GerryChain Practice Day 1: Arithmetic and Updaters

Daryl DeFord

June 24, 2019

1 Introduction

The main purpose of today's session is to get some experience working with the `Partition` class in GerryChain and learn how `updaters` are used to evaluate functions on districting plans. This is also a good opportunity to practice using python for performing arithmetic operations and more complicated functions. As we saw last week, many of the partisan metrics that are currently in use are simple algebraic operations applied to the seats and votes percentages.

One of the overall goals of these sessions is preparing you for the Your State project. Hopefully some of the functions you design in these sessions will make it in to your presentation ☺

1.1 Python Code

The initial code for these investigations is in a separate repo in the VRDI organization. You should be able to do everything in a Jupyter Notebook or whichever other IDE you used on Friday for the original templates. The more detailed guide is a good reference for more examples.

2 Functions on Partitions

The `Partition` class is the fundamental object of GerryChain. It contains information about both the underlying dual graph and the assignment of nodes to districts as well as additional information in the form of updaters. Our first set of tasks is to be able to measure properties of the districting plan, from the corresponding `Partition` object. Many of the components of GerryChain expect functions that take as input a `Partition` and output a value, so this is a natural place to start our investigation of GerryChain.

To begin with we will start with a version of the PA example we looked at on Friday. For PA we have 8 different starting plans, so we can several different starting partitions to compare. The python file provided here builds partitions for each of these plans as well as several recursive tree starting plans for you to compare. The function the outputs the number of cut edges in each plan as an example of how you can compare and display the values.

Here are some initial metrics that you could try to implement and evaluate on the various starting plans:

- The number of nodes and edges in the original graph
- The number of adjacent districts that could be merged with ReCom
- The number of nodes and edges in each district subgraph
- The number of boundary nodes (endpoints of cut edges) in the districting plan
- The number of boundary nodes in each district subgraph
- The maximum population imbalance between districts
- The average population distance from ideal across the entire plan
- The maximum and minimum population precincts in each district

2.1 Election Data

The script also adds three elections worth of data to the graph: Presidential 2012, Presidential 2016, and a turnout weighted average of the three Senate elections 2010, 2012, and 2016. You can access the percentages, votes, or winner for each party in each district using the same syntax as any other updater:

```
>>> partition["PRES12"].percents("Democratic")
```

for example. More details about `Elections` are here. Using the elections we can try to compute some new features of the districts. How do the various plans compare under the partisan metrics? Several of them like Mean–median and Efficiency gap already exist as built in functions. After comparing those, you can try to create some of your own.

- The number of voters for each party in each district
- The turnout ratio[1] for each district
- The "gaps" between the vote percentages
- The most polarized district in the plan or precinct in each district
- The Partisan Bias Score
- The Duke Gerrymandering Index
- The number of wasted votes per district
- Efficiency Gap:
  - Write a function for the $S-2V+\frac{1}{2}$ equal turnout version and compare to the wasted votes version. How does your turnout ratio computation explain the differences?

2.2 Design your own functions

As we pointed out several times last week, gerrymandering research is still very much in the "wild west" stage of development. There is still lots of room to try and create new measures and interesting ways of evaluating MCMC-created districting plans. If you have some time left, brainstorm in your group some new metrics to apply to the plans and try to implement them. This is a great chance to simply try things out!

3 Updaters

The final input to form a `Partition` is a dictionary of updaters. This is a dictionary that maps names to functions, where each function takes as input the current `Partition`. At every step of the Markov chain, these udpaters are computed and the values are stored on the `Partition` corresponding to the current state of the chain. The values of the updaters can be accessed as follows:

```
>>> partition["name_of_updater"]
```

We have already seen examples of this with the elections, which are a special kind of updater, as well as population and cut edges. One helpful feature of updaters is that they allow us to pass additional arguments to the functions we are evaluating. For example, in order to define some of the graph functions above it was probably helpful to have access to the list of cut edges.

3.1 Evaluating Ensembles

Add some of your functions from the previous section as updaters to the other script in the Day 1 repo. Now we are interested in how the value changes over the course of a single Markov chain, instead of comparing different plans. The example shows one way to do this.

[1] Well, at least (# voters)/population.