On the Computational Tractability of a Geographic Clustering Problem Arising in Redistricting

Vincent Cohen-Addad
CNRS and Sorbonne Université, Paris, France

Philip N. Klein
Brown University, Providence, RI, USA

Dániel Marx ✉
CISPA Helmholtz Center for Information Security, Saarland Informatics Campus, Germany

Archer Wheeler
Brown University, Providence, RI, USA

Christopher Wolfram
Brown University, Providence, RI, USA

Abstract

Redistricting is the problem of dividing up a state into a given number k of regions (called *districts*) where the voters in each district are to elect a representative. The three primary criteria are: that each district be connected, that the populations of the districts be equal (or nearly equal), and that the districts are "compact". There are multiple competing definitions of compactness, usually minimizing some quantity.

One measure that has been recently been used is number of *cut edges*. In this formulation of redistricting, one is given atomic regions out of which each district must be built (e.g., in the U.S., census blocks). The populations of the atomic regions are given. Consider the graph with one vertex per atomic region and an edge between atomic regions with a shared boundary of positive length. Define the weight of a vertex to be the population of the corresponding region. A districting plan is a partition of vertices into k pieces so that the parts have nearly equal weights and each part is connected. The districts are considered compact to the extent that the plan minimizes the number of edges crossing between different parts.

There are two natural computational problems: find the most compact districting plan, and sample districting plans (possibly under a compactness constraint) uniformly at random.

Both problems are NP-hard so we consider restricting the input graph to have branchwidth at most w. (A planar graph's branchwidth is bounded, for example, by its diameter.) If both k and w are bounded by constants, the problems are solvable in polynomial time. In this paper, we give lower and upper bounds that characterize the complexity of these problems in terms of parameters k and w. For simplicity of notation, assume that each vertex has unit weight. We would ideally like algorithms whose running times are of the form $O(f(k,w)n^c)$ for some constant c independent of k and w (in which case the problems are said to be *fixed-parameter tractable* with respect to those parameters). We show that, under standard complexity-theoretic assumptions, no such algorithms exist. However, the problems *are* fixed-parameter tractable with respect to each of these parameters individually: there exist algorithms with running times of the form $O(f(k)n^{O(w)})$ and $O(f(w)n^{k+1})$. The first result was previously known. The new one, however, is more relevant to the application to redistricting, at least for coarse instances. Indeed, we have implemented a version of the algorithm and have used to successfully find optimally compact solutions to all redistricting instances for France (except Paris, which operates under different rules) under various population-balance constraints. For these instances, the values for w are modest and the values for k are very small.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases redistricting, algorithms, planar graphs, lower bounds

Digital Object Identifier 10.4230/LIPIcs.FORC.2021.3

Related Version A version without discussion of the implementation is at:
Previous Version: `http://https://arxiv.org/abs/2009.00188`

Funding *Philip N. Klein*: Supported by National Science Foundation grant CCF-1841954.
Dániel Marx: Supported by European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH.
Archer Wheeler: Supported by National Science Foundation grant CCF-1841954.

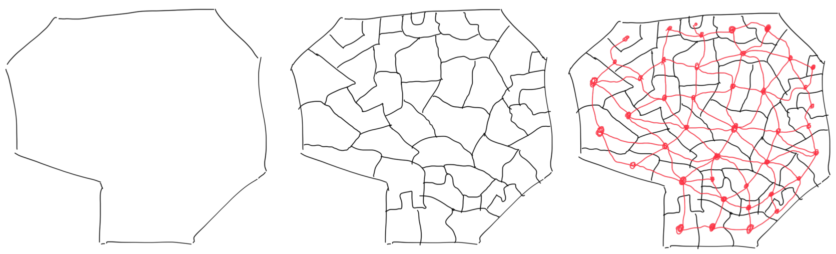


Figure 1 On the left is an imaginary state/department. In the middle, the state is subdivided into smaller regions (*atoms*), e.g. census tracts. On the right, the planar dual is shown. Each atomic region is represented by a node. (There is also a node for the single infinite region outside the state boundary but here we ignore that node here.) For each maximal contiguous boundary segment between a pair of atomic regions, the planar dual has an edge between the corresponding pair of nodes.

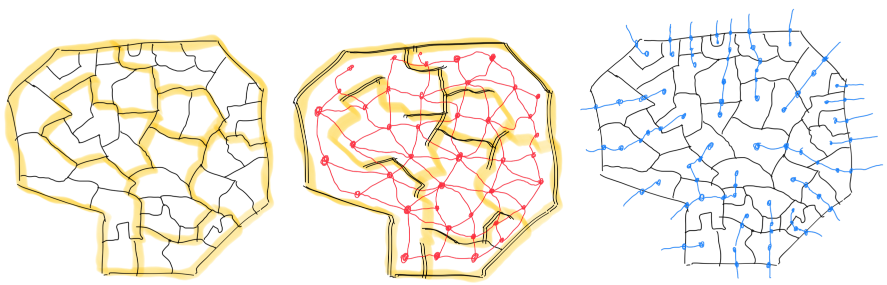


Figure 2 The figure on the left shows an example of a districting plan with seven districts. Each district is the union of several atomic regions. The figure in the middle depicts the districting plan superimposed on the planar dual, showing that it corresponds to a partition of the atoms into connected parts; the cost of the solution is the sum of costs of edges of the dual that cross between different parts. In this paper, a districting plan is compact to the extent that this sum of costs is small. The figure on the right illustrates a breadth-first search in the radial graph of the graph G of atomic regions. As stated in Section 2.2, the *radial graph* of G has a node for every vertex of G and a node for every face of G, and an edge between a vertex-node and a face-node if the vertex lies on the face's boundary. This diagram shows that every face is reachable from the outer face within six hops in the *radial graph* of the graph G of atomic regions. This implies that the branchwidth of G and of its dual are at most six.

1 Introduction

For an undirected planar graph G with vertex-weights and a positive integer k, a *connected partition* of the vertices of G is a partition into parts each of which induces a connected subgraph. If G is equipped with nonnegative integral vertex weights and $[L, U)$ is an interval we say such a partition has *part-weight* in $[L, U)$ if the sum of weights of each part lies in the interval. If G is equipped with nonnegative edge costs, we say the *cost* of such a partition is the sum of costs of edges uv where u and v lie in different parts.

Consider the following computational problems:

- *optimization:* Given a planar graph G with vertex weights and edge costs, a number k, and a weight interval $[L, U)$, find the minimum cost of a partition into k connected parts with part-weight in $[L, U)$.
- *sampling:* Given in addition a number C, generate uniformly at random a cost-C partition into k connected parts with part-weight in $[L, U)$.

These problem arise in political redistricting. Each vertex represents a small geographical region (such as a *census block* or *census tract* or *county*), and its weight represents the number of people living in the region. Each part is a *district*. A larger geographic region (such as a state) must be partitioned into k districts when the state is to be represented in a legislative body by k people; each district elects a single representative. The partition is called a *districting plan*.

The rules governing this partitioning vary from place to place, but usually there are (at least) three important goals: *contiguity*, *population balance*, and *compactness*.[1]

- *Contiguity* is often interpreted as connectivity; we represent this by requiring that the set of small regions forming each district is connected via shared boundary edges.
- *Population balance* requires that two different districts have approximately equal numbers of people.
- One measure of *compactness* that has been advocated e.g. by DeFord, Duchin, Solomon, and Tenner [7, 8, 11, 12] is the number of pairs of adjacent small regions that lie in distinct districts, equivalent to the cardinality of the *cut-set* corrresponding to the partition.

Thus in the definitions of the *optimization* and *sampling* problems above, the connectivity constraint reflects the contiguity requirement, the part-weight constraint reflects the population balance requirement, and the cost is a measure of compactness.

The *optimization* problem described above arises in computer-assisted redistricting; an algorithm for solving this problem could be used to select a districting plan that is optimally compact subject to contiguity and desired population balance, where compactness is measured as discussed above.

The *sampling* problem arises in evaluating a plan; in court cases [4, 35, 24, 23, 36] expert witnesses argue that a districting plan reflects an intention to gerrymander by comparing it to districting plans randomly sampled from a distribution. The expert witnesses use Markov Chain Monte Carlo (MCMC), albeit unfortunately on Markov chains that have not been shown to be rapidly mixing, which means that the samples are possibly not chosen according to anything even close to a uniform distribution. There have been many papers addressing random sampling of districting plans (e.g. [1, 3, 8, 23, 24]) but, despite the important role of random sampling in court cases, there are no results on provably uniform or nearly uniform sampling from a set of realistic districting plans for a realistic input in a reasonable amount of time.

It is known that even basic versions of these problems are NP-hard. If the vertex weights are allowed to very large integers, expressed in binary, the NP-hardness of SUBSET SUM already implies the NP-completeness of partitioning the vertices into two equal-weight subsets. However, in application to redistricting the integers are not very large. For the purpose of seeking hardness results, it is better to focus on a special case, the *unit-weight* case, in which each vertex has weight one. For this case, Dyer and Frieze [13] showed that, for any fixed $p \geq 3$, it is NP-hard to find a weight-balanced partition of the vertices of a planar graph into connected parts of size p. Najt, Deford, and Solomon [33] showed that even without the constraint on balance, uniform sampling of partitions into *two* connected parts is NP-hard.

Following Ito et al. [27, 26] and Najt et al. [33], we therefore consider a further restriction on the input graph: we consider graphs with bounded *branchwidth*/*treewidth*.[2]

[1] These terms are often not formally defined in law.
[2] *Treewidth* and branchwidth are very similar measures; they are always within a small constant factor of each other. Thus a graph has small treewidth if and only if it has small branchwidth.

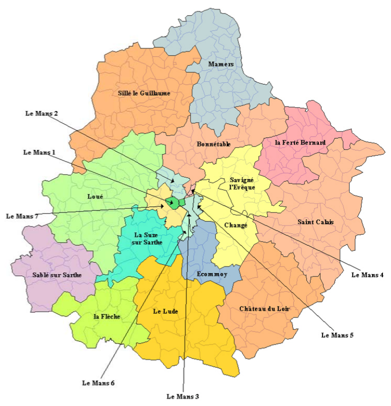


Figure 3 This map shows the twenty-one cantons for the department "Sarthe" of France. The cantons are the atomic regions for the redistricting of Sarthe. The corresponding radial graph has radius six, so there is a branch decomposition of width $w = 6$. For the upcoming redistricting of France, Sarthe must be divided into $k = 3$ districts.

The branchwidth of a graph is a measure of how treelike the graph is: often even an NP-hard graph problem is quickly solvable when the input is restricted to graphs with low branchwidth. For planar graphs in particular, there are known bounds on branchwidth that are relevant to the application. A planar graph on n vertices has branchwidth $O(\sqrt{n})$, and a planar graph of diameter d has branchwidth $O(d)$. There is an stronger bound, which we will review in Section 2.2.

Najt, Deford, and Solomon [33] show that, for any fixed k and fixed w, the optimization and sampling problems *without the constraint on population balance* can be solved in polynomial time on graphs of branchwidth at most w.[3] Significantly, the running time is of the form $O(f(k, w)n^c)$ for some constant c. Such an algorithm is said to be *fixed-parameter tractable* with respect to k and w, meaning that as long as k and w are fixed, the problem is considered tractable. Fixed-parameter tractability is an important and recognized way of coping with NP-completeness.

However, their result has two disadvantages. First, as the authors point out, the big O hides a constant that is astronomical; for NP-hard problems, one expect that the dependence on the parameters be at least exponential but in this case it is a tower of exponentials. As the authors state, the constants in the theorems on which they rely are "too large to be practically useful."

Second, because their algorithm cannot handle the constraint on population balance, the algorithm would not be applicable to redistricting even if it were tractable. The authors discuss (Remark 5.11 in [33]) the extension of their approach to handle balance: "It is easy to add a relational formula...that restricts our count to only balanced connected k-partitions.... From this it should follow that ... [the problems are tractable]. However ... the corresponding meta-theorem appears to be missing from the literature."

In our first result, we show that in fact what they seek does not exist: under a standard complexity-theoretic assumption, **there is no algorithm that is fixed-parameter tractable with respect to both k and w.**

More precisely, we use the analogue of NP-hardness for fixed-parameter tractability, $W[1]$-hardness. We show the following in Section 4.

[3] They use treewidth but the results are equivalent.

▶ **Theorem 1.** *For unit weights, finding a weight-balanced k-partition of a planar graph of width w into connected parts is $W[1]$-hard with respect to $k + w$.*

In the theory of fixed-parameter tractability (see e.g. Section 13.4 of [6]) this is strong evidence that no algorithm exists with a running time of the form $O(f(k, w)n^c)$ for fixed c independent of k and w.

This is bad news but there is a silver lining. The lower bound guides us in seeking good algorithms, and it does not rule out an algorithm that has a running time of the form $f(k)n^{O(w)}$ or $f(w)n^{O(k)}$. That is, according to the theory, while there is no algorithm that is fixed-parameter tractable with respect to both k and w simultaneously, there *could* be one that is fixed-parameter tractable with respect to k alone and one that is fixed-parameter tractable with respect to w alone.

These turn out to be true. First we discuss fixed-parameter tractability with respect to k. Ito et al. [27, 26] show that, even for general (not necessarily planar) graphs there is an algorithm with running time $O((w + 1)^{2(w+1)}U^{2(w+1)}k^2n)$, where U is the upper bound on the part weights. Thus for unit weights, the running time is $O((w + 1)^{2(w+1)}n^{2w+3})$.

However, for the application we have in mind this is not the bound to try for. Indeed, the motivation for this project arose from a collaboration between the first author and some other researchers. That team, in anticipation of the upcoming redistricting in France, sought to find good district plans with respect to various criteria for French departments. Their approach was to develop code that, for each department, would explicitly enumerate all district plans that (a) are connected and (b) are population-balanced to within 20% of the mean. Their effort succeeded on all but three departments (not including Paris, which follows different rules): Doubs (25), Saône-et-Loire (71) and, Seine-Maritime (76). The question arose: could another algorithmic approach succeed in finding optimal district plans for these under some objective function? We observed that the numbers of districts tend to be *very* small (sixty-three out of about a hundred departments have between two and five districts, and the average is a little over three.) The number of atoms of course tends to be much larger, but the diameter of the graph is often not so large, and hence the same is true for branchwidth.[4]

Thus, to address such instances, we need an algorithm that can tolerate a very small number k of districts and a moderately small branchwidth w. We prove the following in Section 5.

▶ **Theorem 2.** *For the optimization problem and the sampling problem, there are algorithms that run in $O(c^wU^kSn(\log U + \log S))$ time, where c is a constant, k is the number of districts, $w \geq k$ is an upper bound on the branchwidth of the planar graph, n is the number of vertices of the graph, U is the upper bound on the weight of a part, and S is an upper bound on the cost of a desired solution.*

Remarks.

1. In the unit-cost case (every edge cost is one), $S \leq n$.
2. In the unit-weight, unit-cost case, the running time is $O(c^wn^{k+2}\log n)$.
3. For practical use the input weights need not be the populations of the atoms; if approximate population is acceptable, the weight of an atom with population p can be, e.g., $\lceil p/1000 \rceil$.

[4] For example, the French redistricting instances all have branchwidth at most eight; the average is about five.

In order to demonstrate that the theoretical algorithm is not inherently impractical, we developed an implementation for the optimization problem, and successfully applied it to find solutions for the redistricting instances in France. French law requires that the population of each department needs to be within 20% of the mean. The implementation found the cut-size-minimizing solutions subject to the 20% population balance constraint, and subject to a 10% population balance constraint. Using a 5% population balance constraint, we found optimal solutions for over half of the departments. We briefly describe the results in Section 6, and we illustrate some district plans in the full version of the paper.

2 Preliminaries

2.1 Branchwidth

A *branch decomposition* of a graph G is a rooted binary tree with the following properties:

1. Each node x is labeled with a subset $C(x)$ of the edges of G.
2. The leaves correspond to the edges of G: for each edge e, there is a leaf x such that $C(x) = \{e\}$.
3. For each node x with children x_1 and x_2, $C(x)$ is the disjoint union of $C(x_1)$ and $C(x_2)$.

We refer to a set $C(x)$ as a *branch cluster*. A vertex v of G is a *boundary vertex* of $C(x)$ if G has at least one edge incident to v that is in $C(x)$ and at least one edge incident to v that is not in $C(x)$. The *width* of a branch cluster is the number of boundary vertices, and the width of a branch decomposition is the maximum cluster width. The branchwidth of a graph is the minimum w such that the graph has a branch decomposition of width w.

For many optimization problems in graphs, if the input graph is required to have small branchwidth then there is a fast algorithm, often linear time or nearly linear time, and often this algorithm can be adapted to do uniform random sampling of solutions. Therefore Najt, Deford, and Solomon [33] had good reason to expect that there would be a polynomial-time algorithm to sample from balanced partitions where the degree of the polynomial was independent of w and k.

2.2 Radial graph

For a planar embedded graph G, the radial graph of G has a node for every vertex of G and a node for every face of G, and an edge between a vertex-node and a face-node if the vertex lies on the face's boundary. Note that the radial graph of G is isomorphic to the radial graph of the dual of G. There is a linear-time algorithm that, given a planar embedded graph G and a node r of the radial graph, returns a branch decomposition whose width is at most the number of hops required to reach every node of the radial graph from r (see, e.g., [30]). For example, Figure 2 shows that the number of hops required is at most six, so the linear-time algorithm would return a branch decomposition of width w at most six.

Using this result, some real-world redistricting graphs can be shown to have moderately small branchwidth. For example, Figure 3 shows a department of France, Sarthe, that will need to be divided into $k = 3$ districts. The number of hops required for this example is six, so we would get a branch decomposition of width w at most six.

2.3 Sphere-cut decomposition

The branch decomposition of a planar embedded graph can be assumed to have a special form. The radial graph of G can be drawn on top of the embedding of G so that a face-node is embedded in the interior of a face of G and a vertex-node is embedded in the same location as the corresponding vertex. We can assume that the branch decomposition has the property that corresponding to each branch cluster C is a cycle in the radial graph that encloses exactly the edges belonging to the cluster C, and the vertices on the boundary of this cluster are the vertex-nodes on the cycle. This is called a *sphere-cut decomposition* [10]. If the branch decomposition is derived from the radial graph using the linear-time algorithm mentioned above, the sphere-cut decomposition comes for free. Otherwise, there is an $O(n^3)$ algorithm to find a given planar graph's least-width branch decomposition, and if this algorithm is used it again gives a sphere-cut decomposition.

3 Related work

There is a vast literature on partitioning graphs, in particular on partitions that are in a sense balanced. In particular, in the area of decomposition of planar graphs, there are algorithms [37, 34, 38] for *sparsest cut* and *quotient cut*, in which the goal is essentially to break off a single piece such that the cost of the cut is small compared to the amount of weight on the smaller side. The single piece can be required to be connected. There are approximation algorithms for variants of balanced partition [19, 17] into two pieces. These only address partitioning into $k = 2$ pieces, the pieces are not necessarily connected, and the balance constraint is only approximately satisfied. In one paper [29], the authors use a variant of binary decision diagrams to construct a compact representation of all partitions of a graph into k connected parts subject to a balance constraint. However, their algorithm does not address the problem of minimizing the size of the cut-set.

There are many papers on algorithms relevant to computer-aided redistricting (a few examples are [5, 14, 22, 25, 32, 18]). Note that in this paper we focus on algorithms that have guaranteed polynomial running times (with respect to fixed parameters k and w) and that are guaranteed to find optimal solutions or that provably generate random solutions according to the uniform distribution. There has been much work on using Markov Chain Monte Carlo as a heuristic for optimization or for random generation but so far such methods are not accompanied by mathematical guarantees as to running time or quality of output.

Finally, there many papers on $W[1]$-hardness and more generally lower bounds on fixed-parameter tractability, as this is a well-studied area of theoretical computer science. Our result is somewhat rare in that most graph problems are fixed-parameter tractable with respect to branchwidth/treewidth. However, there are by now other $W[1]$-hardness results with respect to treewidth [9, 2, 16, 31, 21, 20] and a few results [2, 15] were previously known even under the restriction that the input graph must be planar.

4 W[1]-Hardness

In this section, we show that the problem is W[1]-hard parameterized by $k + w$, where k is the number of districts and w the treewidth of the graph.

We start with the following lemma that shows that it is enough to prove that a more structured version of the problem (bounded vertex weights, each region must have size greater than 1) is W[1]-hard.

▶ **Lemma 3.** *If the planar vertex-weighted version of the problem is W[1]-hard parameterized by $k + w$ when the total weight of each region should be greater than 1, and the smallest weight is 1 and the largest weight is polynomial in the input size, then the planar unweighted version of the problem is W[1]-hard parameterized by $k + w$.*

Proof. Consider a weighted instance of the problem satisfying the hypothesis of the lemma. Let $w_{\min}$ and $W_{\max}$ respectively denote the minimum and maximum weights. First, rescale all the weights of the vertices so as to make them integers. Since the input weights are rationals and $W_{\max}$ is polynomial in the input size, this does not change the size complexity of the problem by more than a polynomial factor. We now make the following transformations to the instance. For each vertex v of weight $w(v)$, create $w(v) - 1$ unit-weight *dummy* vertices and connect each of them to v with a single edge, then remove the weight of v.

This yields a unit-weight graph which satisfies the following properties. First, if the input graph was planar, then the resulting graph is also planar. Second, since the ratio $W_{\max}$ is polynomial in the input size, the total number of vertices in the new graph is polynomial in the input size. Finally, any solution for the problem on the vertex-weighted graph can be associated to a solution for the problem on the unit-weight graph: for each vertex v of the original graph, assign each of the $w(v) - 1$ dummy vertices to the same region as v. We have that the associated solution has connected regions of exactly the same weight as the solution in the weighted graph. Moreover, we claim that any solution for the unit-weight graph is associated to a solution of the input weighted graph: this follows from the assumption that the prescribed weights for the regions is greater than 1 and that the regions must be connected. Thus for each vertex v, in any solution all the $w(v) - 1$ dummy vertices must belong to the region of v.

Therefore, if the planar vertex-weighted version of the problem is W[1]-hard parameterized by $k + w$ when the smallest weight is at least 1, the total weight of each region should be greater than 1, and the sum of the vertex weights of the graph is polynomial in the input size, then the planar unit-weight version of the problem is W[1]-hard parameterized by $k + w$. ◀

By Lemma 3, we can focus without loss of generality on instances $G = (V, E), w : V \mapsto \mathbb{R}_+$ where the vertex weights w lie in the interval $[1, |V|^c]$ for some absolute constant c. We next show that the problem is W[1]-hard on these instances.

We reduce from the Bin Packing problem with polynomial weights. Given a set of integer values $v_1, \ldots, v_n$ and two integers B and k, the *Bin Packing* problem asks to decide whether there exists a partition of $v_1, \ldots, v_n$ into k parts such that for each part of the partition, the sum of the values is at most B. The Bin Packing problem with polynomially bounded weights assumes that there exists a constant c such that $B = O(n^c)$. Note that for the case where the weights are polynomially bounded, we can assume w.l.o.g. that the sum of the weights is exactly kB by adding $kB - \sum_{i=1}^{n} v_i$ elements of value 1. Since the weights are polynomially bounded and that each weight is integer we have that (1) the total number of new elements added is polynomial in n, hence the size of the problem is polynomial in n, and (2) there is a solution to the original problem if and only if there is a solution to the new problem: the new elements can be added to fill up the bins that are not full in the solution of the original problem.

We will make use of the following theorem of Jansen et al. [28].

▶ **Theorem 4** ([28]). *The Bin Packing problem with polynomial weights is W[1]-hard parameterized by the number of bins k. Moreover, there is no $f(k)n^{o(k/\log k)}$ time algorithm assuming the exponential time hypothesis (ETH).*

We now proceed to the proof of Theorem 1. From an instance of Bin Packing with polynomially bounded weights and whose sum of weights is kB, create the following instance for the problem. For each $i \in [2n + 1]$, create

$$\ell_i = \begin{cases} k & \text{if } i \text{ is odd} \\ k+1 & \text{if } i \text{ is even} \end{cases}$$

vertices $s_i^1, \ldots, s_i^{\ell_i}$. Let $S_i = \{s_i^1, \ldots, s_i^{\ell(i)}\}$. Moreover, for each odd $i < n$, for each $1 \le j \le k$, connect s_i^j to s_{i-1}^j and s_{i+1}^j, and when $j < k$, also to s_{i-1}^{j+1} and s_{i+1}^{j+1}. Let G be the resulting graph.

It is easy to see that G is planar. We let f_∞ be the longest face:
$\{s_1^1, \ldots, s_1^k, s_2^{k+1}, s_3^k, \ldots, s_{2n+1}^k, s_{2n+1}^{k-1}, \ldots, s_{2n+1}^1, s_{2n}^1, \ldots, s_2^1\}$.
We claim that the treewidth of the graph is at most $7k$. To show this we argue that the face-vertex incidence graph $\bar{G}$ of G has diameter at most $2k+4$ and by Lemma 3 this immediately yields that the treewidth of G is at most $10k$. We show that each vertex of $\bar{G}$ is at hop-distance at most $k+2$ of the vertex corresponding to f_∞. Indeed, consider a vertex s_i^j (for a face, consider a vertex s_i^j on that face). Recall that for each i_0, j_0, we have that $s_{i_0}^{j_0}$ is adjacent to $s_{i+1}^{j_0}$ and $s_{i+1}^{j_0+1}$ and so, s_i^j is at hop-distance at most $k+1$ from either $s_i^{\ell(i)}$ or s_i^1 in $\bar{G}$. Moreover both s_i^1 and $s_n^{\ell(i)}$ are on face f_∞ and so s_i^j is at hop-distance at most $k+2$ from f_∞ in $\bar{G}$. Hence the treewidth of G is at most $10k$.

Our next step is to assign weights to the vertices. Then, we set the weight $w(s_i^j)$ of every vertex s_i^j of $\{s_1^1, \ldots, s_1^k\}$ to be $(kB)^2$ and the weight $w(s_i^j)$ of every vertex s_i^j of $\{s_{2n+1}^1, \ldots, s_{2n+1}^k\}$ to be $(kB)^4$. For each odd $i \neq 1, 2n+1$ we set a weight of $1/(2n-2)$. Finally, we set the weight of each vertex s_i^j where i is even to be $v_{i/2}$. Let $T = (kB)^2 + (kB)^4 + 1/2 + kB$, and recall that $kB = \sum_{i=1}^n v_i$.

▶ **Fact 1.** *Consider a set S of vertices containing exactly one vertex of S_i for each i. Then the sum of the weights of the vertices in S is T.*

We now make the target weight of each region to be $(kB)^2 + (kB)^4 + kB + B = T + B$. We have the following lemma.

▶ **Lemma 5.** *In any feasible solution to the problem, there is exactly 1 vertex of $\{s_1^1, \ldots, s_1^k\}$ and exactly 1 vertex of $\{s_n^1, \ldots, s_n^{\ell(n)}\}$ in each region.*

Proof. Recall that by definition we have that $\sum_{i=1}^n v_i = kB$. Moreover, the number of vertices with weight equal to $(kB)^2$ is exactly k. Thus, since the target weight of each region is $(kB)^2 + (kB)^4 + B + kB$, each region has to contain exactly 1 vertex from $\{s_1^1, \ldots, s_1^k\}$ and exactly 1 vertex from $\{s_n^1, \ldots, s_n^{\ell(n)}\}$. ◀

We now turn to the proof of completeness and soundness of the reduction. We first show that if there exists a solution to the Bin Packing instance, namely that there is a partition into k parts such that for each part of the partition, the sum of the values is B, then there exists a feasible solution to the problem. Indeed, consider a solution to the Bin Packing instance $\{B_1, \ldots, B_k\}$ and construct the following solution to the problem. For each odd i, assign vertices $s_i^1, \ldots, s_i^k$ to regions $R_1, \ldots, R_k$ respectively. For each $i \in [n]$, perform the following assignment for the even rows. Let u_i be the integer in $[k]$ such that $v_i \in B_{u_i}$. Assign all vertices $s_{2i}^1, \ldots, s_{2i}^{u_i-1}$ to regions $R_1, \ldots R_{u_i-1}$ respectively. Assign both vertices $s_{2i}^{u_i}$ and $s_{2i}^{u_i+1}$ to region R_{u_i}. Assign all vertices $s_{2i}^{u_i+2}, \ldots s_{2i}^{k+1}$ to regions $R_{u_i+1}, \ldots R_k$. The connectivity of the regions follows from the fact that for each odd i, s_i^j is connected to both s_{i+1}^j and s_{i+1}^{j+1} and to both s_{i-1}^j and s_{i-1}^{j+1}.

We then bound the total weight of each region. Let's partition the vertices of a region R_j into two: Let S_{R_j} be a set that contains one vertex from each S_i and let $\bar{S}_{R_j}$ be the rest of the elements. The total weight of the vertices in S_{R_j} is by Fact 1 exactly T. The total weight of the remaining vertices corresponds to the sum of the values v_i such that $|R_j \cap S_i| = 2$ which is $\sum_{v_i \in B_j} v_i = B$ since it is a solution to the Bin Packing problem. Hence the total weight of the region is $T + B$, as prescribed by the problem.

We finally prove that if there exists a solution for the problem with the prescribed region weights, then there exists a solution to the Bin Packing problem. Let $R_1, \ldots, R_k$ be the solution to the problem. By Lemma 5, each region contains one vertex of $s_1^1, \ldots s_1^k$ and one vertex of $s_1^1, \ldots s_{2n+1}^k$. Since the regions are required to be connected, there exists a path joining these two vertices and so by the pigeonhole principle for each odd i, each region contains exactly one vertex of $s_i^1, \ldots s_i^k$. Moreover for each even i, each region contains at least one vertex of $s_i^1, \ldots s_i^{k+1}$ and exactly one region contains two vertices. Let $\phi(i) \in [k]$ be such that $|R_{\phi(i)} \cap \{s_i^1, \ldots s_i^{k+1}\}| = 2$. We now define the following solution for the Bin Packing problem. Define the jth bin as $B_j = \{v_i \mid \phi(i) = j\}$. We claim that for each bin B_j the sum of the weights of the elements in B_j is exactly B. Indeed, observe that region R_j contains exactly one vertex of $s_i^1, \ldots s_i^k$ for each odd i and exactly one vertex of $s_i^1, \ldots s_i^{k+1}$ for each even i except for the sets $s_i^1, \ldots s_i^{k+1}$ where $\phi(i) = j$ for which it contains two vertices. Thus by Fact 1, the total sum of the weights is $T + \sum_{i|\phi(i)=j} v_i$ and since the target weight is $T + B$ we have that $\sum_{i|\phi(i)=j} v_i = B$. Since the weight of B_j is exactly $\sum_{i|\phi(i)=j} v_i$ the proof is complete.

5 Algorithm

In this section, we describe the algorithms of Theorem 2. In describing the algorithm, we will focus on simplicity rather than on achieving the best constant possible as the base of k.

5.1 Partitions

A *partition* of a finite set Ω is a collection of disjoint subsets of Ω whose union is Ω. A partition defines an equivalence relation on Ω: two elements are equivalent if they are in the same subset.

There is a partial order on partitions of Ω: $\pi_1 \prec \pi_2$ if every part of π_1 is a subset of a part of π_2. This partial order is a lattice. In particular, for any pair π_1, π_2 of partitions of Ω, there is a unique minimal partition π_3 such that $\pi_1 \prec \pi_3$ and $\pi_2 \prec \pi_3$. (By *minimal*, we mean that for any partition π_4 such that $\pi_1 \prec \pi_4$ and $\pi_2 \prec \pi_4$, it is the case that $\pi_3 \prec \pi_4$.) This unique minimal partition is called the *join* of π_1 and π_2, and is denoted $\pi_1 \vee \pi_2$.

It is easy to compute $\pi_1 \vee \pi_2$: initialize $\pi := \pi_1$, and then repeatedly merge parts that intersect a common part of π_2.

In a slight abuse of notation, we define the join of a partition π_1 of one finite set Ω_1 and a partition π_2 of another finite set Ω_2. The result, again written $\pi_1 \vee \pi_2$, is a partition of $\Omega_1 \cup \Omega_2$. It can be defined algorithmically: iniitalize π to consist of the parts of π_2, together with a singleton part $\{\omega\}$ for each $\omega \in \Omega_2 - \Omega_1$. Then repeatedly merge parts of π that intersect a common part of π_2.

5.2 Noncrossing partitions

The sphere-cut decomposition is algorithmically useful because it restricts the way a graph-theoretic structure (such as a solution) can interact with each cluster. For a cluster C, consider the corresponding cycle in the radial graph, and let θ_C be the cyclic permutation $(v_1\ v_2\ \cdots\ v_m)$ of boundary vertices in the order in which they appear in the radial cycle. (By a slight abuse of notation, we may also interpret θ_C as the set $\{v_1, \ldots, v_m\}$.

First consider a partition ρ^{in} of the vertices incident to edges belonging to C, with the property that each part induces a connected subgraph of C. Planarity implies that the partition induced by ρ^{in} on the boundary vertices $\{v_1, \ldots, v_m\}$ has a special property.

▶ **Definition 6.** *Let π be a partition of the set $\{1, \ldots, m\}$. We say π is* crossing *if there are integers $a < b < c < d$ such that one part contains a and c and another part contains b and d.*

It follows from connectivity that the partition induced by ρ^{in} on the boundary vertices θ_C is a noncrossing partition. Similarly, let ρ^{out} be a partition of the vertices incident to edges that do *not* belong to C; then ρ^{out} induces a noncrossing partition on the boundary vertices of C.

The asymptotics of the Catalan numbers imply the following (see, e.g., [10]).

▶ **Lemma 7.** *There is a constant c_1 such that the number of noncrossing partitions of $\{1, \ldots, w\}$ is $O(c_1^w)$.*

Finally, suppose ρ is a partition of all vertices of G such that each part is connected. Then $\rho = \rho^{\text{in}} \vee \rho^{\text{out}}$ where ρ^{in} is a partition of the vertices incident to edges in C (in which each part is connected) and ρ^{out} is a partition of the vertices incident to edges not in C (in which each part is connected).

Because the only vertices in both ρ^{in} and ρ^{out} are those in θ_C, the partition ρ induces on θ_C is $\pi^{\text{in}} \vee \pi^{\text{out}}$ where π^{in} is the partition induced on θ_C by ρ^{in} and π^{out} is the partition induced on θ_C by ρ^{out}.

5.3 Algorithm overview

The algorithms for optimization and sampling are closely related.

The algorithms are based on dynamic programming using the sphere-cut decomposition of the planar embedded input graph G.

Each algorithm considers every vertex v of the input graph and selects one edge e that is incident to v, and designates each branch cluster that contains e as a *home cluster* for v.

We define a *topological configuration* of a cluster C to be a pair $(\pi^{\text{in}}, \pi^{\text{out}})$ of noncrossing partitions of θ_C with the following property:

$$\pi^{\text{in}} \vee \pi^{\text{out}} \text{ has at most } k \text{ parts.} \tag{1}$$

The intended interpretation is that there exist ρ^{in} and ρ^{out} as defined in Section 5.2 such that ϕ^{in} is the partition ρ^{in} induces on θ_C and ϕ^{out} is the partition ρ^{out} induces on θ_C.

We can assume that the vertices of the graph are assigned unique integer IDs, and that therefore there is a fixed total ordering of θ_C. Based on this total ordering, for any partition π of θ_C, let p be the number of parts of π, and define representatives(π) to be the p-vector $(v_1, v_2, \ldots, v_p)$ obtained as follows:

- v_1 is the smallest-ID vertex in θ_C,
- v_2 is the smallest-ID vertex in θ_C that is not in the same part as v_1,
- v_2 is the smallest-ID vertex in θ_C that is not in the same part as v_1 and is not in the same part as v_2,

and so on.

This induces a fixed total ordering of the parts of $\pi^{\text{in}} \vee \pi^{\text{out}}$.

We define a *weight configuration* of C to be a k-vector $\boldsymbol{w} = (w_1, \ldots, w_k)$ where each w_i is a nonnegative integer less than U. There are U^k such vectors.

We define a *weight/cost configuration* of C to be a k-vector together with a nonnegative integer s less than S. There are $U^k S$ such configurations.

We define a *configuration* of C to be a pair consisting of a topological configuration and a weight/cost configuration. The number of configurations of C is bounded by $c^w U^k S$.

The algorithms use dynamic programming to construct, for each cluster C, a table T_C indexed by configurations of C. In the case of optimization, the table entry $T_C[\Psi]$ corresponding to a configuration Ψ is *true* or *false*. For sampling, $T_C[\Psi]$ is a cardinality.

Let $\Psi = ((\pi^{\text{in}}, \pi^{\text{out}}), ((w_1, \ldots, w_k), s))$ be a configuration of C. Let count(Ψ) be the number of partitions ρ^{in} of the vertices incident to edges belonging to C with the following properties:

- ρ^{in} induces π^{in} on θ_C.
- Let $\pi = \pi^{\text{in}} \vee \phi^{\text{out}}$. Let representatives$(\pi) = (v_1, \ldots, v_p)$. Then for $j = 1, \ldots, p$, w_j is the total weight of vertices v for which C is a home cluster and such that v belongs to the same part of $\rho^{\text{in}} \vee \pi^{\text{out}}$ as v_j.

For optimization, $T_C[\Psi]$ is true if count(Ψ) is nonzero. For sampling, $T_C[\Psi] = \text{count}(\Psi)$. We describe in Section 5.5 how to populate these tables. Next we describe how they can be used to solve the problems.

5.4 Using the tables

For the root cluster $\hat{C}$, the cluster that contains all edges of G, $\theta_{\hat{C}}$ is empty. Therefore there is only one partition of $\theta_{\hat{C}}$, the trivial partition π_0 consisting of a single part, the empty set.

To detemine the optimum cost in the optimization problem, simply find the minimum nonnegative integer s such that, for some $\boldsymbol{w} = (w_1, \ldots, w_k)$ such that each w_i lies in $[L, U)$, the entry $T_{\hat{C}}[((\pi_0, \pi_0), (\boldsymbol{w}, s))]$ is *true.* To find the solution with this cost, the algorithm needs to find a “corresponding” configuration for each leaf cluster $C(\{uv\})$; that configuration tells the algorithm whether the two endpoints u and v are in the same district. This information is obtained by a recursive algorithm, which we presently describe.

Let C_0 be a cluster with child clusters C_1 and C_2. For $i = 0, 1, 2$, let $(\pi_i^{\text{in}}, \pi_i^{\text{out}})$ be a topological configuration for cluster C_i. Then we say these topological configurations are *consistent* if the following properties hold:

- For $i = 1, 2$, $\pi_i^{\text{out}} = \pi_0^{\text{out}} \vee \pi_{3-i}^{\text{in}}$.
- $\pi_0^{\text{in}} = \pi_1^{\text{in}} \vee \pi_2^{\text{in}}$.

For $i = 0, 1, 2$, let $(\boldsymbol{w}_i, s_i)$ be a weight/cost configuration for C_i. We say they are consistent if $\boldsymbol{w}_0 = \boldsymbol{w}_1 + \boldsymbol{w}_2$ and $s_0 = s_1 + s_2$.

Finally, for $i = 0, 1, 2$, let $\Psi_i = ((\pi_i^{\text{in}}, \pi_i^{\text{out}}), (\boldsymbol{w}_i, s_i))$ be a configuration for cluster C_i. Then we say Ψ_1, Ψ_2, Ψ_3 are consistent if the topological configurations are consistent and the weight/cost configurations are consistent.

▶ **Lemma 8.** *For a configuration Ψ_0 of C_0, count$(\Psi_0) = \sum_{\Psi_1, \Psi_2}$ count$(\Psi_1) \cdot$ count(Ψ_2) where the sum is over pairs (Ψ_1, Ψ_2) of configurations of C_1, C_2 such that Ψ_0, Ψ_1, Ψ_2 are consistent.*

The recursive algorithm, given a configuration Ψ for a cluster C such that $T_C[\Psi]$ is *true*, finds configurations for all the clusters that are descendants of C such that, for each nonleaf descendant and its children, the corresponding configurations are consistent; for each descendant cluster C', the configuration Ψ' selected for it must have the property that $T_{C'}[\Psi']$ is *true.*

The algorithm is straightforward:

Algorithm 1 $\textsc{Descend}(C_0, \Psi_0)$.

```
define $\textsc{Descend}(C_0, \Psi_0)$:
 $\textit{precondition: } T_{C_0}[\Psi_0] = \textit{true}$
assign $\Psi_0$ to $C_0$
 if $C_0$ is not a leaf config
   for each config $\Psi_1 = ((\pi_1^{\text{in}}, \pi_1^{\text{out}}), (\boldsymbol{w}_1, s_1))$ of $C_0$'s left child $C_1$,
     if $T_{C_1}[\Psi_1]$ is $\textit{true}$
       for each topological config $(\pi_2^{\text{in}}, \pi_2^{\text{out}})$ of $C_0$'s right child $C_2$
         let $(\boldsymbol{w}_2, s_2)$ be the weight/cost config of $C_2$ such that
           $\Psi_0, \Psi_1, \Psi_2$ are consistent
             where $\Psi_2 = ((\pi_2^{\text{in}}, \pi_2^{\text{out}}), (\boldsymbol{w}_2, s_2))$
         if $T_{C_2}[\Psi_2] = \textit{true}$
           call $\textsc{Descend}(C_1, \Psi_1)$ and $\textsc{Descend}(C_2, \Psi_2)$
           return, exiting out of loops
```

Lemma 8 shows via induction from root to leaves that the procedure will successfully find configurations for all clusters that are descendants of C_0. For the root cluster $\hat{C}$ and a configuration $\hat{\Psi}$ of $\hat{C}$ such that $T_{\hat{C}}[\hat{\Psi}]$ is *true*, consider the Ψ_C configurations found for each leaf cluster, and let $(\pi_C^{\text{in}}, \pi_C^{\text{out}})$ be the topological configuration of Ψ_C Consider the partition

$$\rho = \bigvee_C \pi_C^{\text{in}}$$

where the join is over all leaf clusters C. Because there are no vertices of degree one, for each leaf cluster $C(\{uv\})$, both u and v are boundary vertices, so ρ is a partition of all vertices of the input graph. Induction from leaves to root shows that this partition agrees with the weight/cost part $(\hat{\boldsymbol{w}}, \hat{s})$ of the configuration $\hat{\Psi}$. In particular, the weights of the parts of ρ correspond to the weights of $\hat{w}$, and the cost of the partition equals $\hat{s}$.

In the step of $\textsc{Descend}$ that selects $(\boldsymbol{w}_2, s_2)$, there is exactly one weight/cost config that is consistent (it can be obtained by permuting the elements of $\boldsymbol{w}_1$ and then subtracting from $\boldsymbol{w}_0$ and subtracting s_1 from s_0). By an appropriate choice of an indexing data structure to represent the tables, we can ensure that the running time of $\textsc{Descend}$ is within the running time stated in Theorem 2. For optimization, it remains to show how to populate the tables.

Algorithm 2 $\textsc{Descend}(C_0, \Psi_0, p)$.

```
define $\textsc{Descend}(C_0, \Psi_0, p)$:
 $\textit{precondition: } p \leq T_{C_0}[\Psi_0]$
 assign $\Psi_0$ to $C_0$
 if $C_0$ is not a leaf config
   for each config $\Psi_1 = ((\pi_1^{\text{in}}, \pi_1^{\text{out}}), (\boldsymbol{w}_1, s_1))$ of $C_0$'s left child $C_1$,
     for each topological config $(\pi_2^{\text{in}}, \pi_2^{\text{out}})$ of $C_0$'s right child $C_2$
       let $(\boldsymbol{w}_2, s_2)$ be the weight/cost config of $C_2$ such that
         $\Psi_0, \Psi_1, \Psi_2$ are consistent
           where $\Psi_2 = ((\pi_2^{\text{in}}, \pi_2^{\text{out}}), (\boldsymbol{w}_2, s_2))$
       $\Delta := T_{C_1}[\Psi_1] \cdot T_{C_2}[\Psi_2]$
       if $p \leq \Delta$
         $q := \lfloor p / T_{C_2}[\Psi_2] \rfloor$
         $r := r \bmod T_{C_2}[\Psi_2]$
         call $\textsc{Descend}(C_1, \Psi_1, q)$ and $\textsc{Descend}(C_2, \Psi_2, r)$
         return
       else $p := p - \Delta$ and continue
```

Induction shows that this procedure, applied to root cluster $\hat{C}$ and a configuration $\hat{\Psi}$ and an integer $p \leq T_{\hat{C}}[\hat{\Psi}]$, selects the p^{th} solution among those "compatible" with $\hat{\Psi}$. This can be used for random generation of solutions with given district populations and a given cost. Again, the running time for the procedure is within that stated in Theorem 2.

5.5 Populating the tables

For this section, let us focus on the tables needed for sampling. Populating the table for a leaf cluster is straightforward. Therefore, suppose C_0 is a cluster with children C_1 and C_2. We first observe that, given noncrossing partitions π_0^{out} of θ_{C_0}, π_1^{in} of θ_{C_1}, and π_2^{in} of θ_{C_2}, there are unique partitions $\pi_0^{\text{in}}, \pi_1^{\text{out}}, \pi_2^{\text{out}}$ such that the topological configurations $(\pi_0^{\text{in}}, \pi_0^{\text{out}}), (\pi_1^{\text{in}}, \pi_1^{\text{out}}), (\pi_2^{\text{in}}, \pi_2^{\text{out}})$ are consistent. (The formulas that show this are in the pseudocode below.)

The second observation: consider a configuration $\Psi_0 = (\kappa_0, (\boldsymbol{w}_0, s_0))$ of C_0. Then $\text{count}(\Psi_0)$ is

$$\sum_{\kappa_1,\kappa_2} \sum_{(\boldsymbol{w}_1,s_1),(\boldsymbol{w}_2,s_2)} \text{count}((\kappa_1, (\boldsymbol{w}_1, s_1))) \cdot \text{count}((\kappa_2, (\boldsymbol{w}_2, s_2))) \tag{2}$$

where the first sum is over pairs of topological configurations κ_1 for C_1 and and κ_2 where $\kappa_0, \kappa_1, \kappa_2$ are consistent, and the second sum is over pairs of weight/cost configurations that are consistent with $(\boldsymbol{w}_0, s_0)$. Note that because of how weight/cost configuration consistency is defined, the second sum mimics multivariate polynomial multiplication. We use these observations to define the procedure that populates the table for C_0 from the tables for C_1 and C_2.

Algorithm 3 COMBINE(C_0, C_1, C_2).

```
def COMBINE($C_0, C_1, C_2$):
 initialize each entry of $T_{C_0}$ to zero
 for each noncrossing partition $\pi_0^{\text{out}}$ of $\theta_{C_0}$
   for each noncrossing partition $\pi_1^{\text{in}}$ of $\theta_{C_1}$
     for each noncrossing partition $\pi_2^{\text{in}}$ of $\theta_{C_2}$
       $\pi_1^{\text{out}} = \pi_0^{\text{out}} \vee \pi_2^{\text{in}}$
       $\pi_2^{\text{out}} = \pi_0^{\text{out}} \vee \pi_1^{\text{in}}$
       $\pi_0^{\text{in}} = \pi_1^{\text{in}} \vee \pi_2^{\text{in}}$
       comment: now we populate entries of $T_{C_0}[\cdot]$ indexed by
                configurations of $C_0$ with
                topological configuration $(\pi_0^{\text{in}}, \pi_0^{\text{out}})$.
       for $i = 1, 2$,
         let $p_i(\boldsymbol{x}, y)$ be a polynomial over variables $x_1, \ldots, x_k, y$
           such that the coefficient of $x_1^{w_1} \cdots x_k^{w_k} y^s$
           is $T_{C_i}[((\pi_0^{\text{in}}, \pi_0^{\text{out}}), ((w_1, \ldots, w_k), s))]$
       let $p(\boldsymbol{x}, y)$ be the product of $p_1(\boldsymbol{x}, y)$ and $p_2(\boldsymbol{x}, y)$
       for every weight/cost configuration $((w_1, \ldots, w_k), s)$
         add to $T[((\pi_0^{\text{in}}, \pi_0^{\text{out}}), ((w_1, \ldots, w_k), s))]$ the
           coefficient of $x_1^{w_1} \cdots x_k^{w_k} y^s$ in $p(\boldsymbol{x}, y)$
```

The three loops involve at most c^w iterations, for some constant c. Multivariate polynomial multiplication can be done using multidimensional FFT. The time required is $O(N \log N)$, where $N = U^k S$. (This use of FFT to speed up an algorithm is by now a standard algorithmic technique.) It follows that the running time of the algorithm to populate the tables is as described in Theorem 2.

6 Implementation, and application to redistricting in France

Our implementation differs from the algorithm described in Section 5 in a few minor ways. Each configuration stores the populations of districts that intersect its boundary in a canonical order, as opposed to storing a k-vector containing the populations of all k districts. This reduces the number of configurations by reducing the redundancy of multiple configurations which are the same up to the ordering of the districts.

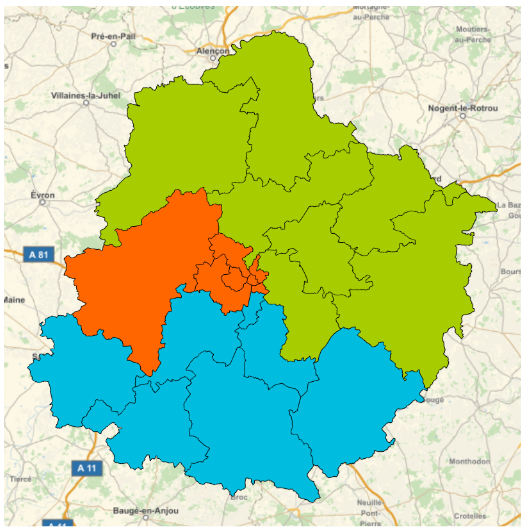
Also, our implementation does not use the FFT-based method for combining configurations; that method is helpful when the number of configurations is close to the maximum possible number but we expect that in practice the number will be substantially lower.

To demonstrate the effectiveness of our implementation, we applied it to the redistricting instances in France. There are about a hundred *departments* in France. The atoms are called *cantons*. For each department, one must find a partition of the cantons. Each part must be connected and each part's population can differ from the average by at most 20%. Omitting the special department of Paris (because its structure and rules are different) and the departments for which the target number of districts is one, we are left with eighty departments. The implementation was able to find solutions for every department. Additionally we were able to find solutions for over half of the departments with a tighter bound of 5%.

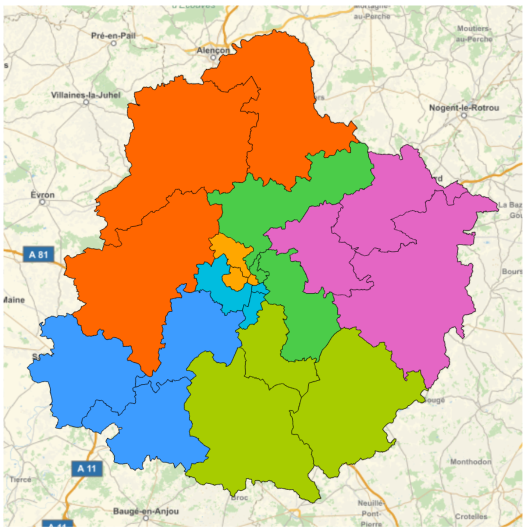
We were able to compute these solutions for all departments on a single machine within eight hours. As shown in Figure 5 the cut edge size of the optimal solution increases only slightly as the population constraint increases. This data suggests there is little downside to creating departments with closer populations when such a solution exists.

6.1 Example: *Sarthe*

Consider for example the department *Sarthe*. We specify that the minimum population of a district is 150,000 and the maximum population is 200,000. The computation took about 30 seconds on a single core of a 2018 MacBook Pro (Figure 4a).



(a) A districting of the cantons of Sarthe, France, generated with four districts.



(b) Sarthe with seven districts.

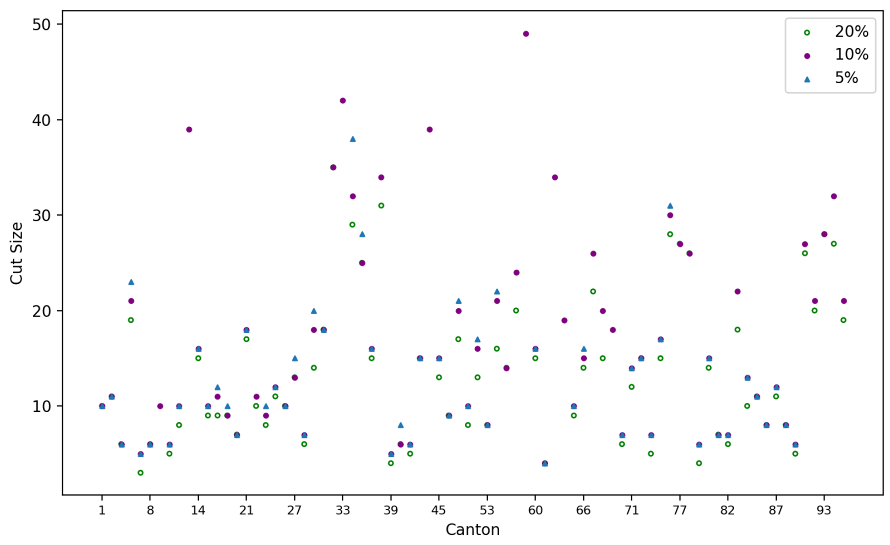


Figure 5 Differences in cut size cost for different population constraints. We include only those instances for which our implementation finds a solution.

References

1 Sachet Bangia, Christy Vaughn Graves, Gregory Herschlag, Han Sung Kang, Justin Luo, Jonathan C. Mattingly, and Robert Ravier. Redistricting: Drawing the line, 2017. `arXiv:1704.03360`.

2 Hans L. Bodlaender, Daniel Lokshtanov, and Eelko Penninkx. Planar capacitated dominating set is $W[1]$-hard. In Jianer Chen and Fedor V. Fomin, editors, *Proceedings of the 4th International Workshop on Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 50–60. Springer, 2009. `doi:10.1007/978-3-642-11269-0_4`.

3 Daniel Carter, Gregory Herschlag, Zach Hunter, and Jonathan Mattingly. A merge-split proposal for reversible Monte Carlo Markov Chain sampling of redistricting plans, 2019. `arXiv:1911.01503`.

4 J. Chen. Expert report of Jowei Chen, Ph.D., Raleigh Wake Citizen's Association et al. vs. the Wake County Board of Elections, 2017. URL: `https://www.pubintlaw.org/wp-content/uploads/2017/06/Expert-Report-Jowei-Chen.pdf`.

5 Vincent Cohen-Addad, Philip N. Klein, and Neal E. Young. Balanced centroidal power diagrams for redistricting. In *Proceedings of the 26th ACM International Conference on Advances in Geographic Information Systems*, pages 389–396, 2018. `doi:10.1145/3274895.3274979`.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015.

7 Daryl DeFord and Moon Duchin. Redistricting reform in Virginia: Districting criteria in context. *Virginia Policy Review*, 12(2):120–146, 2019.

8 Daryl DeFord, Moon Duchin, and Justin Solomon. Recombination: A family of Markov chains for redistricting, 2019. `arXiv:1911.05725`.

9 Michael Dom, Daniel Lokshtanov, Saket Saurabh, and Yngve Villanger. Capacitated domination and covering: A parameterized perspective. In *Proceedings of the 3rd International WorkshopParameterized and Exact Computation*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2008. `doi:10.1007/978-3-540-79723-4_9`.

10 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010. doi:10.1007/s00453-009-9296-1.

11 Moon Duchin. Geography meets geometry in redistricting. Conference at Center for Geographic Analysis at Harvard University, May 2019. URL: https://cga-download.hmdc.harvard.edu/publish_web/CGA_Conferences/2019_Redistricting/slides/Moon_Duchin.pdf.

12 Moon Duchin and Bridget Eileen Tenner. Discrete geometry for electoral geography, 2018. arXiv:1808.05860.

13 Martin E. Dyer and Alan M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discret. Appl. Math.*, 10(2):139–153, 1985. doi:10.1016/0166-218X(85)90008-3.

14 David Eppstein, Michael T. Goodrich, Doruk Korkmaz, and Nil Mamano. Defining equitable geographic districts in road networks via stable matching. In *Proceedings of the 25th ACM International Conference on Advances in Geographic Information Systems*, pages 52:1–52:4, 2017. doi:10.1145/3139958.3140015.

15 Andreas Emil Feldmann and Dániel Marx. The parameterized hardness of the k-center problem in transportation networks. *Algorithmica*, 82(7):1989–2005, 2020. doi:10.1007/s00453-020-00683-w.

16 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011.

17 Kyle Fox, Philip N. Klein, and Shay Mozes. A polynomial-time bicriteria approximation scheme for planar bisection. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing*, pages 841–850, 2015. doi:10.1145/2746539.2746564.

18 R. S. Garfinkel and G. L. Nemhauser. Optimal political districting by implicit enumeration techniques. *Management Science*, 16(8):B–495, 1970.

19 Naveen Garg, Huzur Saran, and Vijay V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. *SIAM J. Comput.*, 29(1):159–179, 1999. doi:10.1137/S0097539794271692.

20 Sushmita Gupta, Saket Saurabh, and Meirav Zehavi. On treewidth and stable marriage. *CoRR*, abs/1707.05404, 2017. arXiv:1707.05404.

21 Gregory Z. Gutin, Mark Jones, and Magnus Wahlström. The mixed Chinese postman problem parameterized by pathwidth and treedepth. *SIAM J. Discret. Math.*, 30(4):2177–2205, 2016. doi:10.1137/15M1034337.

22 Robert E Helbig, Patrick K Orr, and Robert R Roediger. Political redistricting by computer. *Communications of the ACM*, 15(8):735–741, 1972.

23 Gregory Herschlag, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier, and Jonathan C. Mattingly. Quantifying gerrymandering in North Carolina, 2018. arXiv:1801.03783.

24 Gregory Herschlag, Robert Ravier, and Jonathan C. Mattingly. Evaluating partisan gerrymandering in Wisconsin, 2017. arXiv:1709.01596.

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.

26 Takehiro Ito, Kazuya Goto, Xiao Zhou, and Takao Nishizeki. Partitioning a multi-weighted graph to connected subgraphs of almost uniform size. *IEICE Trans. Inf. Syst.*, 90-D(2):449–456, 2007. doi:10.1093/ietisy/e90-d.2.449.

27 Takehiro Ito, Xiao Zhou, and Takao Nishizeki. Partitioning a graph of bounded tree-width to connected subgraphs of almost uniform size. *J. Discrete Algorithms*, 4(1):142–154, 2006. doi:10.1016/j.jda.2005.01.005.

28 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.

29 Jun Kawahara, Takashi Horiyama, Keisuke Hotta, and Shin-ichi Minato. Generating all patterns of graph partitions within a disparity bound. In *International Workshop on Algorithms and Computation*, pages 119–131. Springer, 2017.

30 Philip N. Klein and Shay Mozes. Optimization Algorithms for Planar Graphs. `http://planarity.org/`. accessed June 2018.

31 Dániel Marx, Ario Salmasi, and Anastasios Sidiropoulos. Constant-factorapproximations for asymmetric TSP on nearly-embeddable graphs. In *Proceedings of the 19th Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 60 of *LIPIcs*, pages 16:1–16:54, 2016. `doi:10.4230/LIPIcs.APPROX-RANDOM.2016.16`.

32 Anuj Mehrotra, Ellis L. Johnson, and George L. Nemhauser. An optimization based heuristic for political districting. *Management Science*, 44(8):1100–1114, 1998.

33 Lorenzo Najt, Daryl R. DeFord, and Justin Solomon. Complexity and geometry of sampling connected graph partitions. *CoRR*, abs/1908.08881, 2019. `arXiv:1908.08881`.

34 J. K. Park and C. A. Phillips. Finding minimum-quotient cuts in planar graphs. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 766–775, 1993. `doi:10.1145/167088.167284`.

35 W. Pegden. Pennsylvania's congressional districting is an outlier: Expert report, League of Women Voters vs. Pennsylvania General Assembly, 2017. URL: `https://www.brennancenter.org/sites/default/files/legal-work/LWV_v_PA_Expert_Report_WesleyPegden_11.17.17.pdf`.

36 Richard H Pildes, Tacy F Flint, and Sidley Austin. Brief of political geography scholars as amici curiae in support of appellees. URL: `https://www.brennancenter.org/sites/default/files/legal-work/Gill_AmicusBrief_%20Political%20Geography%20Scholars_InSupportofAppellees.pdf`.

37 Satish Rao. Finding near optimal separators in planar graphs. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pages 225–237, 1987. `doi:10.1109/SFCS.1987.26`.

38 Satish Rao. Faster algorithms for finding small edge cuts in planar graphs. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 229–240, 1992. `doi:10.1145/129712.129735`.