Supplement to Mathematics of Nested Districts: The Case of Alaska

Sophia Caldera*
Harvard University
Daryl DeFord
CSAIL, Massachusetts Institute of Technology
Moon Duchin
Department of Mathematics, Tufts University
Samuel C. Gutekunst
Cornell University
Cara Nix
University of Minnesota

April 9, 2020

Abstract

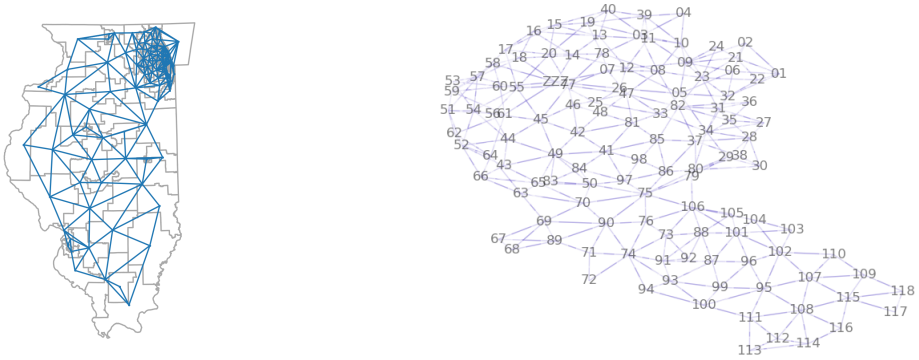
This supplement contains additional plots and technical descriptions of the algorithms used in the paper for enumerating and sampling perfect matchings. Appendix A shows reference figures of the state House dual graphs for the non-Alaska states with strict nesting rules. Further information about the FKT algorithm is given in Appendix B while Appendix C introduces the new Prune–and–Choose algorithm and provides a proof of correctness. In Appendix D, these algorithms are applied to all states with a nesting rule to compare the relative scales of the enumeration problem. Finally, in Appendix E we implement and validate a uniform sampling procedure for matchings that can be applied even in the states where generating all of the matchings would be computationally infeasible.

*Author order is alphabetical. The authors acknowledge the Prof. Amar G. Bose Research Grant at MIT and the Jonathan M. Tisch College of Civic Life at Tufts University for ongoing support. MD is partially supported by NSF DMS-1255442. SG is partially supported by NSF DGE-1650441. SG thanks David P. Williamson and Kenrick Bjelland for helpful discussions. We thank Coly Elhai, Mallory Harris, Claire Kelling, Samir Khan, and Jack Snoeyink for substantial joint work and conversations on other approaches to modeling Alaska redistricting, and particularly acknowledge Samir Khan for his initial implementation of prune-and-choose. Hakeem Angulu, Ruth Buck, and Max Hully provided excellent data and technical support. Finally, we thank Anchorage Assemblyman Forrest Dunbar for bringing this problem to our attention.

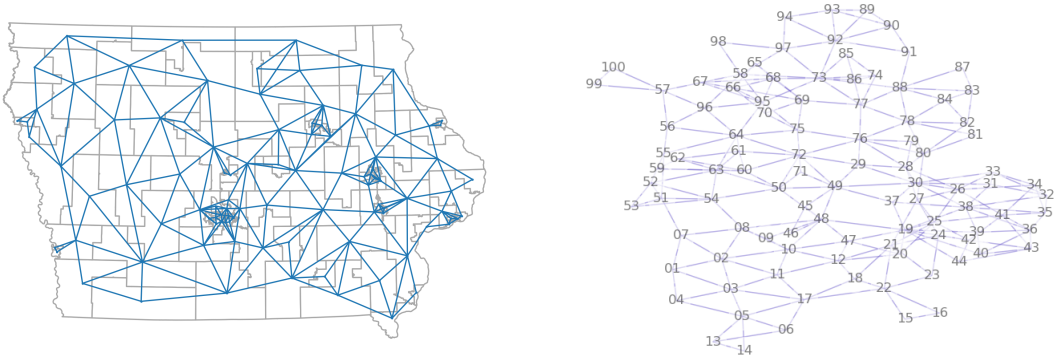
A Dual Graphs of States with Nesting Rules

In this appendix we show the dual graphs for the other states that require two-to-one nesting. Corresponding plots for Alaska are shown in Figure 7 of the main text. The left-hand column shows the House districts, with the dual graph overlaid, and the right-hand column shows a nearly-planar embedding with accurate district labels. Given these House districts, valid Senate plans in these states correspond to perfect matchings of these graphs.

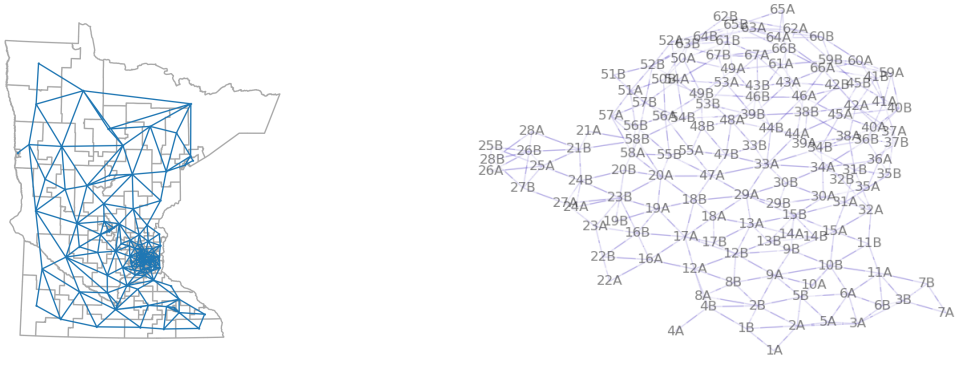
Illinois



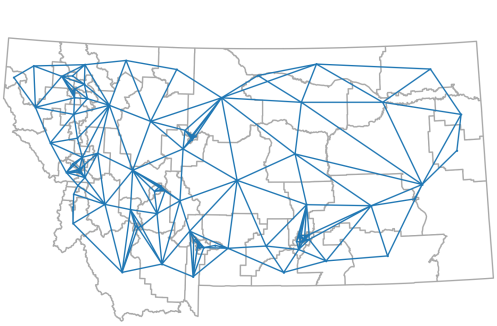
Iowa

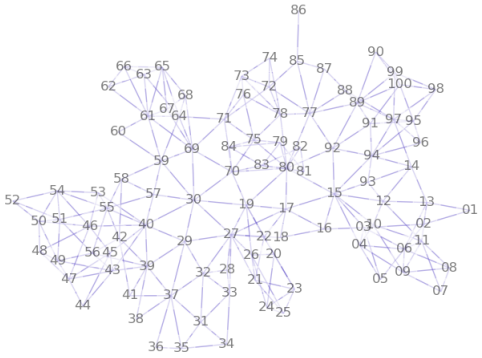


Minnesota

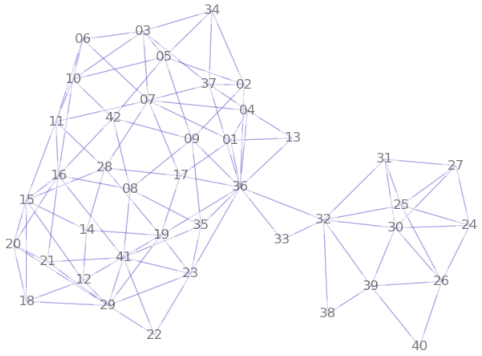


Montana

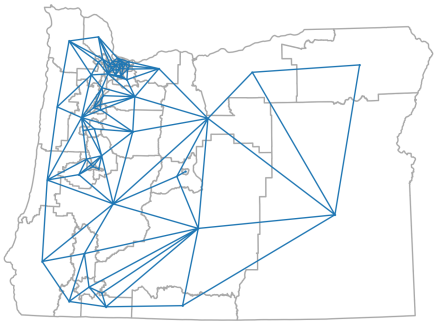


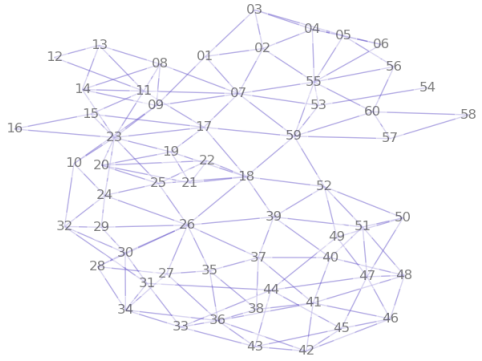


Nevada

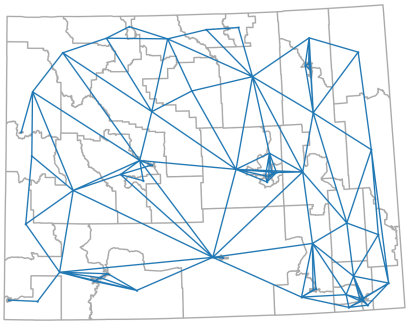


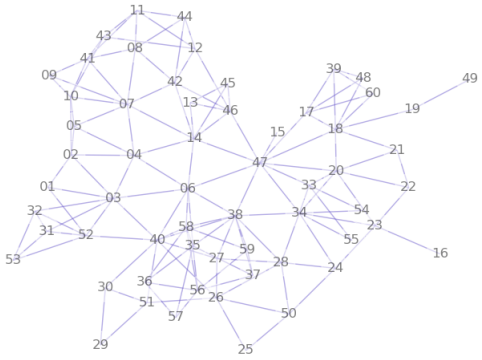
Oregon





Wyoming





B FKT algorithm for enumerating perfect matchings

As described in Section 1.2 of the main text, Fisher, Kasteleyn, and Temperley designed a method now known as the FKT algorithm for enumerating the perfect matchings of a planar graph. This allows us to quickly determine the number of perfect matchings in a dual graph, allowing us to evaluate whether it is computationally tractable to explicitly list all matchings. FKT takes a planar embedding of a graph G as input, then assigns signs to the edges in what is called a *Pfaffian orientation* (every face should have an odd number of counterclockwise edges) to create a signed, skew-symmetric adjacency matrix A. Then $\sqrt{\det A}$ counts the perfect matchings.[1]

This algorithm runs in fractions of a second on each graph, which is fast enough to incorporate at each step of a Markov chain. Our implementation is freely available at [7], and timing details are provided in §D.

C Prune-and-choose algorithm for constructing perfect matchings

In order to evaluate the partisan properties of the pairings of House districts, it is not sufficient to count matchings; we also need to generate and examine the full list of matchings. In this section, we describe a simple method to create a list of all possible perfect matchings of a graph. This is a recursive method that simplifies the search by looking for forced pairs.

The first step is to prune the graph. This means finding all *leaves* of the graph (nodes of degree one, i.e., House districts that are only connected to one other district) and matching each with its only neighbor. We call these matches *forced pairs*. One round of pruning may create new nodes of degree one in the resulting graph, and so we iteratively prune forced pairs until there are no nodes of degree one left.

The second step is a simple check to rule out a parity obstruction to the existence of a matching. If any connected component has an odd number of nodes, then it cannot be perfectly matched, so the whole graph also fails to have a perfect matching. If all connected

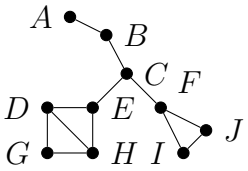
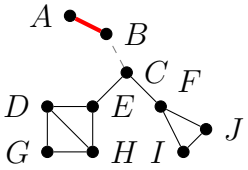
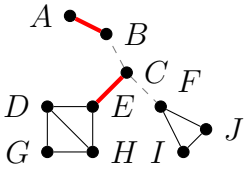
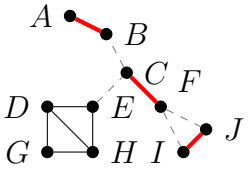
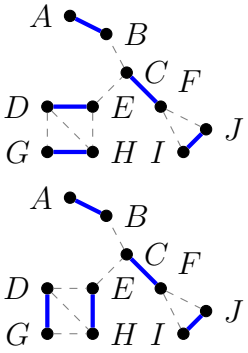
[1] The Pfaffian is a general matrix operation that agrees with $\sqrt{\det A}$ for skew-symmetric matrices. See [6] for more mathematical details.

components have even numbers of nodes, then we proceed.

Next is a choice step. From the remaining graph, we choose a node of minimum degree, then consider pairing it with each of its neighbors. For each of those pairings, we remove both nodes from the graph and apply our algorithm to what remains. We prune, check, choose, and iterate, in sequence, until the process terminates at a connected graph of two nodes, producing a perfect matching of the original graph. We provide a proof of correctness in Appendix C.2 and an example run of the algorithm on a small graph below.

C.1 Prune-and-Choose Example

A run of our algorithm on a sample graph with ten nodes proceeds as follows.

Initiate.	
A has degree one, so we record AB as a forced pair. Remove A, B from the graph. Test for parity. There is one remaining component with eight nodes, so we pass the parity check.	
C is lowest-indexed node of degree 2, so try to pair with E. However, one complementary component (FIJ) has an odd number of nodes, so we abandon this branch of the decision tree.	
Pairing C instead with F and removing both from the graph forces the IJ pairing.	
Now E is the lowest-indexed node of minimal degree. By pairing ED, the next forced pairing completes the matching. Similarly for EH.	

In the end, we find the two perfect matchings $AB/CF/IJ/ED/GH$ and $AB/CF/IJ/EH/GD$.

C.2 Prune-and-choose algorithm validity

In this section we formally describe the prune-and-choose method and provide a proof of correctness. Pseudo-code for the algorithm is given here and our implementation in Python is available at [7]. We introduce some additional notation to describe the method. The subgraph of G induced by deleting nodes u and v will be denoted $G \setminus \{u, v\}$. We will represent a matching as a set of edges $M = \{(u_1, v_1), (u_2, v_2), \ldots, (u_\ell, v_\ell)\}$. We assume that the vertices of G are ordered in order to provide a deterministic algorithm. To generate the full set of matchings for a graph G, we would call FindMatchings$(G, \emptyset)$.

Algorithm 1 Pseudo-code for Prune-and-Choose Algorithm to Find All Perfect Matchings in a Graph G

procedure FindMatchings(G, M) ▷ Input a graph G and the current set of matched edges M
  if G is connected and has exactly two vertices u, v **then**
    return $G \setminus \{u, v\}$, $M \cup (u, v)$
  else if G has any vertex with exactly one neighbor **then**
    prune: let u be the first degree-one vertex; let v be its neighbor
    return FindMatchings$(G \setminus \{u, v\}, M \cup (u, v))$ ▷ Pair forced vertices and recurse.
  else if G contains a component with an odd number of vertices **then**
    break ▷ There are no perfect matchings in G
  else
    let u be first vertex with a minimum number of neighbors $v_1, ..., v_k$
    for $1 \leq i \leq k$, let $G_i = G \setminus \{u, v_i\}$ and $M_i = M \cup (u, v_i)$
    return $\bigcup_{i=1}^{k}$FindMatchings(G_i, M_i) ▷ Recurse to find all perfect matchings with each pair

We next show that the algorithm returns the correct set of perfect matchings on any graph. They key idea of the algorithm and the proof is that for any edge of the graph, the set of perfect matchings that contain edge (u, v) can be computed by finding all perfect matchings in the subgraph $G \setminus \{u, v\}$. This is an example of the self–reducible nature of the perfect matching problem which is discussed in more detail below.

Theorem 1. *The prune-and-choose algorithm correctly finds all perfect matchings in the input graph.*

Proof. We consider any graph G with $n = 2k$ vertices and proceed by induction on k. When $k = 1$, G is either connected (in which case the algorithm correctly finds the unique

perfect matching at lines 2-3) or has two isolated vertices and no perfect matchings (which the algorithm correctly reports in lines 7-8).

For $k > 1$ the algorithm proceeds according to exactly one of the following three cases:

1. If G contains a leaf u with neighbor v, then u must be matched to v in any perfect matching. Line 6 then calls FindMatchings on $G \setminus \{u, v\}$ which returns the correct set of matchings of $G \setminus \{u, v\}$, by our inductive hypothesis. Adding (u, v) to each matching returned by this function gives the full set of matchings for G.

2. If G contains no leaves and some connected component of G has an odd number of vertices, then there are no perfect matchings in G and the algorithm correctly terminates at lines 7-8.

3. If G contains no leaves and each connected component of G has an even number of vertices, then there exists a vertex of minimal index u which has a minimum number of neighbors. Since G has no leaves and no odd components, u has degree at least 2. In any perfect matching, it must be matched to one of its neighbors $v_1, ..., v_k$. The algorithm considers each possibility calling FindMatchings on $G \setminus \{u, v_i\}$ which returns the correct set of matchings by our inductive hypothesis. As in step 1, adding (u, v_i) to the matchings returned on $G \setminus \{u, v_i\}$ provides a complete set of matchings for G.

Thus for any graph G, the first pass through the algorithm either returns $\emptyset$, which only occurs if G has no perfect matchings, or it calls the algorithm recursively on a graph of size $2(k-1)$. These recursive calls satisfy our inductive hypothesis and hence we obtain the complete set of matchings for G. □

We note that well-known classes of planar graphs have exponentially many perfect matchings. For example, this is true of the $n \times n$ grids [1, 5, 9]. This trivially implies that there is no polynomial-time algorithm to list them all as output. As we discuss in Appendix D, the dual graphs of real-world districting plans often have more perfect matchings than a grid graph of comparable size. In that section we also provide timing results for our algorithm that demonstrate that it is adequately fast for several problems at realistic scale,

but not all. In Appendix E below we show how a sampling approach can be employed to those settings in which listing all perfect matchings is computationally infeasible.

D Enumerating matchings

Here we apply FKT and Prune-and-Choose to compute the number of potential matchings for each of the eight states that require Senate districts to be formed from adjacent pairs of House districts. We use the standard Census shapefiles to generate dual graphs for each state, making them parallel to the permissive graph for Alaska. Visualizations of these dual graphs are shown in Appendix A.

	Alaska	Illinois	Iowa	Minnesota
House districts	40	118	100	134
Dual edges	100	326	251	260
Matchings	108,765	9,380,573,911	1,494,354,140,511	6,156,723,718,225,577,984
FKT runtime	0.027 sec	0.39 sec	0.21 sec	0.53 sec

	Montana	Nevada	Oregon	Wyoming
House districts	100	42	60	60
Dual edges	269	111	158	143
Matchings	11,629,786,967,358	313,698	229,968,613	920,864
FKT runtime	0.24 sec	0.038 sec	0.079 sec	0.056 sec

Table 1: Number of matchings possible with respect to the current House plan for each state (with dual graphs generated from census shapefiles) and timings for computing them with FKT.

Alaska's 108,765 (permissive) matchings are the fewest among the eight states. This is partially due to the fact that Alaska has fewer House districts than the other states, and partly due to lower edge density and several forced matchings.[2] The number of matchings varies greatly across the other states, with Minnesota having the most at 6.1×10^{18}, over six quintillion. By contrast, the number of matchings in a 10×10 grid with 100 nodes is 258,584,046,368 (fewer than Iowa, which has 100 House districts) and a 12×12 grid with 144 nodes has 53,060,477,521,960,000 (fewer than Minnesota, which has 134 House districts). To understand why the states' matching numbers exceed those of comparably

[2]The four districts in the Southeast corner of the state must be paired (33–34 and 35–36), further restricting the possible matchings.

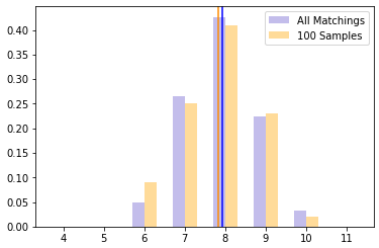
sized grids, consider the impact of just a few extra edges. Adding just four edges to the 10×10 grid—a single diagonal edge from each of the four corner vertices to its diagonal neighbor—increases the number of matchings by 745,241,088.

Extrapolating the prune-and-choose timing from Nevada (299 seconds) and Wyoming (851 seconds) suggests that generating all of the matchings for some of the other states would take prohibitively long—even with linear scaling, the Wyoming timing suggests that the Minnesota computation would take some 180 million years. However, it is possible to sample matchings from planar graphs uniformly, allowing for good estimates of relevant statistics. We have implemented the technique suggested in [4] for this purpose [7] and in Appendix E we validate this approach on Alaska.

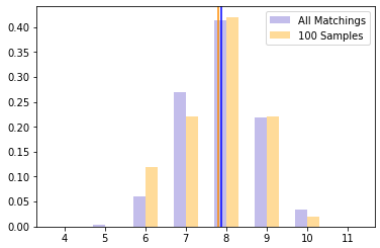
E Sampling and extremization over matchings

For Minnesota's 6.1 quintillion matchings, it would be prohibitively inefficient to list them all, no matter the algorithmic design. On the other hand, we can construct uniform samples of the full set of matchings by making use of the *self-reducible structure* in the perfect matching problem [4] as follows. We can compute the likelihood that a given edge appears in a perfect matching by deleting the edge from the graph and enumerating the matchings on the remaining nodes with FKT. The ratio of matchings on the leftover to total matchings is the probability that the edge is used. With this, we can iterate, starting with the original graph and adding a single edge to the matching at each step with appropriate probability. Since FKT runs in polynomial time, so does our sampling procedure, since a perfect matching requires $\frac{n}{2}$ edges and finding the probabilities used to select each edge requires at most $\binom{n}{2}$ FKT evaluations.

We next demonstrate that the uniform sampling method can attain good accuracy with a reasonably small number of samples, using the case of Alaska where we can compare to the ground truth from the full matching set. For each of our three dual graphs, we sample 100 matchings uniformly and compare the resulting statistics to those of the full set of matchings. Figure 1 shows these comparisons. Although the distributions are not identical, they are quite similar and the sample means vary only by small fractions of a seat from the actual values.



Cong18-A D Senate seats



Gov 18-A D Senate seats

abs. error	Tight	Restricted	Permissive
Cong18-A	0.0082	0.0234	0.0868
Gov18-A	0.0203	0.0361	0.0814

Figure 1: Comparison of number of Democratic Senate districts in a uniform sample of 100 (permissive) matchings to the full collection of matchings. The table shows the absolute error in the average seats total for this and the other two Alaska dual graphs. The histograms show more detail, and illustrate how close the averages are with only 1/1000 of the space being sampled.

This example shows that even a sample of modest size produces a good estimate of the full distribution. This provides support for our assertion that this procedure can be carried out successfully on states like Minnesota, where it would be computationally infeasible to generate all matchings. We note that all materials are available in our code repositories for others to perform this sampling for the other matching states, but there will be a non-trivial data setup cost in choosing appropriate election data and cleaning it for the analysis.

Though the histograms above are quite similar, the sample fails to capture the full range of seat outcomes in the Governor's race: a small number of possible matchings result in five D seats, but that is never observed in the sample. A second algorithm may be employed to provably find the correct range of seats outcomes possible, again without fully listing the matchings. Finding perfect matchings of extremal weight, given an edge-weighted graph, is a classic problem in combinatorics, solved for instance with the Blossom algorithm developed by Edmonds in the 1960s [2, 3]. To apply that in this setting, we use any given pattern of votes to assign a weight to each edge of our dual graph: an edge $\{u, v\}$ linking two House districts u and v is given weight 1 if there are more D than R votes in the hypothetical Senate district that combines u and v. Otherwise, assign weight 0. The weight of the perfect matching is defined as the sum of the weights of its edges.

By construction, this is the number of D seats in that matching. For more background on extremal perfect matchings, see for instance Chapters 25-26 of [8].

As a final note, knowing these extremes also informs the size of a uniform sample necessary to estimate the true distribution to a desired precision. A detailed discussion of the precise number of samples needed for various estimates is presented in [10]. In particular, Theorem 5.3 shows that with failure rate δ, taking $\max\left(\frac{4}{\varepsilon^2}, \frac{4\ln(\frac{1}{\delta})}{\varepsilon^2}\right)$ samples suffices to estimate the probability of each individual outcome to within ε (i.e., an L^∞ bound) whereas $\max\left(\frac{4n}{\varepsilon^2}, \frac{8\ln(\frac{1}{\delta})}{\varepsilon^2}\right)$ samples suffice to bound the sum of the absolute differences between the individual estimates and the true probabilities (an L^1 bound).

References

[1] Anders Björner and Richard P Stanley. *A combinatorial miscellany*. L'Enseignement mathématique, 2010.

[2] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[3] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[4] Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169 – 188, 1986.

[5] P. W. Kasteleyn. The statistics of dimers on a lattice : I. The number of dimer arrangements on a quadratic lattice. *Physica*, 27:1209–1225, December 1961.

[6] Nicholas Loehr. *Bijective Combinatorics*. CRC Press, 2011.

[7] Metric Geometry and Gerrymandering Group. Alaska. *GitHub repository*, 2019. https://github.com/mggg/Alaska.

[8] Alexander Schrijver. *Combinatorial optimization: Polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[9] H. N. V. Temperley and Michael E. Fisher. Dimer problem in statistical mechanics-an exact result. *The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics*, 6(68):1061–1063, August 1961.

[10] Bo Waggoner. Lp testing and learning of discrete distributions. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 347–356, New York, NY, USA, 2015. ACM.