RedistrictR Writeup

Joe Izenman, Nikki Lee, and Nicholas Chen

December 11, 2017

In many places throughout the United States, voting districts at levels from municipal government up through congressional districts are drawn by the individuals and parties in power at the time when districts are scheduled to be redrawn. Because of the personal stake that these decision makers have in the outcome of the redistricting process, it is unrealistic to expect these individuals to draw fair districts. Often times, these individuals select districts that maximize their own chances for reelection and cannot be considered fair, equitable, or supportive of the democratic system by any rigorous measure.

The aim of RedistrictR is to generate more equitable voting districts based on a transparent and rigorous methodology seeking to strengthen the democratic process in an environment where many voters are losing confidence in our electoral system. RedistrictR takes a novel approach to redistricting by deploying an evolutionary algorithm that selects potential district maps based on their performance to several measurable metrics of equity: vote efficiency gap, compactness, and demographic cluster proximity.

RedistrictR focuses on redistricting in the context of the districts of the San Diego County Board of Supervisors which has been criticized in the past for its homogeneity and self-serving redistricting tactics[1]. Tackling the redistricting issue in San Diego County is an exercise to illustrate that our quantitative approach to redistricting is feasible and can result in measurable improvements over the status quo. We use demographic data from the U.S. Census and historical voting data archived by U.C. Berkeley to divide the county into five districts; one for each of the five seats on the Board of Supervisors.

The Data and Motivation for Using an Evolutionary Algorithm

In order to build Board of Supervisor districts from the ground up, we compiled a dataset of small geographic areas comprising San Diego County. These small geographic areas will then be grouped together like puzzle pieces to form the five districts represented by each of the County Supervisors.

The small building blocks that formed the basis of our analysis are called census block groups. These areas represent the smallest units for which census data are publicly available. We compiled data on these building blocks from the 2010 census and American Community Survey on population, racial demographics, income levels, and age distribution in the block groups.

In San Diego County, there are 1,792 census block groups forming the basis of our redistricting analysis. Our task of dividing these building blocks into 5 districts is one with a vast number of potential solutions. In fact, there are nearly $3\text{x}10^{1250}$ possible ways to divide 1,792 objects (census block groups) into 5 distinct groups (districts)[2] - this is almost 15 times the number of atoms estimated to make up the known universe! Of course, many of these potential district maps fail to pass muster, because of legal constraints about equal population and contiguous districts, but the scale of potential and legal district maps is still immense. In fact, because actual

[1] E.g. http://www.sandiegouniontribune.com/sdut-region-prominent-critic-rips-county-redistricting-2011jun16-story.html

[2] The number of ways to divide n objects into k non-empty subsets is known as a Stirling number of the second kind.

redistricting is performed at the block rather than block group level, a full solution would involve closer to 40,000 objects, rendering the space even more intractable.

To further complicate the problem of the immense number of potential district maps to choose from, the error surface of the space is highly irregular and non-differentiable, with many local optima, making it difficult or impossible to systematically identify the best district maps.

Evolutionary Algorithm Background

The expansive solution space, and the highly irregular error surface of our problem, led us to select a class of algorithms called evolutionary algorithms to approach our problem. Prior work in the redistricting space at the University of Illinois at Urbana-Champaign[3] lays out a Parallel Evolutionary Algorithm for Redistricting (PEAR), which we leveraged heavily in our implementation. Evolutionary algorithms are well suited in several key ways to the redistricting problem. First, they require no prior knowledge of or assumptions about the error surface of the problem they are tasked to solve. Second, they are effective at traversing large solution spaces, focusing efforts on promising solutions while not being tied to them. Third, they are highly parallelizable and can be deployed at extremely large scale if resources are available.

Evolutionary algorithms are inspired by biological evolution. They operate on the premise of evolving a population of candidate solutions through a number of generations towards a population deemed to be more fit according to a set of user defined measurement criteria. In the case of evolutionary algorithms, several of the processes that govern evolution in the natural world are defined by the author of the algorithm. Specifically, these elemental processes are mutation, crossover (mating), and selection.

At a high level, mutation is the process by which a potential district map is randomly changed by shifting some number of block groups between districts within a map. Crossover is the process by which two potential district maps are combined to create a new district map that combines the boundaries of its ‘parent’ district maps. Finally, selection chooses individuals from the population to be passed through to the next generation, according to their score on a fitness metric that evaluates each individual in the population according to its performance across a user defined set of criteria - in our case vote efficiency gap, compactness, and cluster proximity.

The Fairness Metrics

The most novel aspect of our approach to quantitative redistricting is our incorporation of multiple evaluation criteria in our evolutionary algorithm. Prior research on quantitative redistricting has largely dealt with the evaluation of a single metric, either vote efficiency gap or district compactness. Our system is able to simultaneously evaluate and optimize on both of these, plus a novel metric of cluster proximity.

Vote efficiency gap is a standard measure of political fairness that has been the focus of much evaluation of voting districts. It attempts to measure the discrepancy in wasted votes cast by members of two parties. Wasted votes are votes that don’t contribute to the victory of the winning party. In cases where gerrymandering has occurred, vote efficiency gap will show an imbalance of wasted votes being cast by the party not in power.

[3] http://www.sciencedirect.com/science/article/pii/S2210650216300220

Compactness is an evaluation of the geometric shape of a district. Regularly shaped districts perform well whereas irregular districts that are often the signature of partisan redistricting perform poorly. Several different measures of compactness exist, typically treating a perfect circle as the optimal shape. Some are geared toward penalizing dispersion, or the overall elongated spread of a district, while others primarily penalize indentation, the amount of irregularity and contortion in the edge, a symptom of drawing around individual neighborhoods, block by block.

For our solution we use the Polsby-Popper compactness score, the ratio of a district's area to the area of a circle with the same perimeter, which is an indentation-focused metric. To speed up processing, we implement PEAR's recommended method for calculating district perimeter—store a matrix of pairwise edges between all individual units, and after district assignment, iterate through this matrix, adding the shared edge of any pair of units with different assignments to their respective district perimeters.

Finally, we measured solutions according to a cluster proximity score that is the most innovative part of our evaluation methodology. Our cluster proximity score measures how consistently a district is composed of building blocks that are similar to one another as measured by their racial, age, and income composition. Similar building blocks according to these measures are identified using k-means clustering. Similar block groups are often located near one another geographically so placing these similar building blocks in the same districts is feasible even when also selecting for regularly shaped districts. This methodology may be easily expanded to include more and different demographic parameters, per the researcher's needs.

In addition, work has been started on an optimization measure for political unit integrity, penalizing a solution for breaking up counties, cities, and census tracts, with a greater penalty for smaller units divided. Redistricting professionals frequently attempt to keep these predefined communities whole when possible, and the lack of ability to optimize for this features is a pain point cited in our domain expert interviews. For simplicity of our initial product, this measure is not yet included.

In the context of the evolutionary algorithm that we deployed, each potential solution is evaluated according to each of these criteria. We developed scoring functions that can look at a district assignment and compute a single numerical score for each metric. The evolutionary algorithm can then take these scores and select districts that perform the best according to the metric or combination of metrics on which it is instructed to optimize.

The Implementation

The implementation of our redistricting evolutionary algorithm was done through a python package called DEAP (Distributed Evolutionary Algorithms in Python) that provides the framework for implementing evolutionary algorithms. From the documentation, 'DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent.'[4] We found this claim to be true and the implementation of our algorithm to be relatively straightforward.

[4] https://deap.readthedocs.io/en/master/

We wrote initialization, mutation, crossover, and selection functions customized for our redistricting problem, starting from a foundation of the PEAR recommendations. We then supply these functions to DEAP and use the framework provided by the package with some basic conditional logic to deploy our functions, generating an initial population and running through a set number of generations, evolving along the way towards a population of candidate solutions that is more fit on average than the randomly generated initial population. As the individuals mutate and mate as the algorithm runs, we also incorporate a series of checks to enforce certain constraints on the individuals in the population such as that districts must have similar populations to one another and that no district may be split up into more than one piece. The mutation and mating operators are deployed probabilistically throughout the process.

Since the ultimate goal of our project is to record the top performing district maps according to our evaluation functions, we also utilized DEAP's hall-of-fame functionality. The DEAP hall-of-fame is exactly what it sounds like. This functionality keeps track of the top n individuals encountered by the algorithm even as the algorithm may iterate through hundreds, thousands, or even tens of thousands individuals. At the point the algorithm completes its iteration through the given number of generations, our program writes all of the maps stored in the hall-of-fame to an Amazon RDS database. In subsequent runs of the algorithm, instead of initializing the population completely at random, we instead pull the top solutions that our algorithm has encountered in the past down from the database to, in part, pick up where the previous run left off.

To improve the speed of the algorithm, we also parallelized the most computationally intensive parts of the code. We selected the Python package scoop because DEAP is specifically written to interface seamlessly with this parallelization framework. To parallelize our algorithm, we use maps[5] in any portion of our code that traverses the population, to distribute these independent tasks to different cores or nodes on a cluster. We parallelized three parts of our algorithm: (1) the initialization of the population, (2) the mutation operation, and (3) the mating operation. On a macbook pro with a 2.5 GHz Intel Core i7 processor with 8 cores, the parallelization we implemented decreased the average time to process a single generation of 100 potential maps more than ten times from more than 300 seconds per generation to less than 30 seconds per generation.

Supporting Technology

Because of the nature of the problem and the immense number of potential solutions, we also explored other technologies available to further distribute beyond the 4 - 8 cores available on our laptops. Amazon EC2 offers large instance types with up to 96 cores, however, as the number of cores scales, the memory of the instance scales similarly. For our algorithm which does not require a significant amount of memory overhead, the cost to utilize the largest Amazon EC2 instances was excessive. Unfortunately Amazon EC2 does not offer any lower priced instance types that have relatively more cores than memory which would be better suited to our computational requirements[6].

[5] We used the scoop.futures.map_as_completed as our map function. The base python map function does not actually parallelize the computation whereas the scoop.futures.map_as_completed does support true parallelization.

[6] Google cloud, however does offer 'high cpu' type instances that do offer relatively more cores than memory that would have been better suited to the running of our algorithm.

As an alternative solution to paying for a large EC2 instance with far more memory than we required, we identified a tool called starcluster[7] that streamlines the process of constructing a cluster of Amazon EC2 instances. This tool would allow us to scale out to a large number of cores, without paying an even higher premium for enormous amounts of memory that our algorithm wouldn't utilize. This strategy of building a cluster of small EC2 instances offers significant cost savings compared to simply running our algorithm on an individual EC2 instance with a comparable number of cores. The table below shows these estimated savings.

			$ Per Week		
Cores	Memory	Comparable EC2 Instance	Starcluster Built of t2.small Instances	Dedicated EC2 Instance	Starcluster Cost Savings per Week
2	4 GB	t2.medium	$7.73	$7.80	$0.07
4	16 GB	t2.xlarge	$15.46	$31.18	$15.72
8	32 GB	t2.2xlarge	$30.91	$62.36	$31.45
48	192 GB	m5.12xlarge	$185.47	$387.07	$201.60
96	384 GB	m5.24xlarge	$370.94	$774.14	$403.20

Conclusion

As noted on numerous occasions above, the redistricting problem that we tackled has an absolutely immense solution space which our computational resources were severely inadequate to sufficiently traverse.[8] As such, our project should be viewed as an illustration of a concept that, given sufficient computing resources, could identify districts that would be in all likelihood closest to the best possible map as measured by user defined measures.

[7] http://star.mit.edu/cluster/docs/latest/index.html

[8] Prior research on computational redistricting such as that completed at the University of Illinois at Urbana-Champaign has deployed true high powered computing clusters with hundreds of thousands of cores to tackle redistricting problems; http://www.sciencedirect.com/science/article/pii/S2210650216300220

Further, we chose to measure our districts on three measures: vote efficiency gap, compactness, and cluster proximity. We make no representations or claims that these are the only three metrics of fairness that matter in the redistricting problem. Instead, we choose these metrics as a starting point to illustrate that it is possible to rigorously and objectively identify better voting districts according to numerous evaluation criteria.

These caveats aside, we identified numerous maps in San Diego County that, according to our measures of fairness, improve upon the status quo of the current County Board of Supervisor districts. The scores on each of our three evaluation criteria of maps that we generated form relatively normal distributions. In the case of all three metrics, the current County Board of Supervisor districts underperform the majority of the districts we generated, falling in the xx, yy, zz percentiles of the distributions of vote efficiency gap, compactness, and cluster proximity scores of the maps generated by our algorithm.