GerryChain Practice Day 2: Conditionals and Constraints

Daryl DeFord

June 25, 2019

1 Introduction

The main purpose of today's session is to get some more experience working with the `Partition` class in GerryChain and the `constraints` that are used to define the state space of permissible partitions. This is also a good opportunity to practice using conditional expressions in python. This is a crucial step in building a good Markov chain sampler, since there are vastly more "bad" partitions than nice ones and keeping the bad ones out of our ensemble saves a lot of wasted computational effort.

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 Validators

Like the `updaters` we studied yesterday, the constraints in GerryChain are defined as functions that take `Partition` objects as inputs. However, the output of a constraint function is either the value `True` or `False`, depending on whether or not the `Partition` satisfies that particular constraint. This allows us to define a partition to be a valid element of the state space if it satisfies all of the constraints that we have added to our `Validator`. At each step of the chain, each proposed plan is evaluated by the full collection of constraint functions - if it fails any of them a new proposal is generated until we find one that lies in the state space. This is necessary because of the size of our state space and the potential number of proposals at each step. For more details see Section 8.3 of this.

We will use the same PA setup as yesterday, first testing our constraint functions on the partitions associated to the various starting plans and then trying them out in the ensemble setting. Here are some examples to try out - which of the initial plans pass these tests?:

- Implement a population bound that uses the number of nodes instead of the populations of the districts.
- Implement the population bound that measures the difference between the maximum and minimum population values instead of the difference from ideal.
- Build a constraint that rejects a plan if there is a district with BPOP% over 60%
- Pick several edges and make an updater that accepts a partition if none of those edges are cut.
- Define a community of interest as a small subset of the nodes. Make an updater that only accepts plans that place all of those nodes in the same district.
- Make a constraint that doesn't increase the number of county splits compared to the 8th grade plan.

- For each initial partition, can you design a constraint that only that partition passes while the rest fail?

3 Bounds

The main constraint documentation is here and the current methods fall into two main categories - contiguity checks and numerical bounds. The contiguity checks are self-explanatory while the value checks require a little more setup. The main versions use the `UpperBound` and `LowerBound` functions that take a function and a number as an argument. The number returned by the function is compared to the value and the constraint returns True if it is on the appropriate side of the bound. For example, the following function returns the average of the node ids in the first district and the corresponding bound returns True if that average is less than the average node id value across the entire graph.

```
>>> def silly_constraint(partition):
>>>     return np.mean([n for n in partition.parts[1]])
>>> silly_bound = constraints.UpperBound(silly_constraint, np.mean([n for n in graph.nodes]))
```

Here are some ensemble experiments to try:

- Vary the population and cut edges constraints that are in the template. How small can you make the bounds before the rejection rate significantly slows down the Markov chain? Using the flip proposal and the number of flips updater from yesterday how do the runs change as you tighten and loosen the constraint.
- Implement a constraint that rejects plans that have a large gap in the election percents. Use the maximum gap in one of the gerrymandered plans to pick the bound.
- Create a constraint that will accept a plan that violates the population constraint if it is very compact.
- Implement a constraint that only accepts a partition if the majority of the cut edges are between precincts won by different parties.
- Implement a constraint that only accepts partitions that have the same number of possible ReCom merges as the initial plan
- Use one of your updaters from yesterday to write a constraint that relies on that updater
- Write a constraint that depends on the number of steps that the chain has take so far
- Write a constraint that definitely disconnects the metagraph for the flip proposal.

4 Your State(s)

For compactness constraints[1], what makes sense in one state[2] may not make sense for another. We have highlighted this with respect to the geographic compactness measures and things like coastline and shapefile resolution but it is also true for discrete measures. Grab several of the .json graphs from the your state repo for your state. They are sorted by FIPS Code in the repo.

- How do the reasonable values for the constraints vary with the sizes of the geographic units?
- Is it possible to use the same constraints on a County level run as a block group level run?
- What do you expect to happen if you enforce the same constraints on all of your runs, regardless of the geographic units?

[1] and lots of other measures.
[2] or one type of geographic unit, or one type of legislative district