[Partition[votes, 6],
 ColorRules → {0 → RGBColor[1, 0, 0], 1 → RGBColor[10 / 255, 0, 1]}, Mesh → All]
```

Out[ ]=



Random Districtings

Completely Random

The next kind of random districtings we consider are those made by randomly assigning each census block to a district. We represent some districting by a list of lists, where each inner list represents a district and contains the indices of the census blocks that it includes:

In[ ]:=
```
completelyRandomDistricting := Values@PositionIndex@RandomInteger[{1, 4}, 36]
```

In[ ]:=
```
completelyRandomDistricting
```

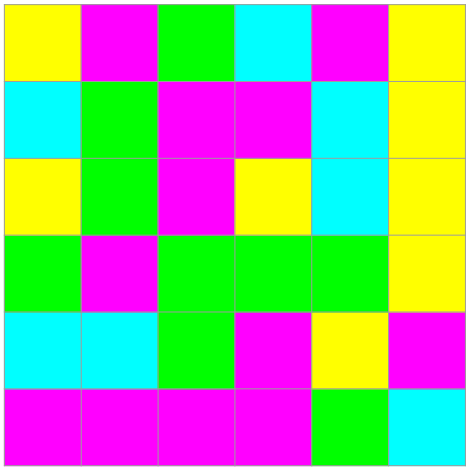
Out[ ]=
```
{{1, 3, 6, 10, 11, 24, 26, 30, 31}, {2, 4, 5, 15, 21, 23, 27, 36},
 {7, 8, 12, 20, 22, 25, 29, 32, 34}, {9, 13, 14, 16, 17, 18, 19, 28, 33, 35}}
```

Next, we can create a function for visualizing districtings:

In[2]:=
```
plotDistricting[d_, w_: 6] := ArrayPlot[Partition[
   Normal@SparseArray[Catenate@MapIndexed[Thread[#1 → First[#2]] &, d]], w],
  ColorRules → {1 → Yellow, 2 → Magenta, 3 → Green, 4 → Cyan},
  Mesh → All, ImageSize → 250]
```

In[ ]:=
```
plotDistricting[completelyRandomDistricting]
```

Out[ ]=

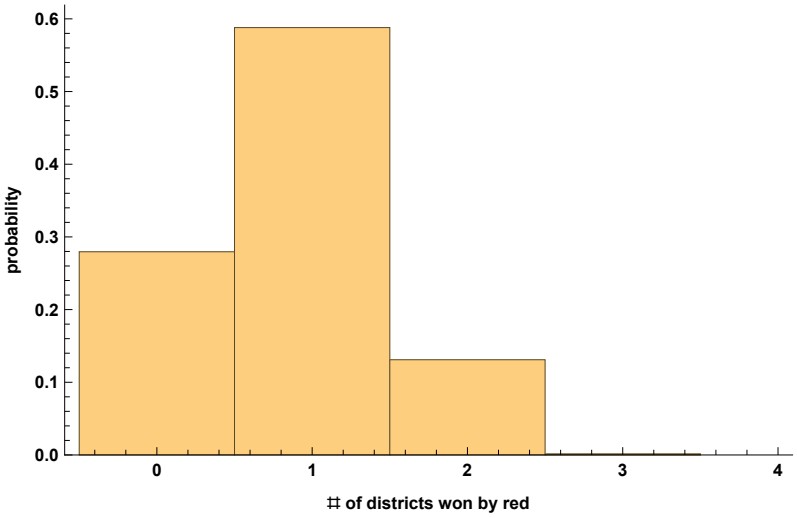


Finally, we can calculate the number of districts that each part wins on average with these noise districtings in North Squarolina:

In[ ]:=
```
completelyRandomDistrictingSimulation = Table[
   Total@Boole[Total[votes[[#]]] < 5 & /@ completelyRandomDistricting], 100 000];
```

In[ ]:=
```
Histogram[completelyRandomDistrictingSimulation,
 {1}, "Probability", Frame → {{True, False}, {True, False}},
 FrameLabel → {"# of districts won by red", "probability"},
 PlotRange → {{-0.5, 4}, All}]
```

Out[ ]=



However, this simulation is not terribly accurate because it allows for districts that are wildly different size. It also allows for non-contiguous districts. In fact, we can see that out of 100 000 completely random districtings, none of them contain only contiguous districts:

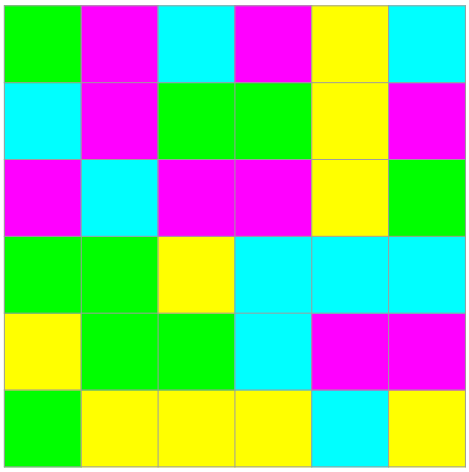
```
In[•]:= Select[Table[Image@RandomInteger[{1, 4}, {6, 6}], 100 000],
          (i ↦ ! AnyTrue[Range[4], Max@MorphologicalComponents[
                  Binarize[i, {#, #}], CornerNeighbors → False] > 1 &])]
Out[•]= {}
```

Random Equal-Sized

The next kind of random districtings we consider are those made by randomly selecting 9 blocks to be in district 1, another 9 in district 2, and so on. These are like the completely random ones, except all districts are the same size:

```
In[15]:= randomDistricting := Partition[RandomSample[Range[36]], 9]
In[•]:= randomDistricting
Out[•]= {{5, 11, 7, 20, 19, 12, 8, 35, 16}, {25, 23, 4, 27, 31, 17, 1, 15, 33},
         {32, 36, 28, 30, 9, 2, 21, 29, 24}, {26, 22, 10, 3, 13, 14, 34, 18, 6}}
In[•]:= plotDistricting[randomDistricting]
```

Out[•]=

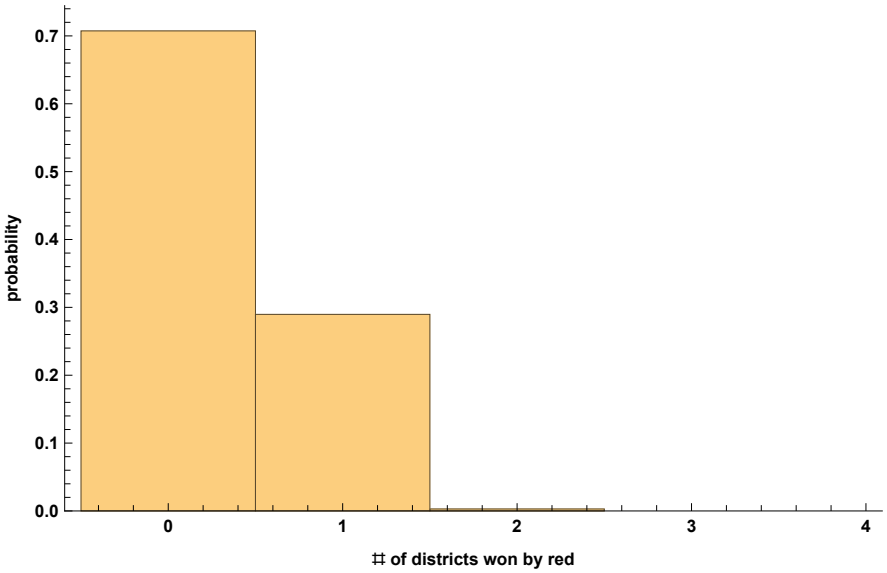


Finally, we can calculate the number of districts that each part wins on average with these noise districtings in North Squarolina:

```
In[•]:= randomDistrictingSimulation =
          Table[Total@Boole[Total[votes[[#]]] < 5 & /@ randomDistricting], 100 000];
```

```
In[•]:= Histogram[randomDistrictingSimulation, {1},
         "Probability", Frame → {{True, False}, {True, False}},
         FrameLabel → {"# of districts won by red", "probability"},
         PlotRange → {{-0.5, 4}, All}]
```

Out[•]=



Out of the 100 000 districtings we generated, only a few hundred managed to win 2 districts for red:

```
In[•]:= KeySort@Counts[randomDistrictingSimulation]
Out[•]= ⟨| 0 → 70 738, 1 → 28 966, 2 → 296 |⟩
```

We can also calculate the expected value of the number of districts won by red:

```
In[•]:= N@Mean[randomDistrictingSimulation]
Out[•]= 0.29558
```

We can see that this is much lower than the directly proportional number of seats we would expect for red:

```
In[•]:= N@4 * (36 - Total[votes]) / 36
Out[•]= 1.22222
```

Markov Chain Monte Carlo (MCMC)

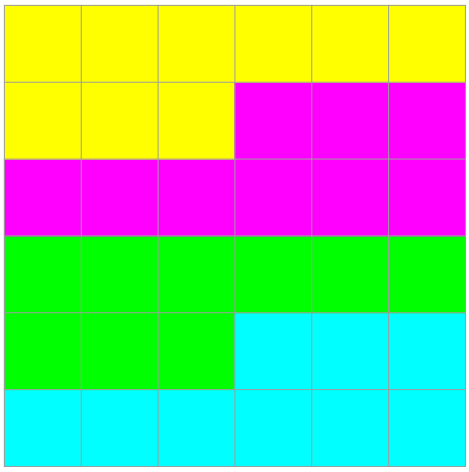
We define a function for randomly mutating districts. It takes partitions of the census blocks (a districting) and a graph that shows the adjacency of the census blocks, and mutates it a single step, preserving contiguity and approximately preserving area:

In[3]:=
```
randomlyMutateDistricts[partitionsI_, g_] :=
 Block[{partitions = partitionsI, partition, otherPartition, vertex},
  partition = RandomChoice[Position[#, Min[#]] &[Length /@ partitions]][[1]];
  vertex = RandomChoice[Complement[
     AdjacencyList[g, partitions[[partition]]], partitions[[partition]]]];
  otherPartition = Position[partitions, _?(MemberQ[vertex]), {1}][[1, 1]];
  If[Length[partitions[[otherPartition]]] > 1 && ConnectedGraphQ[
     Subgraph[g, DeleteCases[partitions[[otherPartition]], vertex]]],
   partitions = MapAt[DeleteCases[#, vertex] &, partitions, otherPartition];
   AppendTo[partitions[[partition]], vertex];
  ];
  partitions
 ]
```

As an example, here is some arbitrary districting:

In[ ]:=
```
plotDistricting@Partition[Range[36], 9]
```

Out[ ]=

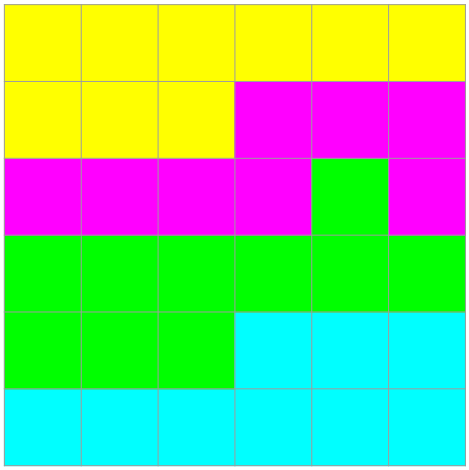


And here we randomly mutate it:

In[ ]:=
```
g = GridGraph[{6, 6}];
```

In[ ]:=
```
plotDistricting@randomlyMutateDdistricts[Partition[Range[36], 9], g]
```

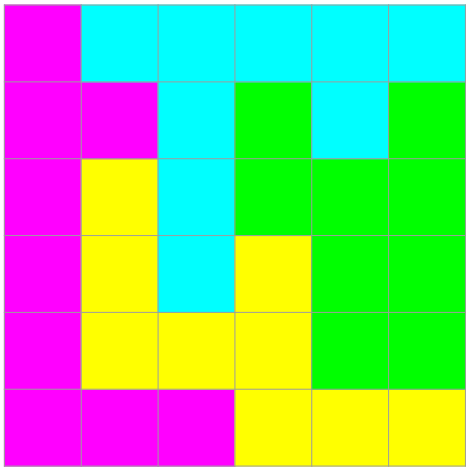
Out[ ]=



Each application of this mutation is equivalent to one step in a Markov chain. Through repeated application, we get a more random districting:

In[ ]:=
```
plotDistricting@
 Nest[randomlyMutateDistricts[#, g] &, Partition[Range[36], 9], 10 000]
```

Out[ ]=

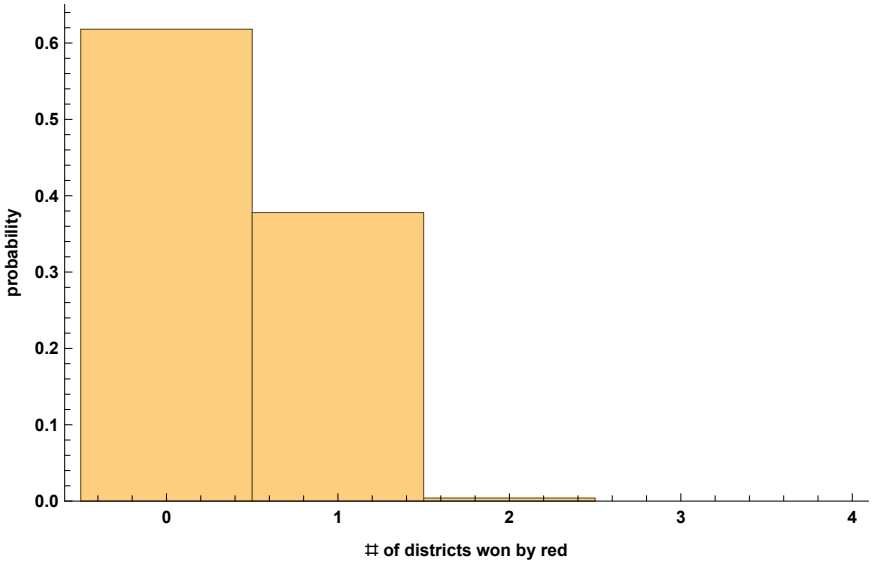


For each random districting we generate, we can calculate the number of districts that red will win:

In[13]:=
```
mcmcSimulation = Total /@ Boole@Map[Total[votes[[#]]] < 5 &,
      NestList[randomlyMutateDistricts[#, g] &, Nest[randomlyMutateDistricts[#, g] &,
        Partition[Range[36], 9], 100 000], 100 000], {2}];
```

In[14]:=
```
Histogram[mcmcSimulation, {1}, "Probability", Frame → {{True, False}, {True, False}},
 FrameLabel → {"# of districts won by red", "probability"},
 PlotRange → {{-0.5, 4}, All}]
```

Out[14]=



In[•]:=
```
KeySort@Counts[mcmcSimulation]
```

Out[•]=
```
⟨| 0 → 62 264, 1 → 37 548, 2 → 189 |⟩
```

In[•]:=
```
N@Mean[mcmcSimulation]
```

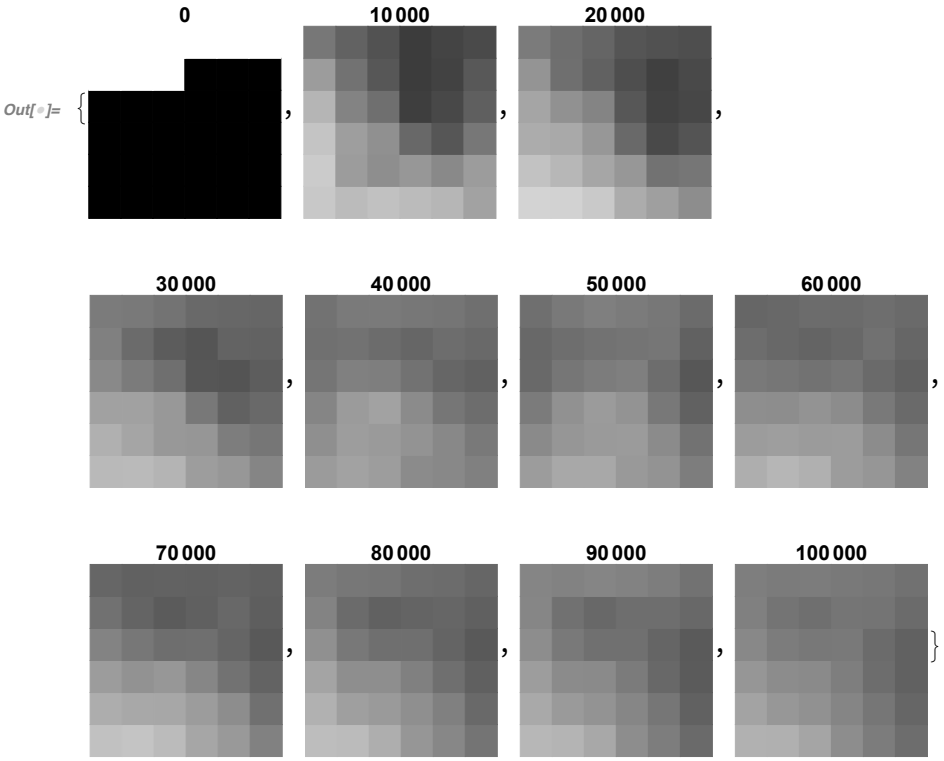
Out[•]=
```
0.379256
```

With this model, we can see that red wins more on average, but it also was less likely to win 2 districts.

Here, we started with a non-random initial condition, and then we mutated for 100 000 steps to get our second initial condition. Once we have our second initial condition, we continue mutating for another 100 000 steps, collecting data. However, we need to confirm that the initial 100 000 steps we mutated for to get the initial condition is enough that there will not be too much of a lingering influence from the non-random initial condition. Here, we start with the non-random arrangement, and then we plot the frequency with which that each cell is in district 1 as we mutate:

```
In[•]:= imgs = (Image@SparseArray[Thread[QuotientRemainder[#[[1]] - 1, 6] + 1 → 1], 6] & /@
        NestList[randomlyMutateDistricts[#, g] &,
         Partition[Range[6^2], 6^2 / 4], 100 001]);
    accImgs = FoldList[ImageAdd, imgs];
    Table[Show[
      Image@Rescale[# / Total[Flatten[#]] &@ImageData[accImgs[[n]]], {0, 2 / 36 // N}],
      PlotLabel → n - 1, ImageSize → 100], {n, 1, 100 001, 10 000}]
```

Out[•]=



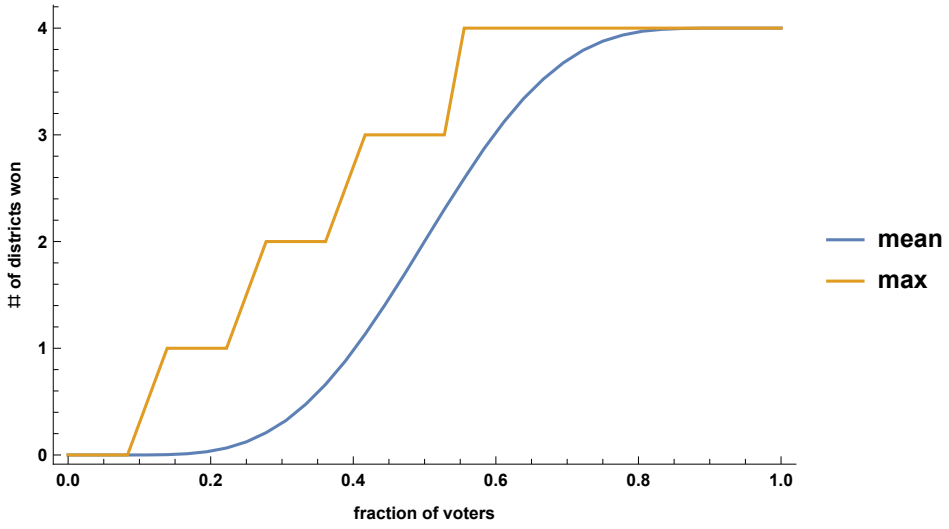
We can see that after 100 000 all configurations are almost equally likely, suggesting that this method for generating random districts works.

We can also use this model to create districts of with random voter arrangements. Here, we create a random voter arrangements with *n* voters, and then compute the average and maximum number of districts that go to each party as a function of *n*:

In[156]:=
```
LaunchKernels[11];
simulationData =
  With[{size = 6},
   ParallelTable[
    {N[n/size^2],
     Module[{createVotes, votes, g, partitions, allDistrictings, outcomes},
      createVotes :=
       RandomSample[Join[ConstantArray[1, n], ConstantArray[0, size^2 - n]]];
      g = GridGraph[{size, size}];
      partitions = Nest[randomlyMutateDistricts[#, g] &,
        Partition[Range[size^2], size^2/4], 100 000];
      allDistrictings = NestList[randomlyMutateDistricts[#, g] &,
        partitions, 100 000];
      outcomes = Total /@ Map[(votes = createVotes;
           (If[#1 === #2, 0.5, If[#1 > #2, 0, 1]] &@@ Lookup[
                Counts@votes[[#]], {0, 1}, 0]) &/@#) &, allDistrictings];
      N@{Mean[outcomes], Max[outcomes]}
     ]
    },
    {n, 0, size^2}]
  ];
CloseKernels[];
```

In[159]:=
```
ListLinePlot[Transpose[Thread /@ simulationData],
 Frame → {{True, False}, {True, False}},
 FrameLabel → {"fraction of voters", "# of districts won"},
 PlotLegends → {"mean", "max"}]
```

Out[159]=



The shape of this function shows us why Gerrymandering is effective. A small difference in the fraction of voters can quickly change the fraction of districts won. We can also see how the best district can do

significantly better than the average district, showing how far Gerrymandering can take you.

We can also see where the proposed districts fall compared to this curve. Here, we import the proposed districts from the data file:

In[193]:=
```
proposedDdistricts =
  Table[Values@KeySort@PositionIndex@Flatten@Reverse@Transpose@Partition[
          Round@Transpose[Rest@Import[FileNameJoin[{rawDataDirectory,
                  "mixon_mcm_data.xlsx"}]][[1]]][[n]], 6], {n, 4, 6}];
```

Then, we calculate the number of districts that each proposal will win for the reds:

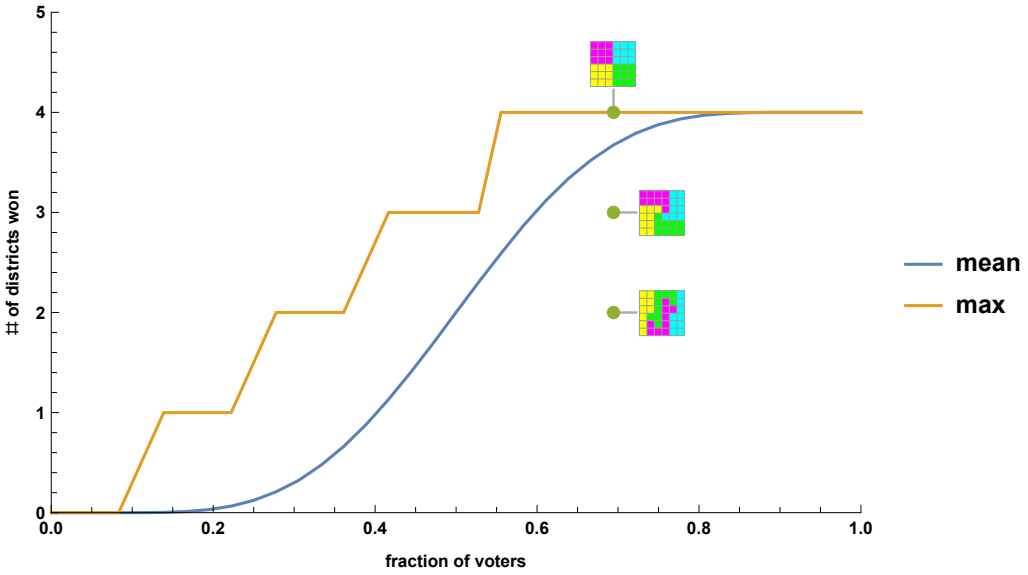
In[202]:=
```
Total /@
 Map[If[#1 === #2, 0.5, If[#1 > #2, 0, 1]] &@@ Lookup[Counts@votes[[#]], {0, 1}, 0] &,
  proposedDistricts, {2}]
```

Out[202]=
```
{4, 3, 2}
```

We can place these on the plot from before to see how these compare with averages for districts with the same proportion of red and blue voters:

In[738]:=
```
ListLinePlot[Append[Transpose[Thread /@ simulationData],
  {Callout[{Total[votes] / Length[votes], 2},
    plotDistricting[proposedDistricts[[3]]], Right],
   Callout[{Total[votes] / Length[votes], 3},
    plotDistricting[proposedDistricts[[2]]], Right],
   Callout[{Total[votes] / Length[votes], 4},
    plotDistricting[proposedDistricts[[1]]], Top]}],
 Frame → {{True, False}, {True, False}}, FrameLabel →
  {"fraction of voters", "# of districts won"}, Joined → {True, True, False},
 PlotLegends → {"mean", "max"}, PlotRange → {{0, 1}, {0, 5}}]
```

Out[738]=



We can see that proposal A yields the maximum number of districts for blue, while C yields fewer.

We can also compare with the data from our simulation on North Squarolina's voting arrangement to see how likely it would be to draw districts to reach each of these election outcomes:

```
In[231]:= 100 * N@Counts[mcmcSimulation] / Length[mcmcSimulation]
Out[231]= ⟨| 0 → 61.8074, 1 → 37.7936, 2 → 0.398996 |⟩
```

We can see that A gets the same result as about 62% of random districts, B gets the same result as about 38%, and C gets the same result as a tiny 0.4%.

Random Seeding

Previously, we created our initial conditions for the MCMC case by running our mutation algorithm on some non-random initial condition 100 000 times. We showed that this produces a relatively uniform distribution of districts. However, if we want an even more uniform distribution, or if we want to generate initial conditions on much larger graphs, we need to randomly general initial conditions.

Our mutation algorithm always brings the number of blocks in each district closer together, so we can even just start with a random seed where the districts are unequal in size. All that is important is that they are contiguous. To do so, we take the grid graph that we used to represent adjacency, we remove all of the edges, and then we repeatedly add edges back to the smallest connected components. (Some seeds will lead to states that our MCMC algorithm cannot mutate into equal-area districts, and so we ignore those seeds.)

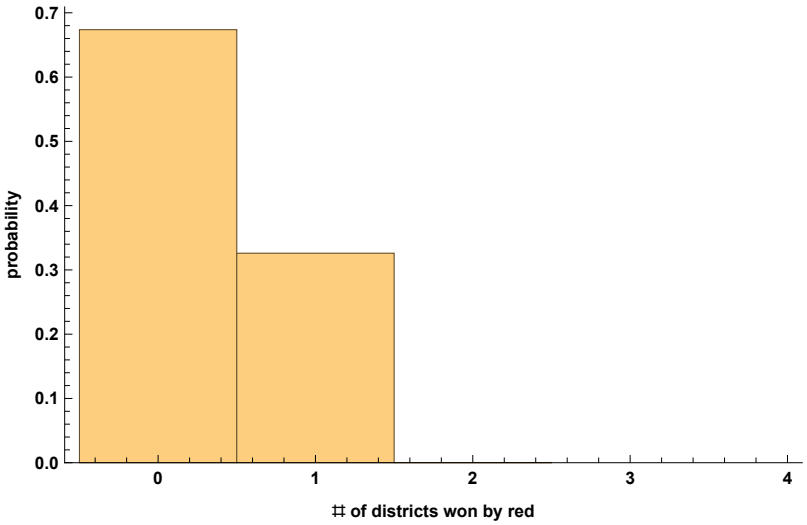
```
In[575]:= seededDistricting[g_, td_: 4, steps_: 1000, maxDist_: 0] := Module[{r, first = True},
  While[first || ! Abs[Subtract @@ MinMax[Length /@ r]] ≤ maxDist,
   first = False;
   r = NestWhile[randomlyMutateDdistricts[#, g] &,
     ConnectedComponents@NestWhile[
       ng ↦ EdgeAdd[ng, RandomChoice@Complement[IncidenceList[g, RandomChoice@
              MinimalBy[Select[ConnectedComponents[ng], Complement[IncidenceList[
                    g, #], EdgeList[ng]] =!= {} &], Length]], EdgeList[ng]]],
       Graph[VertexList[g], {}], Length[ConnectedComponents[#]] > td &],
     ! Abs[Subtract @@ MinMax[Length /@ #]] ≤ maxDist &, 1, steps]
  ];
  r]
```

Like before, we can try running this many times and seeing the distribution of election results we get:

```
In[472]:= LaunchKernels[11];
seededDistrictingSimulation = ParallelTable[
   Total@Boole[Total[votes[[#]]] < 5 & /@ seededDistricting[g]], 100 000];
CloseKernels[];
```

In[477]:=
```
Histogram[seededDistrictingSimulation, {1},
 "Probability", Frame → {{True, False}, {True, False}},
 FrameLabel → {"# of districts won by red", "probability"},
 PlotRange → {{-0.5, 4}, All}]
```

Out[477]=



In[479]:=
```
KeySort@Counts[seededDdistrictingSimulation]
```

Out[479]=
```
⟨|0 → 67 374, 1 → 32 598, 2 → 28|⟩
```

This gives only slightly different results from our earlier algorithm with MCMC.

Real Data

Using real voting data from OpenElections, we can try running our algorithms on real data. First, we import the data from OpenElections. We will be using North Carolina as an example.

We use precinct level data but then generate county level data for alignment reasons:

In[627]:=
```
rawData = Import[
   FileNameJoin[{rawDataDirectory, "20121106__nc__general__precinct__raw.csv"}]];
```

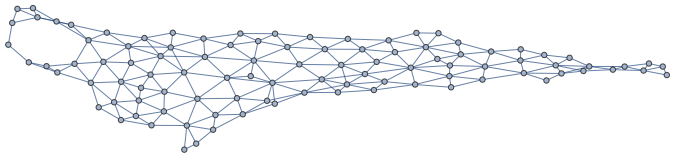
In[628]:=
```
countyMap =
  Association[# → Interpreter["USCounty"][ToLowerCase[#] <> " county NC"] & /@
    Union@rawData[[2 ;;, 16]]];
```

In[630]:=
```
countyVotes = DeleteCases[Query[All, Query[GroupBy[First → Last]] /* Map[Total] /*
      (Boole[Lookup[#, "DEM", 0] > Lookup[#, "REP", 0]] &), {15, 19}]@
    KeyMap[countyMap, GroupBy[Select[Rest[rawData],
       #[[8]] === "US HOUSE OF REPRESENTATIVES" &], #[[16]] &]], {0, 0}];
```

Next, we generate the adjacency graph for counties in North Carolina:

In[634]:=
```
ncg = VertexDelete[Graph[Keys[countyVotes],
    DeleteDuplicatesBy[Catenate@KeyValueMap[Thread@*UndirectedEdge,
       EntityValue[Keys[countyVotes], "BorderingCounties", "EntityAssociation"]],
     Sort]], Except[Alternatives @@ Keys[countyVotes]]]
```

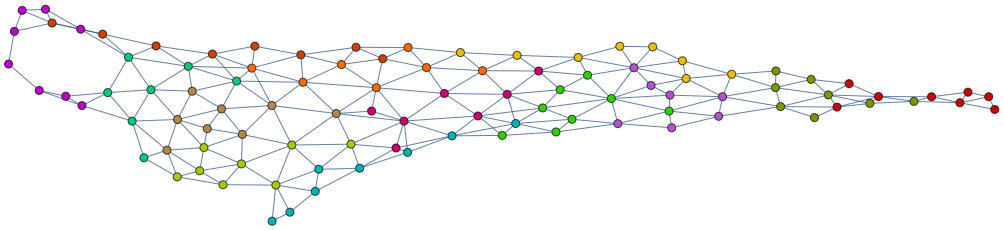
Out[634]=



Finally, we can just use our seeded MCMC districting to generate random districts:

In[636]:=
```
HighlightGraph[ncg, seededDistricting[ncg, 13, 1000, 1]]
```

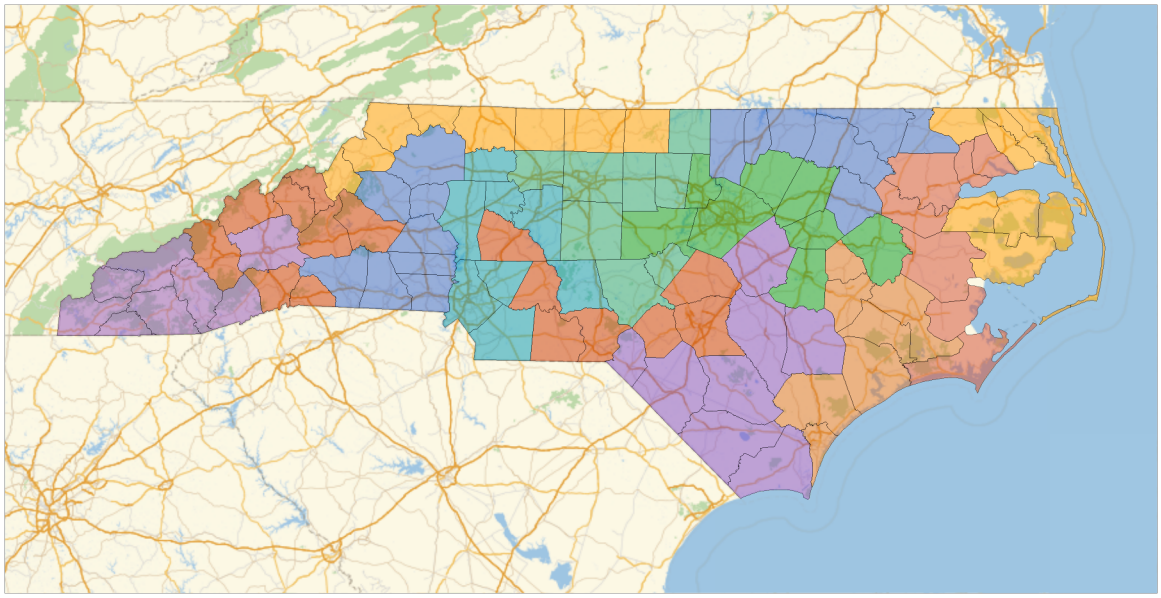
Out[636]=



We can also put them on a map:

In[644]:=
```
GeoListPlot[seededDistricting[ncg, 13, 1000, 1], PlotLegends → None, ImageSize → 600]
```

Out[644]=



Our algorithm is fairly efficient, partitioning NC in about a third of a second on a laptop:

In[646]:=
```
RepeatedTiming[seededDistricting[ncg, 13, 1000, 1];]
```

Out[646]=
```
{0.33, Null}
```

We can then do our same statistical analysis by generating 10 000 random districts and measuring what fraction of them are won by what party:

In[740]:=
```
LaunchKernels[11];
ncDistrinctingSimulation = ParallelTable[Total@Boole[Mean[#] > 0.5 & /@
      Map[countyVotes, seededDistricting[ncg, 13, 1000, 1], {2}]], 10 000];
CloseKernels[];
```

We can see the distribution of election outcomes that we would expect from random districts:

In[743]:=
```
Histogram[13 - ncDistrinctingSimulation, {1},
 "Probability", Frame → {{True, False}, {True, False}},
 FrameLabel → {"# of districts won by red", "probability"},
 PlotRange → {{-0.5, 13}, All}]
```

Out[743]=