Transport Distances between Districting Plans

This coming week will be a little crazy, since Justin will be helping with VRDI in parallel with teaching a week-long course at MIT. This document is to outline a potential "distances between plans" project, to make sure you all know what to do!

1 Motivation

As we've seen in VRDI, we are now capable of generating *tons* of districting plans! These ensembles are extremely useful for statistics and generating a neutral baseline. But, we don't have a way to visualize the "span" of an ensemble. That is, we can measure histograms of individual values (e.g., cut edges, mean-median) to understand an ensemble, but we lack a direct way to visualize how diverse the plans are.

In this project, we'll propose and experiment with a notion of *distance* between two districting plans. That is, given two plans, we'll generate a value that is zero when they're the same, small when they're similar, and large when they're different. These distances are valuable for a number of reasons:

- When we run MCMC, distances can help us understand things like mixing. If our MCMC algorithm generates lots of plans that are close to one another in our distance metric, it probably didn't mix!
- Distances provide an interesting way to visualize and digest an ensemble of plans. Embedding methods like MDS, ISOMAP, LLE, and t-SNE take a matrix of distances and attempt to embed it into Euclidean space $\mathbb{R}^n$, like the plane $\mathbb{R}^2$. This gives us a way to visualize an ensemble directly rather than just derived statistics.
- On the more ambitious side, distances may allow us to *simplify* an ensemble. For example, we could attempt to extract a small subset of plans from an ensemble that is somehow "representative," using techniques like farthest-point sampling, ε-nets, or coresets.

2 What You'll Need

We're going to compute our distances using *convex optimization* tools. These are extremely powerful pieces of software that are worth knowing if you're going to do research in machine learning or operations!

Justin's favorite tool for convex modeling is `cvx`, a Matlab library. He can likely give you an MIT account if you prefer Matlab and want to give this tool a try. Otherwise, a slightly-less-elegant version of `cvx` is `cvxpy`, which you can use in Python. Either one of these tools should sufficen, and you're encouraged to get started by installing and playing with these tools.

3 The Project

I've divided this project into a few parts documented below. These can be parallelized! That is, they're largely independent of one another. Please contact Justin for help understanding any of this—there are likely many typos, since Justin wrote this while watching THE AVENGERS on an airplane.

A bit of notation as a baseline:

- We'll take $G = (V, E)$ to be the "dual graph" on which we build districts. So, V (the set of vertices) is likely the set of precincts or census blocks, and E (the set of edges) connects two units if they share a boundary.[1]
- We're going to represent individual districts as vectors in $\mathbb{R}_+^{|V|}$ that sum to 1; that is, each district contains one nonnegative number per unit in V that sums to 1. There are two options that Justin sees as reasonable here:
  1. We could make the "indicator function" of a district. In particular, if a district is some subset of vertices $D \subseteq V$, then we can take our district vector to have elements
  $$x(v) := \begin{cases} 1/|V| & \text{if } v \in D \\ 0 & \text{otherwise.} \end{cases}$$
  2. Alternatively, we could assign nodes proportionally to population:
  $$x(v) \propto \begin{cases} \text{population}(v) & \text{if } v \in D \\ 0 & \text{otherwise.} \end{cases}$$
  Here $\propto$ denotes "proportional to," because we want d to sum to 1.

  The basic assumption that x sums to 1 is a reflection of the population balance constraint. For now, we'll assume they're exactly population-balanced, because it will make the math below easier.
- We'll denote the set of district vectors defined above as $\text{Prob}(V)$, the set of probability distributions over V.

3.1 Distances Between Districts

In this part of the project, we'll engineer a distance on $\text{Prob}(V)$. Distances are functions $d : \text{Prob}(V) \times \text{Prob}(V) \to \mathbb{R}_+$ that satisfy a few properties:

- Positivity: $d(x, y) \geq 0$ for all $x, y \in \text{Prob}(V)$
- Identity: $d(x, y) = 0$ if and only if $x = y$
- Triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, z \in \text{Prob}(V)$

[1] Something to think about later: Do we want to give the edges lengths?

VRDI task: There are really simple distances we could put on $\mathrm{Prob}(V)$, e.g. the Euclidean distance $d(x, y) = \|x - y\|_2$. Make sure you understand why these distances are not too meaningful. As an extreme example, what happens when districts don't overlap?

In this project, we'll experiment with the *1-Wasserstein distance* between districting plans, from the theory of optimal transport. Take $d(v, w)$ to be the shortest-path distance between $v, w \in V$ along G. Then, the 1-Wasserstein distance between $x, y \in \mathrm{Prob}(V)$ is given by

$$W_1(x, y) := \begin{cases} \min_{T \in \mathbb{R}^{|V| \times |V|}} & \sum_{v,w \in V} T_{vw} d(v, w) \\ \text{subject to} & T_{vw} \geq 0 \ \forall v, w \in V \\ & \sum_w T_{vw} = x(v) \ \forall v \in V \\ & \sum_v T_{vw} = y(w) \ \forall w \in V. \end{cases} \tag{1}$$

Make sure to review our optimal transport breakout session to remember how to interpret this optimization problem.

VRDI task: Write some `cvx` or `cvxpy` code to solve (1). You'll need to compute $d(v, w)$ for all v, w before you can solve this; use the Floyd–Warshall algorithm.

The problem (1) is fairly slow to solve: The number of variables scales like $|V|^2$! Luckily, there's a formulation that scales like $|E|$, which is much smaller for our graphs. We'll take each edge $e \in E$ and *orient* it arbitrarily $e = v \to w$. That is, we think of e as having a direction, which we can choose arbitrarily (it won't matter!). Then, we can compute W_1 as:

$$W_1(x, y) := \begin{cases} \min_{J \in \mathbb{R}^{|E|}} & \sum_{e \in E} |J_e| \\ \text{subject to} & P^\top J = y - x. \end{cases} \tag{2}$$

Here, $P \in \{-1, 0, 1\}^{|E| \times |V|}$ is the signed "node-incidence" matrix:

$$P_{ev} := \begin{cases} -1 & \text{if } e = v \to w \text{ for some } w \\ 1 & \text{if } e = w \to v \text{ for some } w \\ 0 & \text{otherwise.} \end{cases}$$

It is *absolutely critical* to store P as a sparse matrix!

VRDI task: Now implement (2) and make sure it agrees with the implementation of (1). It should be much faster! Try it on actual districts.

3.2 Lifting to Distances Between Plans

Suppose a state is divided into k districts. Then, we can represent a districting plan as a set of vectors $(x_1, \ldots, x_k) \in \mathrm{Prob}(V)^k$, one x_i for each district. Our next task is to show that we can take any distance on $\mathrm{Prob}(V)$ and *lift* it to a distance in $\mathrm{Prob}(V)^k$.

Suppose we have two districting plans $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$. Assuming we have a distance W_1 from §3.1 (or a simpler distance $\|x - y\|_2$ for now will suffice, if teams are working on these in parallel), we can compute a *cost matrix* $C \in \mathbb{R}_+^{k \times k}$ as:

$$C_{ij} := W_1(x_i, y_j).$$

A reasonable notion of distances between districting plans attempts to *match* the set of districts in one plan to the set of districts in another, and then sum up the individual transport distances. We can write this mathematically as follows:

$$\begin{array}{ll} \min_{M \in \{0,1\}^{k \times k}} & \sum_{ij} M_{ij} C_{ij} \\ \text{subject to} & M \text{ is a } k \times k \text{ permutation matrix.} \end{array} \tag{3}$$

The problem as-written seems hard to solve! There are $k!$ possible permutations to try.

Secretly, however, (3) is a *linear assignment problem*. Recall that permutation matrices have a few simple properties: They are nonnegative, their rows sum to 1, and their columns sum to 1. Matrices satisfying these three constraints are called *doubly stochastic*. Permutation matrices are also binary, but if we ignore this constraint we get a *convex relaxation* of (3):

$$\begin{array}{ll} \min_{M \in \mathbb{R}^{k \times k}} & \sum_{ij} M_{ij} C_{ij} \\ \text{subject to} & M\mathbb{1} = \mathbb{1} \\ & M^\top \mathbb{1} = \mathbb{1} \\ & M \geq 0. \end{array} \tag{4}$$

Here, $\mathbb{1}$ denotes the vector of all 1's, and the first two constraints are fancy notation for showing that the rows and columns of M sum to 1.

Here's something magic, if you haven't seen it before:

> **VRDI task:** This one's an optional math challenge, or you can find the answer online (it's a homework exercise for a linear programming course!). Argue that solutions to (4) are generically solutions to (3), even though (4) has fewer constraints. There are many ways to prove this—one simple one is to use *total unimodularity*.

Given the previous VRDI task, you know what's next:

> **VRDI task:** Code up (4) in `cvx` or `cvxpy`.

Congratulations! If you combine your code with the code from §3.1 you now have a true distance between districting plans. Or, at least, I'm 99.8% sure you do. But to be confident we need to solve one more math exercise:

> **VRDI task:** Think of the cost matrix C as a function $C(x_1, \ldots x_k, y_1, \ldots, y_k)$ of two plans, and denote the output of (4) as $A(C(\cdot))$. Prove that if $W_1(\cdot, \cdot)$ satisfies the triangle inequality on $\text{Prob}(V)$, then $A(C(x_1, \ldots x_k, y_1, \ldots, y_k))$ satisfies the triangle inequality on $\text{Prob}(V)^k$.

3.3 Visualizing Metric Spaces

The previous two sections outline a way to compute distances between pairs of districting plans. So, if we have an ensemble of p plans, we now can compute a *pairwise distance matrix* $D \in \mathbb{R}_+^{p \times p}$, where element D_{ij} is the distance between plan i and plan j.

There are really interesting things you can do with the matrix D, which is sometimes known as a *metric matrix*. In particular, you can create plots where each districting plan is a point, and the distances between the points in the plots approximates the distances in D. This is useful to visualize and navigate the whole ensemble of plans at once.

VRDI task: Experiment with MDS, SMACOF, t-SNE, and/or other methods for embedding D into the plane. Visualize an ensemble of ~ 1000 plans. When are plans close in the embedding? When are they far? If you replace W_1 with Euclidean distance, how does the embedding change?

As an example where this visualization might be useful, try the following experiment:

VRDI task: Use your code to compute an embedding of an ensemble generated using `Recom`. Now, run a `Flip` ensemble, and plot it in the same set of axes. Hopefully it'll take up a small amount of space in the overall plot!

3.4 Selecting a Representative Set of Plans

If we generate ensembles of millions of plans, it's likely (maybe!) the case that many of the plans look similar. Another application of a distance metric is to *subsample* the ensemble, e.g. to generate a "representative" subset that is fairly diverse. This is useful in a number of contexts, such as helping summarize the main ways to redistrict a state.

VRDI task: From a pairwise distance matrix D, try the *farthest-point sampling* algorithm to generate a representative subset of plans.

A drawback of the task above is that it requires computing and storing D, which is a huge matrix. The MCMC context is particularly interesting for this representative sampling task, because we can generate arbitrarily many plans. Maybe there are smarter ways to compute representative samples using our distance function in this context. For example, maybe during the Markov chain we keep a pool of interesting plans that gets updated when we see something new that's sufficiently far away.

VRDI task: Combine our new distance with a Markov chain. Try to develop a tool that can discover an interesting set of plans that are far apart from one another.