Counting Spanning Trees on Triangular Lattices

Angie Wang

Claremont McKenna College

Counting Spanning Trees on Triangular Lattices

Submitted to
Professor Sarah Cannon

Angie Wang
B.A. Senior Thesis
April 24, 2023

Abstract

This thesis focuses on finding spanning tree counts for triangular lattices and other planar graphs comprised of triangular faces. This topic has applications in redistricting: many proposed algorithmic methods for detecting gerrymandering involve spanning trees, and graphs representing states/regions are often triangulated. First, we present and prove Kirchhoff's Matrix Tree Theorem, a well known formula for computing the number of spanning trees of a multigraph. Then, we use combinatorial methods to find spanning tree counts for chains of triangles and $3 \times n$ triangular lattices (some limiting formulas exist, but they rely on higher level mathematics). For a chain of t triangles, we find and prove an unexpected result: the number of spanning trees is equal to the $2(t+1)$th Fibonacci number. For $3 \times n$ triangular lattices, we provide lower and upper bounds for the spanning tree count by decomposing the larger lattice into smaller subgraphs and analyzing those subgraphs.

Acknowledgements

I would like to thank Professor Sarah Cannon for her support and guidance throughout this endeavor. I am grateful for the time she devoted to our weekly meetings, for her patience in answering all of my questions, and for her thoughtful and thorough feedback in every step of the process. It has been a privilege to learn from you.

Contents

1 Introduction **4**

1.1 Motivation . . . 4

1.2 Related Work . . . 6

2 Background **9**

3 Kirchhoff's Matrix Tree Theorem **13**

4 Spanning Trees in Triangular Lattices **18**

4.1 Chains of Triangles . . . 18

4.2 $3 \times n$ Triangular Lattices . . . 22

4.2.1 Lower Bound . . . 22

4.2.2 Upper Bounds . . . 24

4.3 $n \times n$ Triangular Lattices . . . 31

5 Conclusion and Next Steps **35**

1 Introduction

1.1 Motivation

In the United States and in many countries around the world, geographic regions are divided into districts, which in turn elect political representatives. *Redistricting* is the process of drawing boundaries of electoral districts. In the United States, redistricting takes place every 10 years to reflect the decennial census. Redistricting is a complex and challenging process because there are many political, legal, and demographic considerations. Moreover, redistricting may lead to *gerrymandering*, which is when district lines are drawn to favor a particular party, candidate, or ethnic group.

In the redistricting process in the United States, most states require two criteria pertaining to the shape of districts: contiguity and compactness. A district is *contiguous* if it is possible to travel between any two points within the district without crossing into another district. A district is *compact* if, roughly speaking, it is a "reasonable" shape. Although compactness is frequently mentioned in the context of redistricting, there is no precise, agreed-upon definition of the term (see, for example, [18]). The vague criteria for drawing district lines, combined with various other legal requirements, the inherently subjective notion of "fairness," and the overwhelming number of possible districting plans, makes gerrymandering extremely difficult to detect.

The use of quantitative methods in tackling challenges related to redistricting can be traced back to the 1960s. During this time, computers were just becoming prevalent in academic and industrial research, and political scientists were becoming increasingly interested in using computers as a tool for mathematical modeling, simulation, and large-scale survey analysis. Moreover, postwar engineers, scientists, and mathematicians trained in operations research, systems theory, dynamic processing, and game theory were eager to put their training to practical use. To them, redistricting was just another problem that could be tackled using their theoretical tools. This interdisciplinary interest led to the earliest efforts in applying computational techniques to redistricting.

One of the earliest papers in this field was published by Hess and Weaver in 1963. Hess and Weaver believed that computerized redistricting had the potential to create nonpartisan plans: in [25], they proposed a new compactness "score" that not only accounted for the district's shape, but also accounted for the population density in specific areas. Their program generated a series of plans through an iterative process and saved the ones that had the highest compactness scores (based on their newly proposed metric) and lowest population deviations (population deviation refers to the difference in population between districts). In 1965, Stuart Nagel, a political scientist, approached the problem from a different perspective: he believed that computers had the potential to create bipartisan plans, rather than nonpartisan ones. To this end, he built a hill-climbing algorithm that took existing districting plans and "improved" them. More concretely, the program assigned a score to each districting plan and proposed adjustments to the districting plan; if the adjustments increased the score, they would be adopted (see [29]). For more information on the history of the intersection of computing and redistricting, see [33].

Recent advances in statistics and computer science have given rise to a new strategy for identifying gerrymandering: *ensemble-based analysis*. The strategy behind ensemble-based analysis is to compare a proposed plan against a large collection of districting plans; if the proposed plan seems like an outlier, there is reason to suspect that gerrymandering may be occurring ([6]).

The first step of ensemble-based analysis is to create a large collection (referred to as an *ensemble*) of districting plans. Markov chain Monte Carlo (MCMC) methods have emerged as the most prominent computational tool for constructing ensembles. A Markov chain is a process that moves from one state to another in a randomized way. A key property of Markov chains is that they are *memoryless*, that is, the probability of moving from one state to another in the next iteration depends only on the current position. Markov chains are appealing due to their convergence behavior: after a certain number of iterations, they are guaranteed to approach a steady state. In the context of redistricting, one type of Markov chain that has been proposed is the Flip walk. In the Flip walk, each iteration reassigns a single geographic unit from one district to another neighboring district. This approach is relatively simple to implement but requires a large number of iterations ([24], [20], [21]). In 2019, DeFord et al. introduced a new family of Markov chains called Recombination, which has been used in numerous academic papers ([2], [3], [4], [10], [9] [11], [13], [16], [38], [17]), technical reports ([8], [15]), and court cases ([5], [12], [19]).

To understand Recombination and next steps for developing the theory behind these computational techniques, we must transform the redistricting problem into a graph partition problem. Most mathematical treatments of the redistricting problem represent the electoral geography in a given state or region as a *dual graph*: the nodes of the graph correspond to geographical units (census blocks, voting precincts, etc.), and the edges indicate whether any two units are adjacent. If we look at the dual graph of a state or large region, the graph has numerous triangular regions since three geographical units are often adjacent to each other. In this formulation, a districting plan is obtained when the nodes are partitioned into k connected subgraphs (with approximately balanced populations), where k is the number of districts.

A class of graphs called trees is particularly useful for the redistricting problem. Intuitively speaking, a *tree* is a minimally connected graph: removing any edge will disconnect the graph. Moreover, a *spanning tree* is a tree that uses all of the vertices of a graph.

Recombination relies heavily on spanning trees. At each step, the algorithm takes a districting plan, fuses 2 neighboring districts, chooses a spanning tree of the dual graph of the fused district, and removes an edge from the spanning tree, which redivides the larger region into 2 new districts (see Figure 4 of [16] for a schematic). Not only does Recombination appear to be more efficient than Flip, Recombination also has a tendency to produce plans consisting of compact districts ([13] proposes a metric for compactness and Recombination generates compact plans according to this measure). It is known that Recombination converges to a distribution called the *spanning tree distribution*, where the probability of a plan being sampled is proportional to the product of the number of spanning trees in each district (see [11]). The spanning tree distribution blends visual compactness (it favors "plump" over "skinny" districts) and functional compactness (it favors plans that have more connections within the districts relative to connections between the districts); see [18] for more information. Moreover, [30] formalized intuition that districts with long boundaries (i.e., "skinny" districts) have a lower probability of being sampled, providing theoretical underpinnings that compact districting plans have more spanning trees. Additionally, for ensemble-based analysis, previous work shows that sampling from spanning tree distributions creates a collection of districting plans that tend to be more contiguous and compact (see [11]).

Thus, because dual graphs representing states/regions often have many triangular faces and spanning trees are crucial in applying algorithmic methods to redistricting, it is useful to study the spanning tree counts on triangulations (graphs where all regions are triangles). In this thesis, we consider the simplest triangulation: the triangular lattice.

1.2 Related Work

Finding spanning tree counts on lattices has been of long-standing interest in mathematics and physics.

Spanning Tree Bounds. In 1847, as part of his study of electrical circuits, physicist Gustav Kirchhoff gave a formula for the spanning tree count of a given graph in terms of a matrix derived from that graph. More specifically, Kirchhoff's Matrix Tree Theorem says that the number of spanning trees of a graph with n vertices is equal to the determinant of any $(n-1) \times (n-1)$ minor of the Laplacian of the graph ([26]). Since the determinant of a matrix is equal to the product of the eigenvalues of that matrix, Kirchhoff's result says that the number of spanning trees of a graph with n vertices is equal to the product of the $n-1$ nonzero eigenvalues of the Laplacian. This formula is simple and elegant in theory, but in practice, the computation becomes unwieldy when the number of vertices gets to be very large. Thus, mathematicians and physicists have been interested in finding other methods for obtaining the spanning tree count of a given graph.

Spanning tree counts of regular graphs (graphs whose vertices all have the same degree) has been studied in a variety of settings: see [1], [27], [28] (we note here that the lattices we consider in this thesis are not regular because they have a finite number of vertices). There is a spectral characterization involving the adjacency matrix for regular graphs, and in [36], Waller extended this characterization to graphs whose vertices do not all have the same degree. [23] applied the arithmetic mean and geometric mean inequality to Waller's result and found that the number of spanning trees of a graph G must be less than or equal to $n^{-1}(2|E|/(n-1))^{n-1}$.

A very recent paper by Tapp ([34]) gives lower and upper bounds on the number of spanning trees of a planar grid subgraph: if G is a simple grid graph, then $\mathfrak{b}^m \leq \tau(G) \leq 4^m$, where $\tau(G)$ denotes the number of spanning trees of G, $\mathfrak{b} = \exp(4C/\pi) \approx 3.2099$ (C is Catalan's constant), and m is the number of faces of G. Rather than appealing to eigenvalues of matrices, Tapp proves these bounds by using a multiplier function, which studies how much the spanning tree count of a grid graph grows with the addition of each new vertex. The paper's proposed bounds are quite general (they apply to regularly and irregularly shaped grid graphs); in this thesis, we restrict our attention to regularly shaped regions. Also, the idea behind the multiplier function is similar to inducting on the number of vertices, which is the strategy we use to prove the formula for the number of spanning trees of a chain of t triangles.

[32] gives asymptotic bounds on the number of spanning trees. More specifically, the paper proves that the number of spanning trees grows as $\exp vz_{\mathcal{L}}$, where v is the number of vertices and $z_{\mathcal{L}}$ is a finite nonzero constant ($z_{\mathcal{L}}$ is often referred to as the *bulk limit*). For the triangular lattice, [32] finds that $z_{\mathcal{L}}$ is approximately equal to 1.615. This paper approaches the problem of finding spanning tree counts using higher level mathematics and physics, while this thesis approaches the problem using simpler combinatorial methods.

Deriving spanning tree counts using electrical network theory. Another approach for studying spanning trees has been through electrical network theory. For example, [7] assigns electrical resistances to the edges of the graph and studies the flow of current between pairs of vertices; the paper shows that this method can be used to study local behavior of spanning trees. Moreover, [35] uses operations on electrical networks to derive relations between the bulk limits of different types of lattices. In this thesis, we are also using smaller pieces of the graph to understand the graph as a whole: we break larger lattices into smaller subgraphs, and in studying those smaller subgraphs, we find information about the spanning tree count of the larger graph.

This thesis is organized as follows. In Section 2 we define notation and review some facts from graph theory. In Section 3 we state and prove Kirchhoff's Matrix Tree Theorem. In Section 4 we present original results about spanning tree counts for chains of triangles and $3 \times n$ triangular lattices and briefly discuss

$n \times n$ triangular lattices. In Section 5 we summarize our results and discuss next steps.

2 Background

We first define notation and present some general vocabulary related to graphs to set the necessary background for the results presented in Section 4. See [37] and Chapter 10 of [31] for more detailed discussions of these concepts.

Intuitively speaking, a graph is a set of dots (called *vertices*), and some of these dots are connected to other dots (these connections are called *edges*).

More formally, a *simple graph* G is a pair of sets (V, E), where V is the non-empty set of *vertices* and E is the (possibly empty) set of edges, where each edge is an unordered pair of distinct elements of V. If we allow E to be a multiset (i.e., allow repeated edges) then G is called a *multigraph.*

If $G = (V, E)$ and $H = (V', E')$ are both graphs, then H is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$.

Next, we provide some definitions frequently encountered in graph theory, and then we present a few useful facts about trees.

Let $G = (V, E)$ be a graph, and let $x_0, x_1, \ldots, x_m \in V$. Then a sequence of m edges of the form

$$\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{m-1}, x_m\}$$

is called a *walk* of length m. If a walk has distinct vertices, then it is called a *path.* If a walk has distinct vertices except for $x_0 = x_m$, then it is called a *cycle.*

A graph is *connected* if, for each pair of distinct vertices, there is a walk joining them (intuitively, the reader can go from any vertex to another without picking up her pencil). Suppose $G = (V, E)$ is a general graph and $W \subseteq V$. If W is connected and no vertex outside of W has a walk joining it to a vertex in W, then W is called a *connected component* of G.

Suppose we have a connected graph $G = (V, E)$ and e is an edge of G. Then e is a *bridge* if removing it from the graph leaves a disconnected graph. More generally, a bridge is an edge whose removal increases the number of connected components.

Next, we discuss two special classes of graphs: planar graphs and trees. A *planar graph* is a graph that can be drawn in the plane with no edges crossing. A *planar drawing* is one such drawing. A planar drawing of a graph partitions the plane into regions, which are often referred to as *faces.* A *dual graph* of a planar drawing is a planar drawing whose vertices correspond to the faces of G; if two faces share an edge, there is an edge connecting the respective vertices.

As mentioned in the previous section, there is another class of graphs called trees that are particularly useful in the context of the redistricting problem. G is called a *tree* if it has no cycles and is connected. There are two characterizations of trees that we use throughout this thesis (see Section 10.2 of [31] for the proofs of these statements):

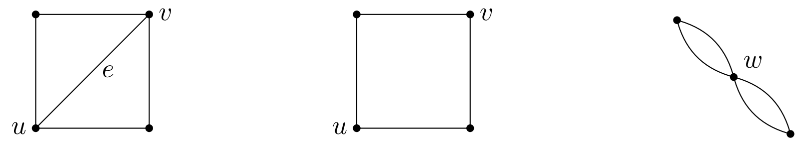
(a) $G = (V, E)$ is a tree if and only if G is connected and every edge is a bridge;

(b) $G = (V, E)$ is a tree if and only if G is connected and $|E| = |V| - 1$.

Every connected graph has subgraphs that are trees. In particular, a *spanning tree* of a connected graph G is a subgraph of G that is a tree and contains every vertex of G. Spanning trees have wide applications, particularly in theoretical computer science and engineering, and as discussed in the previous section, they are relevant for the redistricting problem. A natural first question would be to ask how many spanning trees there are for a given graph.

Kirchhoff's Matrix Tree Theorem offers one method of counting the number of spanning trees for any given multigraph. Before we formally state the theorem and its proof, we introduce a few more ingredients that are needed for the proof: deletion, contraction, the Laplacian, and the combinatorial definition of a determinant.

First, we discuss deletion and contraction, two operations on a graph. Let G be a general graph and $e = \{u, v\}$ be an edge of G (see Figure 1a). We define two new graphs, $G - e$ and $G \cdot e$ that are obtained from G by *deletion* and *contraction*, respectively, of the edge e. The graph $G - e$ has the same vertices as the graph G and all of the edges of G except e (see Figure 1b). The graph $G \cdot e$ is the same as the graph G except the vertices u and v are replaced with a single vertex w, and any vertex that was adjacent to either u or v is now adjacent to w. If a vertex was adjacent to both u and v, then the contraction of e will lead to repeated edges and thus yield a multigraph (see Figure 1c).



(a) G (b) $G - e$ (c) $G \cdot e$

Figure 1: Deletion and contraction of G

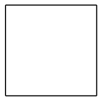
In this thesis, we use $\tau(G)$ to denote the number of spanning trees of G. The following proposition is needed for proving Kirchhoff's Matrix Tree Theorem.

Proposition 2.1. *(Deletion-Contraction). Let G be a simple graph and let e be an edge of G. Then*

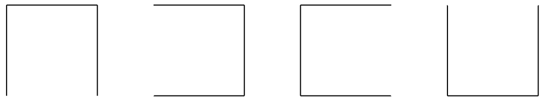
$$\tau(G) = \tau(G \cdot e) + \tau(G - e).$$

Proof. Every spanning tree of G either contains e or it does not. It is clear that $\tau(G - e)$ counts the number of spanning trees that do not use e. We claim that $\tau(G \cdot e)$ counts the number of spanning trees that use e because there exists a bijection between the spanning trees of G that include e and the spanning trees of $G \cdot e$. To prove this claim, take a spanning tree of G that uses e and call this spanning tree T. If we contract e, we will obtain a spanning tree of $G \cdot e$. We can reverse this process. That is, if we are given a spanning tree of $G \cdot e$, we can take the corresponding edges of T and add e to obtain a spanning tree of G that uses e. □

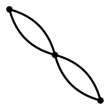
Example 2.2. *Let G be the graph in Figure 1a. Observe that $\tau(G - e) = 4$ (see Figure 2b) and $\tau(G \cdot e) = 4$ (see Figure 2d). Thus $\tau(G) = 8$.*



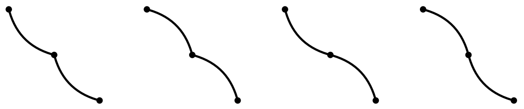
(a) $G - e$



(b) Spanning trees of $G - e$



(c) $G \cdot e$



(d) Spanning trees of $G \cdot e$

Figure 2: Spanning trees of $G - e$ and $G \cdot e$.

Next, we introduce the *Laplacian*, which is a matrix representation of a graph. The entries of $L = [l_{ij}]$ are given by

$$l_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ adj. to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the diagonal entries of L list the degrees of each vertex, and the off-diagonal entries count -1 for every edge joining vertices i and j. The Laplacian can also be written as $L = D - A$, where D is the degree matrix and A is the adjacency matrix of the graph. Observe that since every row of L sums to 0, L is not invertible (and thus $\det(L) = 0$).

Finally, we give the combinatorial definition of the determinant. A *permutation* of the list $1, 2, \ldots n$ is a one-to-one function $\sigma : \{1, 2, \ldots n\} \to \{1, 2, \ldots n\}$

that reorders the elements of the list. A permutation that swaps two elements of the list and leaves all others fixed is called a *transposition.* Each permutation can be written as a composition of transpositions (there are many possible compositions for every permutation, but regardless of which composition is considered, the parity of the number of transpositions will always be the same). We say that σ is *even* if the number of required transpositions is even, and we say that σ is *odd* if the number of required transpositions is odd. The *sign* of σ is

$$\operatorname{sgn}\sigma = \begin{cases} 1 & \text{if } \sigma \text{ is even,} \\ -1 & \text{if } \sigma \text{ is odd.} \end{cases}$$

The determinant of a $n \times n$ matrix M can be written

$$\det A = \sum_{\sigma} \left(\operatorname{sgn}\sigma \prod_{i=1}^{n} a_{i\sigma(i)} \right), \tag{1}$$

where the sum is over all $n!$ permutations of $1, 2, \ldots n$. For more information, see Appendix C.4 of [22].

3 Kirchhoff's Matrix Tree Theorem

In this section, we state and prove Kirchhoff's Matrix Tree Theorem, which gives a formula for the number of spanning trees of a graph.

Theorem 3.1 (Kirchhoff's Matrix Tree Theorem)**.** *Let G be a multigraph with n vertices, let L_G be the Laplacian of G, let $k \in \{1, 2, \ldots, n\}$ and let $L_G(k)$ denote the matrix obtained from L_G by deleting its kth row and kth column. Then*

$$\tau(G) = \det L_G(k).$$

Proof. The overall structure of the proof is as follows: we first present two base cases and then induct on the number of edges and vertices. Two base cases are necessary because $\tau(G)$ depends on both $\tau(G - e)$ and $\tau(G \cdot e)$ (see Proposition 2.1 in Section 2); for any graph, the deletion operation reduces the number of edges by 1 and the contraction operation reduces the number of edges by 1 and the number of vertices by 1. Thus, if we repeatedly delete and contract edges, we will either obtain a graph with 2 vertices and n edges or a graph that is disconnected, which makes it clear that 2 separate base cases are necessary. See Figure 3 for a visual depiction of the induction structure.

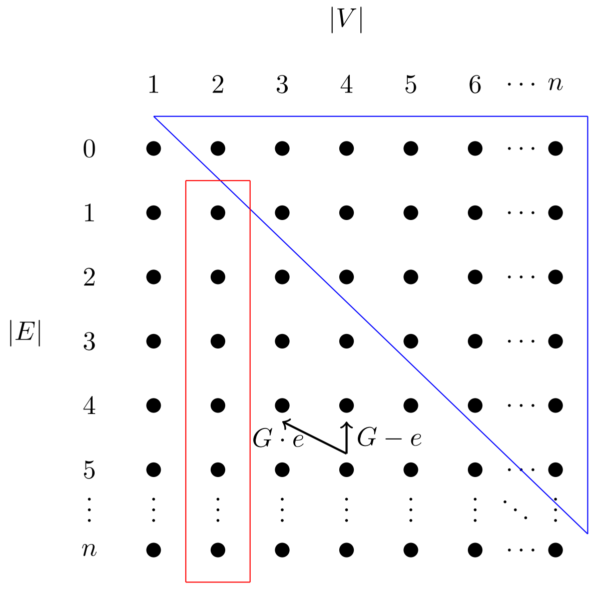


Figure 3: Depiction of the induction structure for the proof of Kirchhoff's Matrix Tree Theorem. The dots inside the blue triangle represent the graphs covered by Base Case 1 (graphs that are disconnected), and the dots inside the red box represent the graphs covered by Base Case 2 (graphs with 2 vertices and n edges).

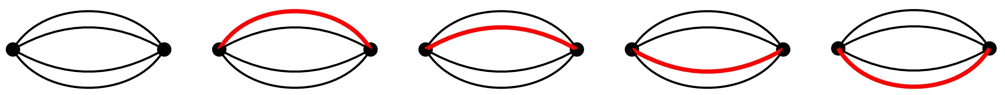


Figure 4: G is a multigraph with 2 vertices and $n = 4$ edges, so G has 4 spanning trees.

Base Case 1. First, suppose $G = (V, E)$ is disconnected. That is, suppose $|V| = n$ and $|E| \le n - 2$. Since spanning trees are trees and trees are connected, we know that $\tau(G) = 0$. Now, we want to show that $\tau(G) = \det L_G(k)$. Without loss of generality, suppose G has 2 connected components $G_1 = (V_1,\ E_1)$ and $G_2 = (V_2,\ E_2)$, where $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$, $E_1 \cup E_2 = E$, and $E_1 \cap E_2 = \emptyset$ (if G has more than 2 connected components, we can induct on the number of components and obtain the same result), and suppose that $k \in V_1$.[1] Let L_1 and L_2 be the Laplacians of G_1 and G_2, respectively. Then the Laplacian of G can be written as $L_G = L_1 \oplus L_2$. Thus,

$$\det L_G(k) = \det L_1(k) \cdot \underbrace{\det L_2}_{=0} = 0,$$

so $\tau(G) = 0 = \det L_G(k)$.

Base Case 2. Suppose G is a multigraph with 2 vertices and an arbitrary number of edges (that is, suppose $|V| = 2$ and $|E| = n \ge 1$). We can obtain a spanning tree by picking any one of the n edges, so $\tau(G) = n$. For an example, see Figure 4. Now, we want to show that $\tau(G) = \det L_G(k)$. The Laplacian of G is

$$L_G = \begin{bmatrix} n & -n \\ -n & n \end{bmatrix}.$$

Thus, $\tau(G) = n = \det L_G(k)$, where $k = 1$ or 2.

Inductive Step. Assume $\tau(G \cdot e) = \det L_{G \cdot e}(k)$ and $\tau(G - e) = \det L_{G-e}(k)$. We want to show that $\tau(G) = \det L_G(k)$. Since $\tau(G) = \tau(G \cdot e) + \tau(G - e)$ (see Proposition 2.1 from Section 2), it is enough to show that

$$\det L_G(u) = \det L_{G \cdot e}(w) + \det L_{G-e}(u),$$

where $e = \{u, v\}$ and w is the new vertex obtained from contracting e.

First, observe that if we compare L_G and L_{G-e} entry-by-entry, they only differ in the $(u,u), (u,v), (v,u)$ and (v,v) entries. In particular, $L_G(u)$ and $L_{G-e}(u)$

[1]If $k \in V_2$, the analysis is similar: $\det L(k) = \det L_1 \cdot \underbrace{\det L_2(k)}_{=0} = 0$, so $\tau(G) = 0$ as well.

differ only in the (v, v) entry by 1 (for a concrete example, compare (4) and (5) in Example 3.2). To understand this intuitively, first recall that the diagonal entries of the Laplacian give the degree of each vertex and the off diagonal entries of the Laplacian count -1 if there is an edge between vertices i and j. Thus, since e is an edge between vertices u and v, removing e will not only reduce the degrees of both u and v by 1 (which explains why the (u, u) and (v, v) entries will differ), but it will also increase the (u, v) and (v, u) entries by 1 (because there is 1 less edge between u and v).

Let E_{vv} be the matrix with a 1 in the (v, v) entry and 0 everywhere else. Then

$$L_G(u) = L_{G-e}(u) + E_{vv}. \tag{2}$$

Without loss of generality let $v = 1$. Let $M = L_{G-e}(u)$. Thus, we can rewrite Equation (2) as $L_G(u) = M + E_{11}$. Computing the determinant of $L_G(u)$ using the combinatorial definition (see (1) in Section 2), we obtain

$$\begin{aligned}
\det(M + E_{11}) &= \sum_{\sigma} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} (m + e_{11})_{i\sigma(i)} \\
&= \sum_{\sigma(1)=1} \operatorname{sgn}(\sigma)(m_{11} + 1) \prod_{i=2}^{n} m_{i\sigma(i)} + \sum_{\sigma(1)\neq 1} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} m_{i\sigma(i)} \\
&= \sum_{\sigma(1)=1} \operatorname{sgn}(\sigma) m_{11} \prod_{i=2}^{n} m_{i\sigma(i)} + \sum_{\sigma(1)=1} \operatorname{sgn}(\sigma) \prod_{i=2}^{n} m_{i\sigma(i)} \\
&+ \sum_{\sigma(1)\neq 1} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} m_{i\sigma(i)} \\
&= \sum_{\sigma} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} m_{i\sigma(i)} + \sum_{\sigma(1)=1} \operatorname{sgn}(\sigma) \prod_{i=2}^{n} m_{i\sigma(i)} \\
&= \det M + \det M(1).
\end{aligned}$$

Resubstituting $M = L_{G-e}(u)$ and $v = 1$ gives

$$\det L_G(u) = \det(L_{G-e}(u) + E_{vv}) = \det L_{G-e}(u) + \det L_{G-e}(u, v). \tag{3}$$

Now, we claim that $L_{G-e}(u, v) = L_{G\cdot e}(w)$. To prove this claim, we first consider $G \cdot e$. After we contract e, the degree of all of the vertices (except w) and the vertices they are adjacent to is unchanged. This is the same as considering $G - e$ and observing that after we delete e, the degrees of all of the vertices of G (except u and v) and the vertices they are adjacent to is also unchanged. Thus, $L_{G-e}(u, v) = L_{G\cdot e}(w)$, so $\det L_{G-e}(u, v) = \det L_{G\cdot e}(w)$. Using this fact, we can rewrite (3) as $\det L_G(u) = \det L_{G-e}(u) + \det L_{G\cdot e}(w)$.

Since we assumed that $\tau(G \cdot e) = \det L_{G\cdot e}(k)$ and $\tau(G - e) = \det L_{G-e}(k)$, we know that $\det L_G(u) = \tau(G \cdot e) + \tau(G - e)$. Then, Proposition 2.1 from

Section 2 allows us to conclude that $\tau(G) = \det L_G(u)$.

$\square$

Example 3.2. *Suppose G is the following graph:*

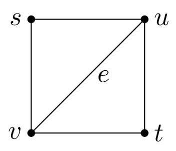


Figure 5: G has 4 vertices: u, v, s, t.

Observe that:

$$L_G = \begin{array}{c c} & \begin{array}{cccc} u & v & s & t \end{array} \\ \begin{array}{c} u \\ v \\ s \\ t \end{array} & \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix} \end{array}, \tag{4}$$

so $\det L_G(u) = \det L_G(v) = \det L_G(s) = \det L_G(t) = 8$. *Thus Kirchhoff's Matrix Tree Theorem tells us that G has 8 spanning trees.*

Moreover, observe that in this example, deletion and contraction of e produce the following 2 graphs:

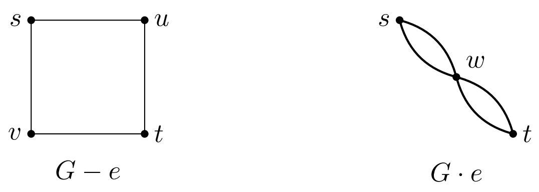


Figure 6: Deletion and contraction of e.

We consider the Laplacians of both graphs in Figure 6:

$$L_{G-e} = \begin{array}{c} \\ u \\ v \\ s \\ t \end{array} \begin{array}{c} \begin{array}{cccc} u & v & s & t \end{array} \\ \begin{bmatrix} 2 & 0 & -1 & -1 \\ 0 & 2 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix} \end{array}, \tag{5}$$

$$L_{G\cdot e} = \begin{array}{c} \\ w \\ s \\ t \end{array} \begin{array}{c} \begin{array}{ccc} w & s & t \end{array} \\ \begin{bmatrix} 4 & -2 & -2 \\ -2 & 2 & 0 \\ -2 & 0 & 2 \end{bmatrix} \end{array}. \tag{6}$$

Thus, if we compare (4) and (5), it is clear that $L_G(u)$ and $L_{G-e}(u)$ differ only in the (v, v) entry by 1. So if E_{11} is the matrix with a 1 in the $(1, 1)$ entry and 0 everywhere else, we have $L_G(u) = L_{G-e}(u) + E_{11}$. This example also makes it clear that $L_{G-e}(u, v) = L_{G\cdot e}(w)$.

4 Spanning Trees in Triangular Lattices

Kirchhoff's Matrix Tree Theorem provides a simple formula for computing the number of spanning trees of a graph, but the computation becomes unwieldy if we are given a graph with many vertices and edges. Thus, it is useful to find more efficient methods (ideally closed forms) of obtaining spanning tree counts. In this section, we narrow the scope of this problem by considering triangular lattices and other planar graphs with triangular faces.

The triangular lattice is constructed by taking an $m \times n$ grid graph (where m and n are the height and length of the lattice, respectively, given by the number of vertices) and drawing diagonal edges through each square unit. In the first subsection, we consider chains of triangles, which include $2 \times n$ triangular lattices and fan graphs and prove that the spanning tree count corresponds to every other number of the Fibonacci sequence. In the second subsection, we consider $3 \times n$ triangular lattices and provide lower and upper bounds for the spanning tree count. In the third subsection, we consider $n \times n$ triangular lattices and see whether they approach the bulk limit proposed in [32].

4.1 Chains of Triangles

First, we consider chains of triangles. Intuitively speaking, these graphs consist of a single "layer" of triangles. More formally, the graphs we consider in this subsection are planar triangulations whose dual graphs (excluding the exterior face) are paths. Let G_t denote a graph with t triangles. Each G_{t+1} is constructed by taking G_t and adding 2 new edges to G_t: l_{t+1} and k_{t+1}. We define l_t as the edge that separates the t and $(t+1)$th triangles and k_t as the edge in triangle t that is not also in another neighboring triangle. For the last triangle in the chain, it does not technically matter which edge we refer to as k and which edge we refer to as l (see Figures 7a, 7b, 7c). For a series of t triangles constructed in this particular way, we ask whether there is a formula for the number of spanning trees g_t, and we find the unexpected result that g_t is equal to every other number in the Fibonacci sequence.

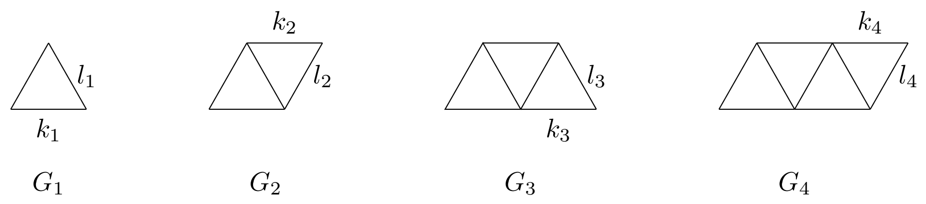
Theorem 4.1. *For a chain of t triangles G_t, g_t is equal to the $2(t+1)$th Fibonacci number.*

Proof. First, observe that the spanning trees of G_t can include (1) either k_t or l_t, or (2) both k_t and l_t.

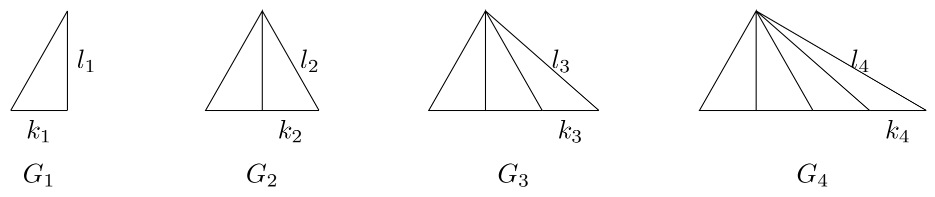
Let b_{t-1} be the number of spanning trees of G_t that include both k_t and l_t.

The number of spanning trees of G_t that include either k_t or l_t is equal to $2g_{t-1}$: we can add either k_t or l_t to each spanning tree of G_{t-1} and obtain 2 new spanning trees of G_t. Thus, we can write g_t as

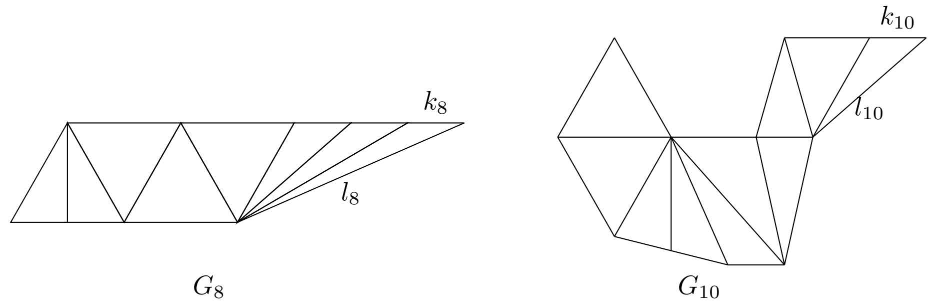
$$g_t = 2g_{t-1} + b_{t-1}. \tag{7}$$



(a) Rows of triangles

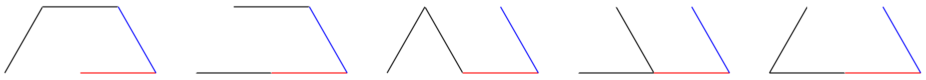


(b) Fan graphs

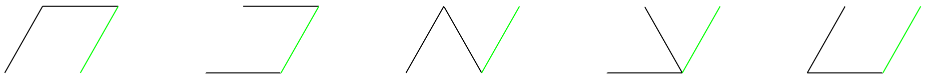


(c) Chains of triangles

Figure 7: Examples of the types of graphs considered in this section.



(a) Spanning trees of G_3 that include k_3 (red) and l_3 (blue)



(b) Spanning trees of G_2 that include l_2 (green)

Figure 8: There is a bijection between the spanning trees of G_3 that include both k_3 and l_3 and the spanning trees of G_2 that include l_2.

Next, we claim that there exists a bijection between the set of spanning trees of G_t that include both k_t and l_t and the set of spanning trees of G_{t-1} that include l_{t-1}.

Proof of Claim. Let T be a spanning tree of G_{t-1} that includes l_{t-1}. Define a function f that removes l_{t-1} from T and adds k_t and l_t. This resulting graph $f(T)$ still has no cycles, contains all of the vertices of G_t, and is connected, so $f(T)$ is a spanning tree. This function is surjective because all spanning trees of G_t with k_t and l_t can be constructed from a spanning tree of G_{t-1} that includes l_{t-1}. The function is injective because it is not possible to obtain a spanning tree of G_t with k_t and l_t from 2 different spanning trees of G_{t-1} that include l_{t-1}. Thus, b_{t-1} is equal to the number of spanning trees of G_{t-1} that include l_{t-1} (see Figure 8 for an example).

The spanning trees of G_{t-1} that include l_{t-1} can include (1) only l_{t-1} or (2) both l_{t-1} and k_{t-1}. For the first case, the number of spanning trees of G_{t-1} that include l_{t-1} is equal to g_{t-2} (for each spanning tree of G_{t-2}, add l_{t-1}). For the second case, the number of spanning trees of G_{t-1} that include both l_{t-1} and k_{t-1} is equal to b_{t-2}. Thus, we can write b_{t-1} as

$$b_{t-1} = g_{t-2} + b_{t-2}. \tag{8}$$

Now, we claim that $b_{t-1} = g_{t-1} - g_{t-2}$. We prove this claim using induction.

Base Case. For $t = 3$, there are 5 spanning trees of G_3 that include both k_3 and l_3 (see Figure 8a). Since $b_{3-1} = b_2 = g_2 - g_1 = 8 - 3 = 5$, we have shown that the theorem holds for $t = 3$.

Inductive Step. We assume for $t \geq 4$,

$$b_{t-2} = g_{t-2} - g_{t-3} \tag{9}$$

is true. We want to show $b_{t-1} = g_{t-1} - g_{t-2}$. Using Equations (7) and (9), we know that

$$g_{t-1} = 2g_{t-2} + (g_{t-2} - g_{t-3}),$$

which implies that

$$2g_{t-2} = g_{t-1} - g_{t-2} + g_{t-3}. \tag{10}$$

Combining Equations (8), 9), and (10) gives us

$$\begin{aligned} b_{t-1} &= g_{t-2} + b_{t-2} \\ &= g_{t-2} + (g_{t-2} - g_{t-3}) \\ &= 2g_{t-2} - g_{t-3} \\ &= (g_{t-1} - g_{t-2} + g_{t-3}) - g_{t-3} \\ &= g_{t-1} - g_{t-2}. \end{aligned}$$

Then, substituting $b_{t-1} = g_{t-1} - g_{t-2}$ into Equation (7) gives

$$\begin{aligned} g_t &= 2g_{t-1} + (g_{t-1} - g_{t-2}) \\ &= 3g_{t-1} - g_{t-2}. \end{aligned} \tag{11}$$

Next, we claim that Equation (11) gives every other number of the Fibonacci sequence.

Recall that the Fibonacci sequence is defined recursively as

$$F_n = F_{n-1} + F_{n-2}. \tag{12}$$

Rearranging Equation (12) gives $F_{n-1} = F_n - F_{n-2}$. Additionally, the recurrence relation tells us that

$$F_{n+1} = F_n + F_{n-1}. \tag{13}$$

If we substitute $F_{n-1} = F_n - F_{n-2}$ into Equation (13), we obtain

$$F_{n+1} = F_n + (F_n - F_{n-2}) = 2F_n - F_{n-2}.$$

Thus for F_{n+2} we have

$$F_{n+2} = F_{n+1} + F_n = (2F_n - F_{n-2}) + F_n = 3F_n - F_{n-2}.$$

$\square$

Recall that the first 11 numbers of the Fibonacci sequence are the following:

F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}
1	1	2	3	5	8	13	21	34	55	89

Thus, g_1 corresponds to $F_4 = 3$, g_2 corresponds to $F_6 = 8$, g_3 corresponds to $F_8 = 21$, $g_4 = 55$ corresponds to F_{10}, which makes it clear that $g_t = F_{2(t+1)}$.

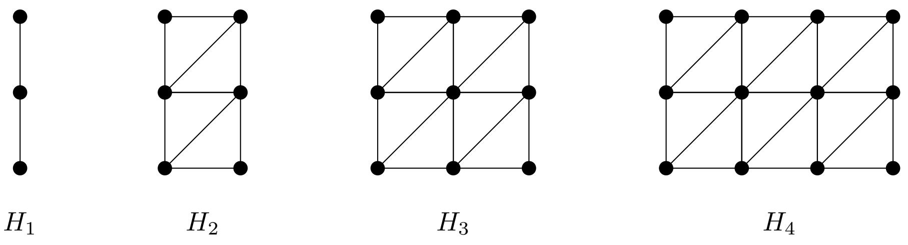


Figure 9: Triangular lattices of height 2 for small n.

4.2 $3 \times n$ Triangular Lattices

Next, we consider $3 \times n$ triangular lattices and provide lower and upper bounds for the spanning tree count. Let H_n denote the lattice of height 3 and length n. Observe that H_n has $3n$ vertices and $7n - 5$ edges. Let h_n denote the number of spanning trees of H_n. See Figure 9 for examples of graphs considered in this section.

We can use Kirchhoff's Matrix Tree Theorem to find spanning tree counts for small n: [2]

n	h_n
1	1
2	55
3	2,080
4	76,987
5	2,844,577
6	105,089,725
7	3,882,384,640

Since h_n is not a known sequence, we hope to find constants α and β (and polynomial functions $f(n)$ and $g(n)$) such that

$$f(n)\ \alpha^n \leq h_n \leq g(n)\ \beta^n.$$

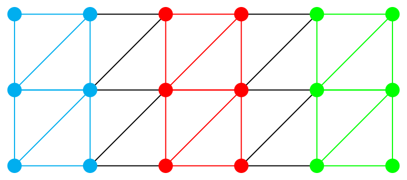
4.2.1 Lower Bound

Before stating our proposed lower bound α, we use an example to explain our strategy for finding an undercount.

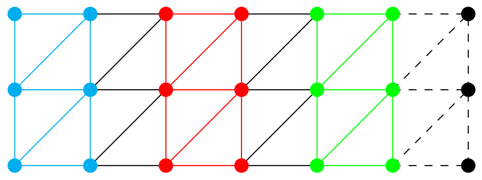
Example 4.2. *Consider H_6. First, suppose we divide the vertices of H_6 into 3×2 subgraphs (see Figure 10a).*

If we pick a spanning tree from each of these subgraphs, we will have a graph

[2]The code for this computation and all other computations are available at this link: `https://github.com/angiewang23/Senior-Thesis-Code`.



(a) H_6. The blue vertices and edges denote Subgraph 1, the red vertices and edges denote Subgraph 2, and the green vertices and edges denote Subgraph 3. To obtain a spanning tree of H_6, we can select a spanning tree from each subgraph and select an edge to connect Subgraphs 1 and 2 and another edge to connect Subgraphs 2 and 3.



(b) H_7. The blue vertices and edges denote Subgraph 1, the red vertices and edges denote Subgraph 2, and the green vertices and edges denote Subgraph 3. We can select a spanning tree from each subgraph, and we can select an edge to connect Subgraphs 1 and 2 and an edge to connect Subgraphs 2 and 3. However, since n is odd, we need to select 3 of the dashed edges to ensure that our graph includes the 3 rightmost vertices (otherwise our subgraph will not be a spanning tree).

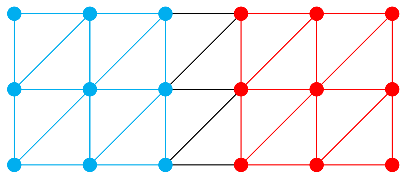


Figure 11: H_6. The blue vertices and edges denote Subgraph 1, and the red vertices and edges denote Subgraph 2.

with 3 connected components. We can connect these components by selecting an edge between Subgraphs 1 and 2, as well as between Subgraphs 2 and 3. Kirchhoff's Matrix Tree Theorem tells us that each subgraph has 55 spanning trees, and there are 5 edges that can connect the subgraphs. Thus, a lower bound for h_6 *is*

$$55^3\ 5^2 = 4,159,375.$$

We can also consider "longer" subgraphs and apply a similar strategy. Instead of dividing the vertices into 3 parts, we can divide the vertices into 3×3 *subgraphs (see Figure 11). From previous computations, we know that each subgraph has* 2080 *spanning trees and there are* 5 *edges that can connect the subgraphs. Thus, another lower bound for* h_6 *is*

$$2080^2 \cdot 5 = 21,632,000.$$

More generally, if n is an even integer, a lower bound for h_n is

$$\frac{1}{5}\left((55 \cdot 5)^{1/2}\right)^n.$$

If n is an odd integer, we have some "leftover" edges, and there is at least one way to choose three of the dashed edges (see Figure 10b) to ensure that all of the vertices are included, so a lower bound for h_n is

$$\frac{1}{5 \cdot (55 \cdot 5)^{1/2}}\left((55 \cdot 5)^{1/2}\right)^n.$$

Observe that regardless of whether n is an even or odd integer, we find that $\alpha = (55 \cdot 5)^{1/2} \approx 16.6$. Thus, when we break the lattice into 3×3 subgraphs, we can analyze when n is a multiple of 3 or not, but this analysis will not change α (the polynomial $f(n)$ will absorb the "off-by-one" error). Thus, we consider the simplest case, that is, when n is a multiple of 3. A tighter lower bound would therefore be

$$\frac{1}{5}\left((2080 \cdot 5)^{1/3}\right)^n,$$

so $\alpha = (2080 \cdot 5)^{1/3} \approx 21.8$.

More generally, if we decompose our lattice into subgraphs C_k (where C_k is the $3 \times k$ triangular lattice), we have

$$\frac{1}{5}\left(\tau(C_k) \cdot 5)^{1/k}\right)^n.$$

As k increases, it is clear that we will obtain a better lower bound. We computed α_k for small k and obtained the following results:

k	2	3	4	5	6	7	8	9	10	11	12
α_k	16.6	21.8	24.9	27.0	28.4	29.5	30.3	31.0	31.6	32.0	32.4

As k becomes large, α increases at an slower rate, and the computation becomes less fruitful. The best lower bound we calculated using this method is 32.4, but we expect small improvements to be made with additional computation time.

4.2.2 Upper Bounds

Next, we present 4 upper bounds. Recall that for the upper bound, we want to find some constant β such that $h_n \leq g(n)\ \beta^n$, where $g(n)$ is a polynomial function. Each proposed bound has its strengths and weaknesses: two of the bounds use relatively simple counting methods (but are less precise), and two of the bounds use more elaborate counting methods (but are more precise).

Upper Bound 1.

We can obtain a particularly simple overcount by using the fact that H_n has $3n$ vertices, so each tree must have $3n-1$ edges. Since each H_n has a total of $7n-5$ edges, we have

$$h_n \leq \binom{7n-5}{3n-1}. \tag{1}$$

We want to rewrite (1) in the form $g(n)\ \beta^n$. To do so, we first claim that $\binom{7n-5}{3n-1} < \binom{7n}{3n}$. To show that this is true, observe that

$$\binom{7n-5}{3n-1} = \frac{(7n-5)!}{(3n-1)!\ (4n-4)!} = \frac{(3n)(4n)(4n-1)(4n-2)(4n-3)}{(7n)(7n-1)(7n-2)(7n-3)(7n-4)} \cdot \binom{7n}{3n}.$$

Now, observe that for $n \geq 2$, $\frac{3n}{7n}, \frac{4n}{7n-1}, \frac{4n-1}{7n-2}, \frac{4n-2}{7n-3}, \frac{4n-3}{7n-4}$ are all less than 1. Thus, we know that the fraction $\frac{(3n)(4n)(4n-1)(4n-2)(4n-3)}{(7n)(7n-1)(7n-2)(7n-3)(7n-4)}$ is less than 1, so $\binom{7n-5}{3n-1} < \binom{7n}{3n}$. To obtain a bound in the form $g(n)\ \beta^n$, we use the well known fact that $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$ (for more information, see Appendix C of [14]):

$$h_n \leq \binom{7n-5}{3n-1} < \binom{7n}{3n} \leq \left(\frac{7n \cdot e}{3n}\right)^{3n} = \left(\left(\frac{7e}{3}\right)^3\right)^n < (255.2)^n.$$

<u>Upper Bound 2.</u>

Another method for obtaining an upper bound is, intuitively speaking, "standing" at every vertex and picking an edge. We claim that the set of subgraphs generated using this method includes all spanning trees. To show that this claim is true, suppose we have a graph G and label the n vertices $v_1, v_2, \dots v_n$ and label the m edges $e_1, e_2 \dots, e_m$. If we take a spanning tree of G, we can write it as a set of n pairs of the form (v_i, e_j). That is, every spanning tree of G can be constructed by standing at a vertex v_i and picking (a particular) edge e_j. Since H_n has 2 vertices with degree 2, 2 vertices with degree 3, $2n-2$ vertices with degree 4, and $n-2$ vertices with degree 6, we have the following bound:

$$h_n \leq 2^2 \cdot 3^2 \cdot 4^{2n-2} \cdot 6^{n-2} = \left(\frac{1}{4}\right)^2 (96)^n,$$

which is a notable improvement from $\beta = 255.2$.

Now, we consider 2 relatively more elaborate methods of finding an overcount. Both of these methods consider H_n as a union of subgraphs. First, we consider H_n as a union of subgraphs denoted S_i (see Figure 12), and then we consider H_n as a union of "longer" subgraphs denoted L_i (see Figure 14). We begin by explaining our strategy through an example.

Example 4.3. *Consider H_5 and one of its spanning trees (see Figure 13). Observe that H_5 is the union of 4 subgraphs: S_1, S_2, S_3, S_4. Additionally, observe that the spanning tree uses 5 edges from S_1 and S_4, 1 edge from S_2, and 3 edges from S_3.*

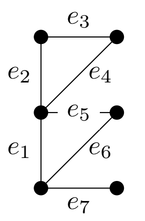


Figure 12: Subgraph S_i has 6 vertices and 7 edges.

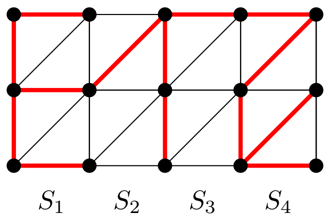


Figure 13: H_5 and one spanning tree.

Upper Bound 3.

Example 4.3 shows that a spanning tree of H_n must use $1, 2, 3, 4$ or 5 edges from each S_i (since S_i has 6 vertices, a spanning tree using 6 or more edges from S_i would lead to a cycle). We analyze each case and provide an upper bound for the number of ways of using that number of edges from a given S_i.

1 edge case. Suppose the spanning tree uses 1 edge from S_i. If e_1 or e_2 are used (see Figure 12), the resulting graph will be disconnected because there is no path connecting S_i to S_{i+1}. Thus, there are $7 - 2 = 5$ possible ways for a spanning tree to use 1 edge from S_i.

2 edge case. Suppose the spanning tree uses 2 edges from S_i. There are $\binom{7}{2}$ possible ways to select 2 edges from S_i. However, spanning trees cannot simultaneously use e_1 and e_2 because the resulting graph would be disconnected, since there would be no path connecting S_i to S_{i+1}. Thus, there are at most $\binom{7}{2} - 1 = 20$ ways for a spanning tree to use 2 edges from S_i.

3 edge case. Suppose the spanning tree uses 3 edges from S_i. There are $\binom{7}{3}$ possible ways to select 3 edges from S_i. However, trees cannot have cycles and there are 2 ways to include a cycle of length 3 (if we use the labeling conventions from Figure 12, we can include a cycle of length 3 by picking e_2, e_3, e_4 or by picking e_1, e_5, e_6) so there are at most $\binom{7}{3} - 2 = 33$ ways for a spanning tree to use 3 edges from S_i.

4 edge case. Suppose the spanning tree uses 4 edges from S_i. There are $\binom{7}{4}$ possible ways to select 4 edges from S_i, but some of these configurations may include a cycle of length 3. There are 2 ways to select a cycle of length 3 and $7-3=4$ ways to select an additional edge, so there are at most $\binom{7}{4}-2\times 4=27$ ways for a spanning tree to use 4 edges from S_i.

5 edge case. Suppose the spanning tree uses 5 edges from S_i. Since S_i has 6 vertices, the number of ways to select 5 edges with no cycles is equivalent to finding the spanning tree count of S_i. Kirchhoff's Matrix Tree Theorem tells us that the spanning tree count of S_i is 9.

Thus, another upper bound for h_n is

$$\left(5+\binom{7}{2}-1+\binom{7}{3}-2+\binom{7}{4}-8+9\right)^n=94^n,$$

which is an improvement over Upper Bound 2.

<u>Upper Bound 4.</u>

We can make this upper bound more precise by using a similar strategy but considering H_n as a union of "longer" subgraphs L_i (see Figure 14). Every spanning tree must use between 4 and 8 edges of L_i, inclusive. We claim that 4 is the minimum number of edges that H_n must use in order to be a connected graph: we need a path between vertices l_i and r_j (where $i,j=1,2$ or 3), which requires at least 2 edges, and we also need m_1, m_2, and m_3 to be connected, which requires 2 more edges (intuitively speaking, we need at least 4 edges to "get from the left to the right side" of L_i and ensure that m_1, m_2, and m_3 are not isolated). Moreover, since L_i has 9 vertices, if we use more than $9-1=8$ edges of L_i, we are guaranteed to have a cycle.

Similar to our computations for Upper Bound 3, we will count the number of ways to include $4,5,6,7$ and 8 edges from L_i and tighten these upper bounds by accounting for "forbidden" cases (i.e., the number of ways to include cycles). However, because L_i has more edges and vertices than S_i, it is not only possible to obtain cycles of length 3 (like we did when using S_i) but also cycles of lengths $4,5,6$, and 7. Before we present the upper bounds for each case, we use an example to show that counting cycles of lengths greater than 3 requires more careful analysis.

Example 4.4. *Suppose we are counting the number of ways to select* 5 *edges of* L_i. *One "forbidden" possibility is selecting a cycle of length* 4 *and one additional edge (see Figure 15). If the additional edge selected is the blue edge, then the five edges together can be categorized as a cycle of length* 4 *(with 1 additional edge), or they can be categorized as 1 cycle of length* 3 *(with* 2 *additional edges). To avoid overcounting "forbidden" cases, when we count the number of ways to include cycles of length* 4 *and an additional edge, we do not permit the edges*

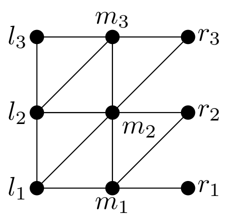


Figure 14: Subgraph L_i has 9 vertices and 14 edges.

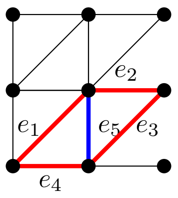


Figure 15: This selection of 5 edges can be considered a cycle of length 4 and 1 additional edge, or it can be considered a cycle of length 3 with 2 additional edges. We consider this selection of edges to be 1 cycle of length 3 with 2 additional edges.

"inside" the cycle to be selected as the additional edge.

More generally, if we have a cycle of length c *and we are not permitted to select remaining edges from "inside" the cycle, there are* $14 - c - (c - 3)$ *edges to choose from for the additional edges.*

We categorize this selection of 5 *edges in Figure 15 as a cycle of length* 3 *with* 2 *additional edges. Moreover, there is one more issue we need to address: observe that this selection is counted twice (depending on whether* e_1, e_4, e_5 *or* e_2, e_3, e_5 *is considered the cycle of length* 3*), so we must also account for double counting cases like this one.*

Now, we analyze each case and provide an upper bound for the number of possible ways of using that number of edges from a given L_i.

4 edge case. Suppose the spanning tree uses 4 edges from L_i. There are $\binom{14}{4}$ ways to select 4 edges from L_i, but some of these selections include cycles of lengths 3 and 4:

- There are 5 ways to include a cycle of length 4.
- There are 6 ways to obtain a cycle of length 3 and $14 - 3 = 11$ edges to choose from for the 1 additional edge.

Thus, an upper bound for the number of ways for a spanning tree to use 4 edges from L_i is

$$\binom{14}{4} - 5 - 6 \cdot 11 = 931. \tag{2}$$

5 edge case. Suppose the spanning tree uses 5 edges from L_i. There are $\binom{14}{5}$ ways to select 5 edges from L_i, but some of these selections include cycles of lengths $3, 4$, and 5:

- There are 5 ways to select a cycle of length 5.
- There are 5 ways to select a cycle of length 4 and $14 - 4 - 1 = 9$ edges to choose from for the 1 additional edge.
- There are 6 ways to select a cycle of length 3 and $14 - 3 = 11$ edges to choose from for the 2 additional edges. However, observe that it is possible for the 5 edges to form 2 cycles of length 3 (see Figure 15); these selections are counted twice, depending on which edges are considered to be a part of the cycle of length 3 and which edges are considered to be the additional edges. Moreover, observe that the number of selections that are double-counted is equivalent to the number of ways to select a cycle of length 4 (because if we select a cycle of length 4 and the edge in the "middle" of the cycle, we will have 2 cycles of length 3), which we know is 5.

Thus, an upper bound for the number of ways for a spanning tree to use 5 edges from L_i is

$$\binom{14}{5} - 5 - 5\binom{9}{1} - 6\binom{11}{2} + 5 = 1627. \tag{3}$$

6 edge case. Suppose the spanning tree uses 6 edges from L_i. There are $\binom{14}{6}$ ways to select 6 edges from L_i, but some of these selections include cycles of lengths $3, 4, 5$, and 6.

- There are 4 ways to select a cycle of length 6.
- There are 5 ways to select a cycle of length 5 and $14 - 5 - 2 = 7$ edges to choose from for the 1 additional edge.
- We disregard the case involving the number of ways to select a cycle of length 4 with 2 additional edges because there is significant overlap with the case involving cycles of length 3. This not only creates challenges with overcounting, but it also does not significantly improve our bound.
- There are 6 ways to select a cycle of length 3 and $14 - 3 = 11$ edges to choose from for the 3 additional edges. However, observe that it is possible for the 6 edges to form 2 cycles of length 3, and these selections are counted twice. Observe that the number of selections that are double-counted can be categorized into 2 subcases:

1. Suppose that 5 of the 6 edges form 2 cycles of length 3 (i.e., the 2 cycles of length 3 share an edge; see Figure 15), and the 6th edge is selected from the remaining $14 - 5 = 9$ edges. There are $5 \cdot 9 = 36$ ways for this case to occur.
2. Suppose the 2 cycles of length 3 do not share an edge. Observe that L_i has 6 cycles of length 3, but 5 pairs of cycles share an edge. Thus, there are $\binom{6}{2} - 5$ ways for this case to occur.

Thus, an upper bound for the number of ways for a spanning tree to use 6 edges from L_i is

$$\binom{14}{6} - 4 - 5\binom{7}{1} - 6\binom{11}{3} + 5\binom{9}{1} + \left(\binom{6}{2} - 5\right) = 2029. \tag{4}$$

7 edge case. Suppose the spanning tree uses 7 edges from L_i. For this case, we use a different approach to obtain an overcount. First, observe that since L_i has 9 vertices, we know that a spanning tree of L_i must have 8 edges. Moreover, recall that since every edge of a tree is a bridge, removing an edge will increase the number of connected components by 1. Thus, if we take a spanning tree of L_i and remove an edge, we will obtain a 7 edge configuration that has 2 connected components. Since the set of spanning trees of L_i captures all 8 edge configurations that are connected and have no cycles, we know that every 7 edge configuration with no cycles can be constructed by deleting an edge from a spanning tree of L_i.

Kirchhoff's Matrix Tree Theorem tells us that there are 369 spanning trees of L_i. Since each spanning tree has 8 possible edges we can remove, there are $8 \cdot 369$ ways to remove an edge and obtain a 7 edge configuration that has 2 connected components and no cycles. The number of times a 7 edge configuration is counted is equal to the number of edges that has one vertex in each component (because the number of edges that has a vertex in each component is the number of ways we can remove an edge from a spanning tree of L_i and obtain that 7 edge configuration). We can observe which parts of $8 \cdot 369$ are an exact count and which are an overcount and correct the overcount appropriately:

(1) Suppose there is only 1 edge that has a vertex in each component (that is, one of the components consists of an isolated vertex; in Figure 16a, v is the isolated vertex and e is the edge that has a vertex in each component). There is a bijection between the set of spanning trees of L_i and the set of 7 edges such that v is an isolated vertex (just remove e from each spanning tree). Thus, there are 369 ways to take a spanning tree of L_i and remove an edge (e) to obtain a set of 7 edges, and this is an exact count.

(2) Suppose there are at least 2 edges connecting the 2 components. In this case, there are at least 2 ways to remove an edge from a spanning tree to obtain that specific 7 edge configuration. Since there are a total of $8 \cdot 369$ ways to remove an edge from a spanning tree and obtain a 7 edge

configuration with no cycles and 369 of these are an exact count, we know that the remaining $8 \cdot 369 - 369$ 7 edge configurations are overcounted by at least a factor of 2. Thus, we must divide $8 \cdot 369 - 369$ by 2. For an example, see Figure 16b.

Thus, an upper bound for the number of ways for a spanning tree to use 7 edges from L_i is

$$\frac{8 \cdot 369 - 369}{2} + 369 = 1660.5. \tag{5}$$

8 edge case. Finally, suppose the spanning tree uses 8 edges from L_i. As we discussed in the 7 edge case, the number of ways to select 8 edges with no cycles is exactly the number of spanning trees of L_i, which we know from Kirchhoff's Matrix Tree Theorem is 369.

Thus, combining Equations (2), (3), (4), (5) and the fact that there are 369 spanning trees of L_i gives us an improved upper bound for h_n:

$$(931 + 1627 + 2029 + 1660.6 + 369)^{n/2} = \left((6616.6)^{1/2}\right)^n \approx (81.3)^n.$$

In summary, we propose 4 upper bounds:

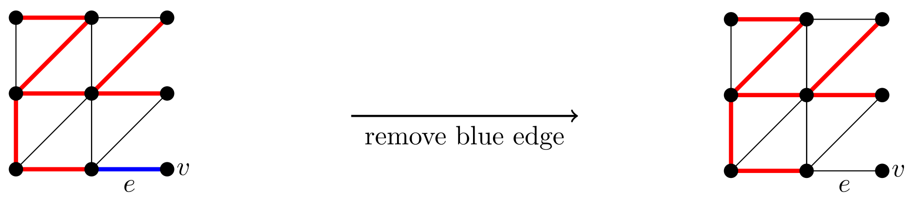
Upper Bound	β
1	255.2
2	96.0
3	94.0
4	81.3

On one hand, the combinatorial strategies behind Upper Bounds 1 and 2 are simpler but result in higher βs (i.e., less precise upper bounds). On the other hand, the combinatorial strategies behind Upper Bounds 3 and 4 are more elaborate but result in lower βs (i.e., more precise upper bounds). Thus, for a $3 \times n$ triangular lattice, the best lower bound we computed was $\alpha = 32.4$ and the best upper bound we computed was $\beta = 81.3$.

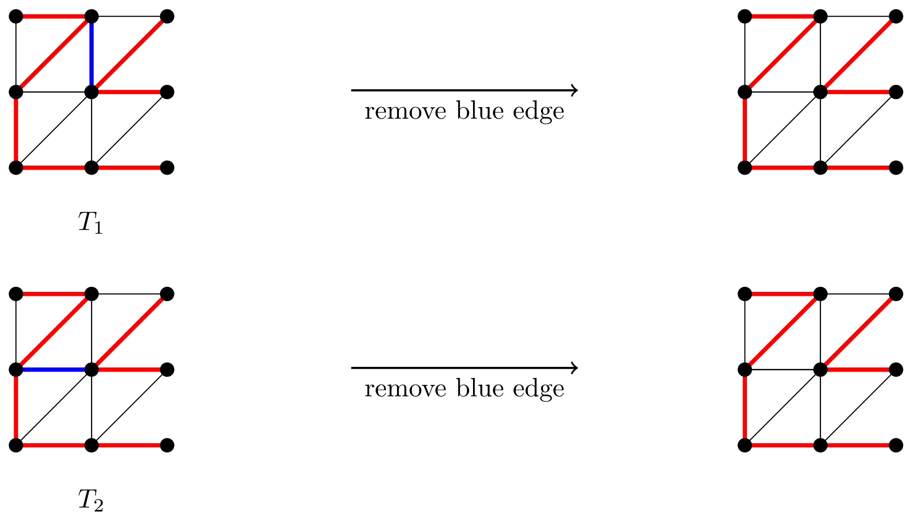
4.3 $n \times n$ Triangular Lattices

Dual graphs of electoral geographies tend to resemble "plumper" shapes, rather than the skinny strips we considered in the previous 2 subsections. Thus, it is useful to consider $n \times n$ triangular lattices, which we denote D_n. We use Kirchhoff's Matrix Tree Theorem to compute $\tau(D_n)$ for small n:

n	$\tau(D_n)$
2	8
3	$2,080$
4	$14,899,040$
5	$2,822,177,161,216$



(a) This is an example of Case 1. If we take a spanning tree of L_i and remove e, we will obtain a 7 edge configuration where v is in its own component. There is a bijection between the spanning trees of L_i and the set of 7 edge configurations where v is in its own component.



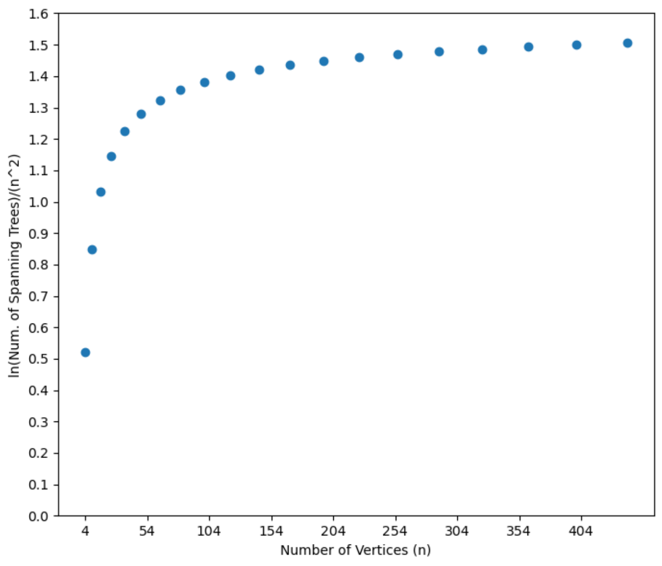
(b) This is an example of Case 2. If we take 2 different spanning trees of L_i and remove an edge from each, it is possible to obtain the same 7 edge configuration. Thus, these cases are all overcounted by at least a factor of 2. In this particular example, observe that there are 6 bridges incident to each connected component, so this 7 edge configuration is overcounted by a factor of 6.

Figure 16: If a spanning tree of H_n uses 7 edges from L_i, we can split our counting into 2 cases and analyze each separately.

It is evident that as n increases, the number of spanning trees grows extremely quickly. These overwhelmingly large numbers are difficult to interpret, so we use the result from [32]: $\tau(D_n) \sim \exp(vz_{\mathcal{L}})$, where v represents the number of vertices of D_n. In the case of the triangular lattice, the paper computes $z_{\mathcal{L}} \approx 1.615$. In the case of $n \times n$ triangular lattices, we know that $v = n^2$, so if we take the natural log of both sides and isolate $z_{\mathcal{L}}$, we have

$$z_{\mathcal{L}} \approx \frac{\ln \tau(D_n)}{n^2}.$$

We compute $z_{\mathcal{L}}$ up until $n = 21$ and find that these numbers are consistent with the limiting behavior given by Shrock and Wu (we were not able to do calculations for large enough n to be sure because the the values of $\tau(D_n)$ become too large for Python to handle). See Figure 17.



(a) This plot shows values of $z_{\mathcal{L}}$ for D_n, where $n = 2, 3, \ldots, 21$.

n	$z_{\mathcal{L}}$
2	0.520
3	0.849
4	1.032
5	1.147
6	1.224
7	1.280
8	1.322
9	1.355
10	1.381
11	1.403
12	1.421
13	1.436
14	1.449
15	1.460
16	1.470
17	1.479
18	1.487
19	1.494
20	1.500
21	1.505

(b) This table shows values of $z_{\mathcal{L}}$ for D_n, where $n = 2, 3, \ldots, 21$.

Figure 17: Computing $z_{\mathcal{L}}$ for $n \times n$ triangular lattices up until $n = 21$.

5 Conclusion and Next Steps

This thesis studies spanning tree counts on triangular lattices. First, we give an inductive proof of Kirchhoff's Matrix Tree Theorem, a well known formula for computing the spanning tree count of a graph. Typical proofs of this theorem use linear algebra; in contrast, we give a 2-dimensional inductive proof that mainly relies on fundamental graph operations. Additionally, we explore the number of spanning trees for chains of triangles and $3 \times n$ triangular lattices. We find and prove the unexpected result that the spanning tree count for a chain of t triangles is equal to the $2(t+1)$th Fibonacci number. Moreover, we give lower and upper bounds for $3 \times n$ triangular lattices by breaking the triangular lattice into smaller pieces and considering the spanning tree counts of those smaller subgraphs.

Now that we have these results, there are several next steps to consider. Developing a more rigorous understanding of the spanning tree count of triangular lattices is necessary in advancing efforts to apply computational techniques to redistricting, since spanning tree counts are crucial for algorithms like Recombination and have been shown to provide a quantitative measure of compactness.

First, we believe there are opportunities to tighten the lower and upper bounds for the spanning tree count of $3 \times n$ triangular lattices. We believe that our approach does not have much potential for major improvements; using a similar inclusion/exclusion approach for larger subgraphs (3×4 subgraphs, 3×5 subgraphs, etc.) will be challenging due to the difficulty in keeping track of double counting. However, better bounds may be obtained by considering other "clever" counting strategies to find undercounts and overcounts. Moreover, in this thesis, we considered three special cases of regularly shaped triangular lattices: chains of triangles, $3 \times n$ triangular lattices, and $n \times n$ triangular lattices, and a next step would be further exploring the spanning tree count of $n \times n$ triangular lattices and considering more general versions of the triangular lattice ($m \times n$ and irregularly shaped triangular lattices). Additionally, this thesis uses simple combinatorial techniques to deduce results about spanning tree counts. Previous results counting spanning trees on lattices rely on advanced techniques such as electrical network theory and statistical mechanics, but we were able to obtain promising results using basic combinatorics. Since much of the literature about finding spanning tree counts uses higher level mathematics, another step would be to reprove existing results using simpler, more elegant methods.

A major next step would be to find a closed, non-asymptotic formula for the spanning tree count that applies to the general triangular lattice, which would significantly enhance the efficiency and precision of computing the number of spanning trees for a given graph. Moreover, it could have direct implications in the computational methods that are at the forefront for detecting gerrymandering and tackling challenges in redistricting.

All code for this thesis is available at this link: `https://github.com/angiewang23/Senior-Thesis-Code`.

References

[1] Noga Alon. The number of spanning trees in regular graphs. *Random Structures & Algorithms*, 1(2):175–181, 1990. doi: https://doi.org/10.1002/rsa.3240010204. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.3240010204`.

[2] Eric Autry, Daniel Carter, Gregory Herschlag, Zach Hunter, and Jonathan C. Mattingly. Metropolized forest recombination for Monte Carlo sampling of graph partitions, 2021.

[3] Eric A. Autry, Daniel Carter, Gregory J. Herschlag, Zach Hunter, and Jonathan C. Mattingly. Metropolized multiscale forest recombination for redistricting. *Multiscale Modeling & Simulation*, 19(4):1885–1914, 2021. doi: 10.1137/21M1406854. URL `https://doi.org/10.1137/21M1406854`.

[4] Amariah Becker, Moon Duchin, Dara Gold, and Sam Hirsch. Computational redistricting and the Voting Rights Act. *Election Law Journal: Rules, Politics, and Policy*, 20, 10 2021. doi: 10.1089/elj.2020.0704.

[5] Amariah Becker, Daryl R. DeFord, Dara Gold, Sam Hirsch, Mary E. Marshall, and Jessica Ring Amunson. Brief of computational redistricting experts as amici curiae in support of Appellees and Respondents. Merrill v. Milligan; Merrill v. Caster; United States Supreme Court, 2022. nos. 21-1086, 21-1087., 2022. URL `https://www.supremecourt.gov/DocketPDF/21/21-1086/230272/20220718153650363_21-1086%2021-1087%20bsac%20Computational%20Redistricting%20Experts.pdf`.

[6] Mira Bernstein and Olivia Walch. Measuring partisan fairness. *Political Geometry*, pages 39–75, 2022.

[7] Robert Burton and Robin Pemantle. Local characteristics, entropy and limit theorems for spanning trees and domino tilings via transfer-impedances. *The Annals of Probability*, 21(3):1329 – 1371, 1993. doi: 10.1214/aop/1176989121. URL `https://doi.org/10.1214/aop/1176989121`.

[8] Sophia Caldera, Daryl DeFord, Moon Duchin, Samuel C. Gutekunst, and Cara Nix. Mathematics of nested districts: The case of Alaska. *Statistics and Public Policy*, 7(1):39–51, 2020. URL `https://EconPapers.repec.org/RePEc:taf:usppxx:v:7:y:2020:i:1:p:39-51`.

[9] Sarah Cannon, Ari Goldbloom-Helzner, Varun Gupta, JN Matthews, and Bhushan Suwal. Voting rights, Markov chains, and optimization by short bursts. *Methodology and Computing in Applied Probability*, 25(1), 2020. URL `https://par.nsf.gov/biblio/10399864`.

[10] Sarah Cannon, Moon Duchin, Dana Randall, and Parker Rule. Spanning tree methods for sampling graph partitions, 2022.

[11] Moses Charikar, Paul Liu, Tianyu Liu, and Thuy-Duong Vuong. On the complexity of sampling redistricting plans, 2022.

[12] Jowei Chen, Christopher S. Elmendorf, Ruth Greenwood, Theresa J. Lee, Nicholas O. Stephanolpoulos, and Christopher S. Warshaw. Brief of amici curiae professors Jowei Chen, Christopher S. Elmendorf, Nicholas O. Stephanolpoulos, and Christopher S. Warshaw in support of Appellees/Respondents. Merrill v. Milligan; Merrill v. Caster; United States Supreme Court, 2022. nos. 21-1086, 21-1087., 2022. URL `https://www.supremecourt.gov/DocketPDF/21/21-1086/230239/20220718132621523_91539%20HARVARD%20BRIEF%20PROOF3.pdf`.

[13] Jeanne N. Clelland, Nicholas Bossenbroek, Thomas Heckmaster, Adam Nelson, Peter Rock, and Jade VanAusdall. Compactness statistics for spanning tree recombination, 2021.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[15] Daryl DeFord and Moon Duchin. Redistricting reform in Virginia: Districting criteria in context. *Virginia Policy Review*, XII:120–146, Spring 2019.

[16] Daryl DeFord, Moon Duchin, and Justin Solomon. Recombination: A family of Markov chains for redistricting, 2019.

[17] Daryl DeFord, Moon Duchin, and Justin Solomon. A computational approach to measuring vote elasticity and competitiveness. *Statistics and Public Policy*, 7(1):69–86, 2020.

[18] Moon Duchin. Introduction. *Political Geometry*, pages 1–28, 2022.

[19] Moon Duchin, Jeanne Clelland, Daryl DeFord, Jordan Ellenberg, Tyler Jarvis, Nestor Guillen, Dmitry Morozov, Elchanan Mossel, Dana Randall, Justin Solomon, Ari Stern, Guy-Uriel Charles, Luis Fuentes-Rohwer, Anna Dorman, Dana Paikowsky, and Robin Tholin. Amicus brief of mathematicians, law professors, and students in support of appellees and affirmance. Rucho v. Common Cause, United States Supreme Court, 2019, 2019. URL `https://mggg.org/SCOTUS-MathBrief.pdf`.

[20] Benjamin Fifield, Michael Higgins, Kosuke Imai, and Alexander Tarr. Automated redistricting simulation using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 29(4):715–728, 2020.

[21] Alan Frieze and Wesley Pegden. *Subexponential mixing for partition chains on grid-like graphs*, pages 3317–3329. Society for Industrial and Applied Mathematics, 2023.

[22] Stephan Ramon Garcia and Roger A. Horn. *A Second Course in Linear Algebra*. Cambridge Mathematical Textbooks. Cambridge University Press, 2017. doi: 10.1017/9781316218419.

[23] Geoffrey R. Grimmett. An upper bound for the number of spanning trees of a graph. *Discrete Math.*, 16(4):323–324, 1976. doi: 10.1016/S0012-365X(76)80005-2. URL `https://doi.org/10.1016/S0012-365X(76)80005-2`.

[24] Gregory Herschlag, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier, and Jonathan C. Mattingly. Quantifying gerrymandering in North Carolina. *Statistics and Public Policy*, 7(1):30–38, 2020.

[25] S. W. Hess, J. B. Weaver, H. J. Siegfeldt, J. N. Whelan, and P. A. Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13 (6):998–1006, 1965. ISSN 0030364X, 15265463. URL `http://www.jstor.org/stable/167658`.

[26] G. Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.18471481202`.

[27] A. V. Kostochka. The number of spanning trees in graphs with a given degree sequence. *Random Structures & Algorithms*, 6(2-3):269–274, 1995. doi: https://doi.org/10.1002/rsa.3240060214. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.3240060214`.

[28] Brendan D. McKay. Spanning trees in regular graphs. *European Journal of Combinatorics*, 4(2):149–160, 1983. ISSN 0195-6698. doi: https://doi.org/10.1016/S0195-6698(83)80045-6. URL `https://www.sciencedirect.com/science/article/pii/S0195669883800456`.

[29] Stuart S. Nagel. Simplified bipartisan computer redistricting. *Stanford Law Review*, 17(5):863–899, 1965. ISSN 00389765. URL `http://www.jstor.org/stable/1226994`.

[30] Ariel D. Procaccia and Jamie Tucker-Foltz. Compact redistricting plans have many spanning trees, 2021.

[31] Shahriar Shahriari. *An Invitation to Combinatorics*. Cambridge Mathematical Textbooks. Cambridge University Press, 2021. doi: 10.1017/9781108568708.

[32] Robert Shrock and F Y Wu. Spanning trees on graphs and lattices in$d$ dimensions. *Journal of Physics A: Mathematical and General*, 33(21):3881–3902, may 2000. doi: 10.1088/0305-4470/33/21/303.

[33] Alma Steingart. Law, computing and redistricting in the 1960s. *Political Geometry*, pages 163–177, 2022.

[34] Kristopher Tapp. Spanning tree bounds for grid graphs, 2022.

[35] Elmar Teufl and Stephan Wagner. On the number of spanning trees on various lattices. *Journal of Physics A: Mathematical and Theoretical*, 43: 415001, 09 2010. doi: 10.1088/1751-8113/43/41/415001.

[36] Derek A. Waller. Regular eigenvalues of graphs and enumeration of spanning trees. In *Proc. Coll on the Theory of Combinatorics, Rome*, volume 1, pages 313–320, 1973.

[37] Douglas West. *Introduction to Graph Theory.* Prentice Hall, 2001.

[38] Zhanzhan Zhao, Cyrus Hettle, Swati Gupta, Jonathan Christopher Mattingly, Dana Randall, and Gregory Joseph Herschlag. Mathematically quantifying non-responsiveness of the 2021 Georgia Congressional districting plan. In *Equity and Access in Algorithms, Mechanisms, and Optimization*, EAAMO '22. Association for Computing Machinery, 2022. URL `https://doi.org/10.1145/3551624.3555300`.