NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

Final Report

Date: November 26th, 2024
Team Name: FairyMander
Sponsor: Bridget Bero
Professor: Isaac Shaffer
Mentor: Vahid Nikoonejad Fard

Team Members:
Izaac Molina (Team Lead)
Dylan Franco
Jeysen Angous
Sophia Ingram
Ceanna Jarrett

NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

Table of Contents

1 Introduction

Congressional district lines are redrawn following the release of the decennial census, a survey that is taken every ten years. This census provides information on population, demographics, economics, geography, etc. The number of seats each state gets in the US House of Representatives is determined by the state's population, thus determining the state's number of congressional districts. Congressional districts are the voting districts for state representatives, which are members of Congress. These districts must comply with state and federal laws, and represent the population. Every two years, people within each district vote for their representative in Congress. This is an important decision since Congress holds all legislative power; it is the only part of the government that can make and change laws.

Currently, thirty-nine state's redistricting processes are controlled by their state's legislative body. In these states, district lines are manipulated and approved by their members of Congress. The first draft of the proposed district lines is drawn by a legislative committee. Members of the legislative committee are chosen by the legislature. Some of their responsibilities include: monitoring government operations, recommending courses of action to the Senate, etc. The draft of the proposed districts may be vetoed by the governor, but this veto can be overridden by a ⅔ legislative vote. Final approval of the proposed districts varies by state, but it is most commonly approved by legislature with a ⅔ vote. However, the legislature holding all the power to create new districts causes a problem to arise.

Gerrymandering is the act of drawing district lines in a way that benefits certain political parties; this leads to underrepresentation and political manipulation. Representatives have the power to manipulate electoral outcomes during redistricting, thus guaranteeing their electoral success. This power poses a threat to our democracy, having the potential to make voting obsolete. There are active solutions in place for gerrymandering, but they are not as widespread as they should be. One solution against gerrymandering is for voting districts to be drawn by the Independent Redistricting Commission (IRC). The IRC is separate from the legislature, meaning that politicians do not have the opportunity to manipulate voting districts. This makes the redistricting process fair and transparent. The IRC must adhere to certain criteria when creating districts, such as equal population, protecting racial minorities, and making districts compact. However, only eight state's districts are currently being drawn using this system. This means the other 42 states' voters are being gerrymandered and suppressed.

In search of a more widespread solution, our client, Dr. Bero, reached out to us to create an open-source, non-biased redistricting algorithm. Dr. Bero, of NAU's Department of Civil Engineering, Construction Management, and Environmental Engineering, has the goal of creating a user-friendly tool to educate citizens on how congressional districts can be created fairly through the use of our algorithm. To achieve this goal FairyMander will be composed of two primary components. The first is a Python package for generating "FairyMandered" district maps, and the second is a website with an interactive US map to display these districts, along with supplemental information about redistricting laws and the metrics used to calculate district fairness.

An essential goal of this project is to generate state district maps and evaluate their fairness. To do so, we first acquire the data needed for redistricting calculations, such as state geography, population, and voting outcomes from the US Census and Redistricting Data Hub (RDH). The Census provides data on geography, populations, and racial distributions, and the RDH provides data on voting outcomes. We used Python to develop our redistricting utilities along with a few essential libraries. GeoPandas, a library for processing geographic data, is used to process our Census and RDH shapefiles, which are geographic data files, and output this data into a visual map. Folium is used to make our GeoPandas maps interactable so users can examine the generated districts; these maps are also embedded into our FairyMander website. We use PySal, a common library for analyzing geospatial data, to cluster geographic bodies based on population; this is what creates our districts. Such tools and libraries will be massive time savers both in providing already established utility and in using pre-existing algorithms to build our work on. Finally, we have a set of fairness metrics that we use to evaluate the districts for fairness, which is defined in a Python module. These metrics are derived from previous redistricting research and district evaluation protocols.

2 Process Overview

For the design and development of FairyMander, we divided the team with the focus on specific roles assigned. One member of the team worked on the front-end portion of designing the website to display the team's work while the rest dedicated solely to the redistricting algorithm. With the redistricting module complete, the team referred to the Excel sheet which listed all fifty states and the progress that was made for each state. For version control, the team used Github to track and update their changes to guarantee others were working on the latest version of FairyMander. For more in-depth information on team standards, you may refer to the Team Standards document.

3 Requirements

The team's requirements were curated from the extensive research on gerrymandering and fairness by closely examining the current literature on computation redistricting. Below is a more thorough assessment of both functional and non-functional requirements.

3.1 Functional Requirements

These requirements are tasks that our application will be capable of performing.

3.1.1 Creating Districts

The algorithm the team used to generate the districts for all states uses geographic, electoral, and demographic data as input. Following these inputs, the algorithm will perform a series of steps to create a set of potential district plans. Upon developing the district plans, the generated map(s) will be further evaluated using district fairness metrics, which measure compactness, political competitiveness, and minority representation.

3.1.2 Visualizing Results

Once the district maps have passed the series of steps needed, they will then be displayed in a web application featuring a map of the United States where users can see and interact with the "FairyMandered" maps. The district maps will be visualized using folium in a responsive and interactive map format that will promote engagement with our results. Along with the ability to view the interactive maps, users will also be able to view demographic distribution for each district in the form of a pie chart as well as fairness metrics scores for both current congressional districts and "FairyMandered" districts. Additionally, a glossary page will be available if a user chooses to read a more in-depth description of a fairness metric score or view the formula.

3.1.4 Redistricting Education

The web application encompasses an educational aspect by informing users about the redistricting process and covering topics on redistricting, gerrymandering, and how gerrymandering is done. In addition to the redistricting process users are also informed about our generated maps, promoting civic engagement with both our results and the redistricting process as a whole.

3.2 Non-Functional Requirements

These requirements are tasks on how our system must perform.

3.2.1 Compatibility

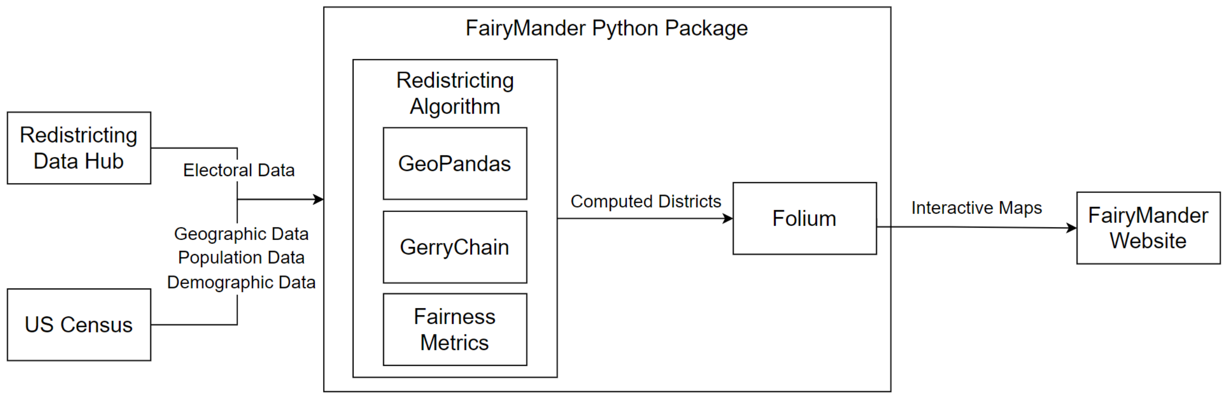
The web application is compatible with various web browsers and is responsive across various devices and platforms. Additionally, the web application, along with the interactive map, loads quickly and functions smoothly when a user hovers over or clicks on a state.

4 Architecture and Implementation

We now describe the architecture of our system and how we developed a solution that fulfills the requirements in section 3

4.1 High Level Architecture

Our software will take in data from the US Census and the Redistricting Data Hub, use this data to execute our redistricting algorithm and then serve our results on a website. The general Architecture for this process is outlined below:



4.1.1 Data Component

The Data component of the project consists of downloaded files from the U.S. Census and Redistricting Data Hub (RDH). For this project's needs, a one-time download for this data will suffice, as there is no need to pull real-time data using the Census and/or RDH APIs. This data will then be "cleaned" by removing unnecessary table entries and unifying the Census and RDH data into an aggregated data file for each state. Future work should likely be focused on automating this data pulling, which we will touch more on in section 7. The data cleaned in this component will be fed into the python package, enabling district map generation. The district generation algorithm pulls geographic, population, and electoral information for the state to be redistricted from the data component. To enable comparison between the generated district maps and the current district maps, developers will also be able to pull the current district data from this component.

4.1.1 Python Package

From here, the Python Package component will take in the data to compute and evaluate state district maps. This forms the core of our project, as it wraps our redistricting algorithm in an easy to use interface. Our redistricting algorithm will utilize GerryChain, a computational redistricting library developed at Tufts University, to produce maps based on GeoPandas dataframes obtained from the Data component. Developers will be able to tweak different parameters in the algorithm to achieve a desired redistricting outcome, which will be elaborated on later in section 4.2. Once the districts are generated, they can then be evaluated using the "Fairness Metrics" module in the package, which contains a series of functions that are used to compute different fairness values for a given map. This module also contains utility for displaying a comprehensive analysis of a map by applying all of the fairness metric calculations, as well as performing a comparison between two maps' fairness metrics. Finally, the generated districts can be converted into interactive maps using Folium, which is what we will use to present our redistricting results on the FairyMander website.

4.1.3 Website

With the maps generated, our team examined the maps and analyzed their various pros and cons using best redistricting practices. This process reflected the real life process that redistricting officials would go through when using computer generated districts in order to consider the nuances between each state's redistricting needs. Once a map was chosen, upload the map into the FairyMander website, where it will be integrated into the interactive map. From there, any accessory information about the state's redistricting laws and our justification for the redistricting of said district will also be input manually by the team. This website then acts as the primary way in which users will interact with FairyMander, showcasing the project's results and educating users on redistricting practices.

4.1.4 Architectural Style

Due to the unique needs of our project, we needed to adopt an architecture that would allow a clear and defined data flow for obtaining redistricting data, creating districts from that data, then presenting the created districts. As such, our software most closely aligns with the 'Pipe-Filter' architecture style. This architectural style focuses on transforming data by sending it through defined 'pipes' and processing it at defined 'filters', leading to a desired end result. In our project, each component is a 'filter', which will take in data through 'pipes' as the redistricting data is transformed into interactable district maps. This approach allows us to keep each 'filter' highly modular, decreasing coupling between the different components and subcomponents of the project. It also allows flexibility as at any point in the pipeline, a new 'filter' could be added to further enhance the generated districts. However, it is worth noting that a key next step for FairyMander would be to further develop the 'pipe' parts of our project. The current scope does not define modules for automating the transfer of data between certain components, such as the consolidation of data between the RDH and Census, as well as the upload of maps directly from Folium to our website. Such 'pipes' could greatly enhance the efficiency at which FairyMander could operate, and would be an appropriate next step for this project, to be discussed further in section 7.

In the remaining subsections, we further break down each component to detail its structure and role in developing our solution

4.2 Data Component

4.2.1 Obtaining the Data

We use three types of data to gather the information we need. First is voter registration data, which is available for each state and organized by census block. Second is census block data collected from the census itself, which lists the census blocks that make up each county in every state. We then connect these data sets, ensuring there are 1:1 connections between each entry in both sets. To prepare the data, we process it in two stages. In the first stage, we combine the census block data. This helps us organize the voter registration data by location. In the second stage, we merge this combined data with the voter registration data, resulting in a file for each county in each state, sorted by census block and containing all relevant voter data.

To achieve this, we wrote several programs in C and Python. The first program was a Python web scraper that efficiently gathered the necessary census block data from www2.census.gov using a recursive algorithm to download only the required files. Next, a C program combined these two datasets (voter registration and block data) for each county and state. Finally, another C program merged this output with voter registration data gathered manually from redistrictingdatahub.org. The resulting files are now ready for integration with redistricting shapefiles. The scraper used can be found in fairymander/census_data_scraper.py.

4.2.2 Data Component Access

As mentioned, we obtained the cleaned dataset for each state by pulling from the U.S. census and redistricting data hub. The cleaned data currently lives in our git repository, under Data/finalData, which holds the data used for initializing our redistricting algorithm, and Data/CurrentCongressionalDistricts, which has the data for the current state congressional district maps. When wanting to make a comparison between generated maps and the current map, the function get_curr_district_file in data.py can be used to pull from Data/CurrentCongressionalDistricts, the result of which can then be fed into the comparison utilities present in our fairness module. The DistrictGenerator class pulls from the data component when generating districts, using GeoPandas' default read_file utility.

4.3 Python Package

4.3.1 Definitions

In this section, we will be using a number of terms and metrics related to redistricting. For reference, here is a table containing definitions for these terms:

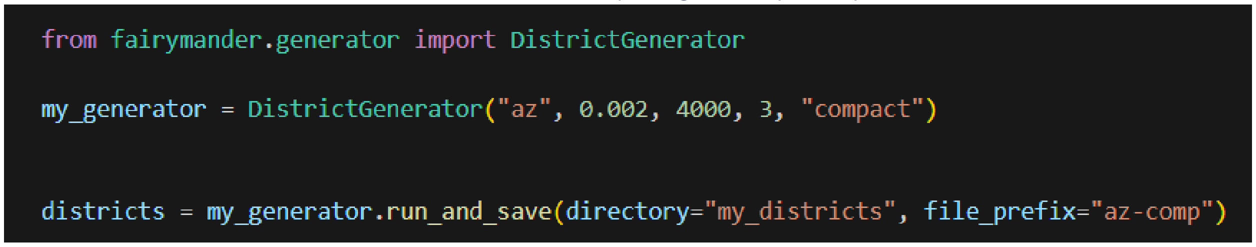
Name	**Definition**

Polsby-Popper Score	Metric for district compactness. The ratio of the district's area to the area of a circle whose circumference is equal to the district perimeter.
Reock Score	Metric for district compactness. The ratio of a district's area to the area of a minimum bounding circle that the district can fit in.
Efficiency Gap	Metric for district partisanship. Determined by calculating the ratio of the difference between the dem/rep votes and the total votes.
Lopsided Margin Test	District competitiveness measurement that calculates the difference between the average percentages by which each party won in a district.
Mean Median Difference	Examines how the distribution of district vote-shares affects potential seat outcomes, rather than solely focusing on the nominal seat outcome. Calculates the difference between the median and mean vote-share across all districts for one party.
Dissimilarity Index	Measure of minority representation within a district. Each index value indicates how the minority population is distributed across the districts in the state. This metric is measured on a scale from 0 to 1 where values closer to 1 indicate higher segregation of minority populations between districts.

4.3.2 DistrictGenerator

DistrictGenerator acts as the main access point for generating districts, located in fairymander/generator.py. When a developer wants to generate a set of districts, they will create a DistrictGenerator object, initializing it with the state to redistrict, the acceptable population deviation between district, the number of "steps" to run the algorithm, a number of district maps to generate, and whether they would like to optimize for "compact" (Polsby-Popper score) or "competitiveness" (Efficiency Gap). They then call 'run()' to execute the generator, returning a list of district maps, with the option to save the maps as .shp files to the local filesystem.

An example of this process is shown below:

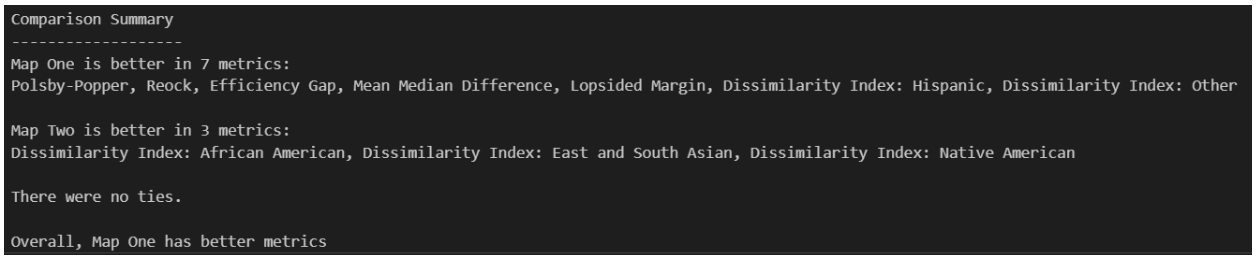


```
from fairymander.generator import DistrictGenerator

my_generator = DistrictGenerator("az", 0.002, 4000, 3, "compact")


districts = my_generator.run_and_save(directory="my_districts", file_prefix="az-comp")
```

This external interface allows developers to easily generate districts. The internal “guts” of this component uses a modified version of Markov Chain Monte Carlo to produce new district plans, which is implemented in GerryChain. Explaining the intricacies of Markov Chain Monte Carlo and its use in computational redistricting is quite lengthy, so we recommend using the provided documentation in GerryChain’s github repository, found at https://github.com/mggg/GerryChain, to learn more about this process.

4.3.3 Fairness Metrics

To effectively evaluate generated district maps, built out utilities that can compute district fairness metrics given a GeoPandas dataframe containing a district map. Such utilities are included in fairymander/fairness.py. This file contains utilities for each fairness metric defined in 4.3.1, but the primary way these utilities are meant to be interacted with is via the full_analysis and compare_maps utilities. The first utility, full_analysis, optionally displays the district map along with the values for each fairness score. comapre_maps then extends this utility, essentially performing full_analysis on two maps the developer wants to compare. It also extends this analysis by providing a summary of which map scored better in each metric. See the below example:



```
Comparison Summary
-------------------
Map One is better in 7 metrics:
Polsby-Popper, Reock, Efficiency Gap, Mean Median Difference, Lopsided Margin, Dissimilarity Index: Hispanic, Dissimilarity Index: Other

Map Two is better in 3 metrics:
Dissimilarity Index: African American, Dissimilarity Index: East and South Asian, Dissimilarity Index: Native American

There were no ties.

Overall, Map One has better metrics
```

This utility is used to compare generated maps, both between themselves and the current district plan, fulfilling a critical role in ensuring our generated maps are high quality.

4.3.4 Folium Converter

Finally, generated district plans can be converted into an interactive map using folium. The utility for this is found in fairymander/folium_converter.py. It is quite simple, providing the map_to_folium function for converting the district plan to a folium map, with a separate function map_to_folium_from_file doing the same but for a .shp file. These maps will be the final output we serve on our website.

4.4 Website Component

4.4.1 Landing Page

The landing page of the FairyMander website consists of a header, body, and footer. The header includes navigation tabs like “Home,” “About,” “Contact,” and “More” for quick and simple access to various sections on the website. The body includes an interactive United States map in which users can hover over any state and see its name, population, and the number of districts/seats in that state as of the 2020 census. To the right of the map, a hamburger dropdown menu is displayed with all the states. Moreover, under the interactive map, readable information about the redistricting process is displayed with topics such as “What is Redistricting?”, “What is Gerrymandering?”, and lastly “How is Gerrymandering Done?”. Also accessible on the body of the landing page is more information about our website, project, and acknowledgments. Lastly, the footer contains copyright notice.

4.4.2 Interactive Map/UI

Each state, upon clicking on from the United States interactive map or hamburger dropdown menu, will have a side-by-side comparison of the current congressional districts and our “FairyMandered” districts with respect to states with one district and Hawaii. Each district on both maps will be displayed in different colors, once hovered, the district number will be displayed along with the population in that district. Moreover, demographic distributions in a pie chart form will be displayed for each district once clicking on the dropdown menu. Fairness metrics are also displayed for both current districts and “FairyMandered” districts, with the better score of the two underlined. A glossary page is also available, which gives a more in-depth definition of each metric and formula should the user click on the “Learn More About Fairness Metrics” button. Lastly, each state will display information on the redistricting laws after the 2020 census, an option to explore other states as they all will be listed in alphabetical order, or provide feedback about their experience using the FairyMander website.

4.5 Architecture and Implementation Conclusion

We wrap up this section by discussing how the final FairyMander implementation differed from our initial design for the product.

Firstly, the data we chose to operate on was based on the census block group level as opposed to the census tract level. We were initially concerned with the block group level being too slow, as it is more fine grained than the tract level. However, after we discussed this concern with Peter Rock, the lead developer for GerryChain, he recommended that block group data be used as in his experience it led to much higher quality maps without sacrificing too much computational time. Sure enough, when we organized our data into block groups instead of tracts, the maps came out higher quality and at not much slower of a rate compared to census tracts. Ultimately, this change allowed us to produce better results from our generated maps.

The second major change was deciding to switch to GerryChain instead of SKATER, the library we initially went with for helping us implement our algorithm. SKATER was more outdated than we had originally thought, as many of the algorithms it uses are more general purpose for multiple geographic analyses. GerryChain, on the other hand, is specifically designed to assist in computational redistricting, making it much more suitable for our use case. The team was not aware of GerryChain in our initial feasibility analysis, indicating that our feasibility research could have certainly been more thorough in the initial stages of the project. Nevertheless, GerryChain proved immensely helpful for implementing our algorithm in a clean and easy to understand way.

Finally, we performed major overhauls to the Fairness Metric module in the python package. Firstly, we decoupled it from the DistrictGenerator package, as we initially planned to implement fairness metrics directly in our algorithm, but instead we opted to use the built-in utility of GerryChain, enabling us to decouple the generator and metric modules. Secondly, we decided to not use a "composite score" system, as we realized that doing so would require assigning particular weights to different metrics, therefore biasing our analysis toward what we personally felt was more "fair". We instead decided to just display the results of each metric plainly, letting the user decide which metrics they value more. Overall, this resulted in a more simple and less biased module for evaluating district fairness.

5 Testing

Upon completing our implementation, the team created and enacted a testing plan to verify our project's accuracy and effectiveness. For the python package, this included a suite of integration and unit tests, and for the website, this involved a series of user acceptance tests from our peers to verify the website's usability.

5.1 Unit and Integration Testing

Our testing suite for the python package contains 47 unit and 8 integration tests, using pytest as our testing framework. The unit tests are focused on ensuring the accuracy of our fairness module, our utilities for interacting with the data component, and the steps within the district generation process. The integration tests, then, are mainly focused on district generation, ensuring that the system as a whole functions as expected when each individual unit is working together. We concluded from these tests that our district generation and fairness utilities properly meet the functional requirements for these modules outlined in section 3. The test suite can be found in fairymander/tests

5.2 Usability Testing

We also conducted usability testing for the website portion of our project, ensuring that the website effectively communicates our results, along with being responsive and easy to use. These tests were conducted via user acceptance tests, where test users were asked to complete a series of tasks on the site, reporting any issues or comments they had while doing so. The primary insight that arose from these tests was ensuring that it was intuitive for the user to understand the fairness metrics and what they mean in our results. To address this, we implemented hover icons that give a description of the metric, as well as a link to the breakdown for each fairness metric within our state results pages, making it easy for users to learn about the metrics. Overall, these tests ensured that our product meets the key nonfunctional requirements we outlined in section 3, resulting in a clean and responsive website.

6 Project Timeline

Below is the timeline that the team followed for this semester. The project was divided into different parts to ensure an organized workflow. At the beginning of the semester, there was a focus on researching fairness requirements and designing the algorithm. The team divided the tasks as follows: Sophia and Izaac were responsible for researching and testing algorithms to determine the best one. As a team, we agreed to use GerryChain. Dylan handled the task of cleaning census data for all 50 states, Ceanna focused on researching fairness metrics, and Jeysen worked on the website.

Around October, the team started developing the different modules. Sophia and Izaac continued their work on the GerryChain algorithm, while Ceanna and Izaac collaborated on the fairness module. Jeysen worked on the Folium module. Izaac later created the FairyMander package, which integrated all the modules into a single tool for developers. The FairyMander package was a significant milestone for the team.

Using the package, Sophia, Ceanna, and Dylan were able to efficiently generate maps for each state and run them through the fairness metric to identify the fairest possible maps. Jeyson transferred all the processed data to the website, while Izaac and Jeysen took charge of testing both the package and the website.

Now, in December, the team has successfully completed everything required for the project.

FairyMander

	September				October				November				December			
Preliminary Research On Fair Redistricting	x	x	x	x												
Algorithm Design	x	x	x	x	x	x	x									
Algorithm Design Case Study					x	x	x									
Pull data for Algorithm		x	x	x												
Initial Algorithm Implementation				x	x	x										
Iterate on Initial Algorithm Implementation								x	x	x	x					
Create Fairness Utility Module					x	x										
Create Folium Module						x	x									
Website	x	x	x	x	x	x	x	x	x	x	x	x				
50 state web pages								x	x	x	x	x				
Testing								x	x	x	x					
State redistricting law								x	x	x						
State definition of why districts fair								x	x	x						

7 Future Work

While our team is satisfied with the work we have completed and this project has met all the requirements, we now identify some key areas of the project that could be improved in the future.

7.1 Hawaii

One of the most challenging aspects of designing our algorithm was resolving issues concerning islands. A fundamental rule for all districts is that they should not be discontinuous, i.e., each district should be one solid area with no breaks. This is difficult to consider algorithmically, however, when islands exist, as they necessitate some form of allowed "breaking". For most states, such as California and Massachusetts, this is not too difficult, as it simply requires logically connecting each island to the nearest piece of mainland. However, this is much more difficult in the specific case of Hawaii. Since Hawaii is composed of so many islands, thousands of connections would need to be made between them, leading to a whole host of political decisions that would have to be decided in order to properly decide which islands should be connected to each other. In an effort to avoid this political bias, our team opted to leave Hawaii out of our analysis. With that said, future effort could be taken to research the most appropriate connections to make through analyzing the relationships between islands in Hawaii and consulting political officials to make sure the connections being made are sensible.

7.2 State Specifics

Our goal with creating our algorithm was to make an algorithm that can be run on all 50 United States (with the exception of Hawaii). As such, we decided to omit state specific redistricting laws when designing our algorithm for the sake of making it broadly applicable. For instance, some states require that counties are not split up in the redistricting plan. These state specifics are not considered in the algorithm, but future refinements could be made to the project to enable these state laws within the algorithm.

7.3 Dynamic Data Upload

To obtain the data, we primarily downloaded and cleaned the data manually. This is fine for the current district data, but if the data were to be updated, then the same manual download and cleaning would need to be done for the new datasets. In an effort to mediate this, more dynamic systems for obtaining the data necessary for this project could be developed, such as dynamic data download from the U.S. Census using the Census API.

8 Conclusion

In this project, our primary motivation was to address the issue of gerrymandering, a pervasive problem that undermines democratic representation. Gerrymandering leads to politically manipulated voting districts, which can disenfranchise voters and skew legislative outcomes. Our client, Dr. Bridget Bero, sought an open-source, unbiased solution to educate citizens on fair congressional districting and provide tools to counteract gerrymandering. Through FairyMander, we developed an innovative system comprising a Python package for generating fair district maps and an interactive website to visualize and educate about these districts.

Our solution offers several standout features not available in other open-source projects:

- **Algorithmic District Generation:** Utilizing GerryChain, our project can generate any number of high-quality, fairness-driven voting districts.
- **Interactive Visualizations:** Embedded Folium-based interactive maps allow for detailed analysis and user engagement.
- **Fairness Evaluation Metrics:** With the implemented comprehensive suite of fairness metrics, users are able to monitor for and analyze any potential biases.

- **Additional Educational Material:** The user-friendly companion website explains local redistricting laws, fairness metrics, and the importance of combating gerrymandering.

9 Glossary

Gerrymander: The act of manipulating district lines to control political outcomes. This means that you are not actually choosing the member of Congress that you want representing you, they are being chosen for you. Gerrymandering harms our democracy and could lead to laws being passed that do not represent you, or your community.

Packing: The action of packing like-minded voters to as few districts as possible.

Cracking: The action of splitting like-minded voters with similar characteristics across different districts

Reock Score: A measurement of compactness for each district that ranges from 0 to 1. It is the ratio between the area of a district and the area of the minimum bounding circle around the district. To determine the compactness of the district, the score needs to be closer to 1 for the district to be considered compact.

Polsby Popper: Another measurement of compactness. It is the ratio between a district's area and the square of its perimeter. The closer the score is to 1, the more compact the district is.

Efficiency Gap: Measurement of competitiveness, where the wasted votes in an election (the votes that did not contribute to the election outcome for a specific party) quantify the degree of partisanship. The metric can be either positive or negative. A negative value indicates a Democratic advantage, while a positive value indicates a Republican advantage in the outcomes.

Lopsided Margin Test: District competitiveness measurement that calculates the difference between the average percentages by which each party won in a district. The formula which is represented as $LMT = WA - WB$ where WA is the average percentage by which party A won in a district and WB is the average percentage by which party B won in a district.

Mean Median Difference: Examines how the distribution of district vote-shares affects potential seat outcomes, rather than solely focusing on the nominal seat outcome. Calculates the difference between the median and mean vote-share across all districts for one party.

Dissimilarity Index: Measure of minority representation within a district. Each index value indicates how the minority population is distributed across the districts in the state. This metric is measured on a scale from 0 to 1 where values closer to 1 indicate higher segregation of minority populations between districts.

10 Appendix A: Development Environment and Toolchain

10.1 Hardware

When working with the FairyMander package, any standard computer—whether a PC or a Mac—will suffice. However, additional memory may be required when cleaning large datasets, as this process can be resource-intensive.

10.2 Toolchain

To set up the environment for the FairyMander package, you will need Python version 3.11 or higher. Python is a dynamically-typed programming language that supports both structured and object-oriented programming. It is widely used and serves as the foundational language for this project.

Another essential tool is Jupyter Notebook, an open-source application with a flexible interface that allows users to configure and arrange workflows efficiently. Jupyter Notebooks are particularly useful for testing all 50 states' data.

We recommend using Visual Studio Code (VSC) as the integrated development environment. VSC is a versatile tool that helps developers edit, build, and test programs effectively. Working with both Visual Studio Code and Jupyter Notebook provides a seamless development experience.

The Geopandas library in Python is crucial for handling geometric operations and working with shapefiles, which are essential for the algorithm to run successfully. Shapefiles contain the geographic data needed for generating and analyzing maps.

The primary algorithm used in the project is GerryChain, which addresses the problem of political redistricting using the Markov Chain Monte Carlo (MCMC) method. GerryChain is responsible for creating the various districting maps.

Pytest is a python testing software, to make sure that the code that is created is tested well.

Additionally, the Folium library in Python is used to visualize shapefiles as interactive Leaflet maps. This library enables the conversion of shapefiles into web-friendly maps, which are displayed on the project's website.

10.3 Setup

One is to set up the environment to use the FairyMander packet. Install Visual Studio Code, and install python 3.11 or above. Create a virtual environment, press ctrl-shift-p, select the python: create environment. Make sure to hit the check mark. Then you will create the environment. You must activate the environment, use the command for mac and linux: source .venv/Scripts/activate. Or if one is working in Windows command prompt use the command call .venv/Scripts/activate. Move into the main Fairy Mander folder and build the FairyMander package using the line "pip install -e ." . Once you have that all installed, create a jupyter notebook and you are ready to generate maps and run the fairness metrics.

10.4 Production Cycle

Now that you are ready to generate maps and calculate their fairness, let's begin by setting up for Arizona. With the environment configured and a Jupyter notebook ready, you can generate the maps. In the first code cell, import the required function from the generator module and initialize the generator with parameters such as state, deviation, steps, number of maps, and type. The two fairness metrics being tested are Efficiency Gap and Polsby-Popper. The districts will then be processed through the my_generator.run function, which saves the maps into a file system for further testing. At the end of the process, the maps are dissolved. Here is the example code:

```
from fairymander.generator import DistrictGenerator
my_generator = DistrictGenerator("az", 0.004, 100, 3, "competitiveness")
districts = my_generator.run_and_save(directory="my_districts", file_prefix="ct-comp")
districts = my_generator.run() # note, now returns the maps dissolved
```

Next, to compare the generated maps to the current district, use the shapefile of the existing districts. Import the necessary files as shown below:

```
from fairymander.data import get_curr_district_file
gdf = get_curr_district_file('az')
```

To compare the three FairyMander-generated districts to the current district, use the following code to generate a full report:

```
from fairymander.fairness import full_analysis, compare_maps
compare_maps(gdf , districts[0])
```

Once you identify a map that is fairer than the current district, convert the shapefile into an interactive map using Folium. Here's the setup for that:

```
from fairymander.folium_converter import map_to_folium
res = map_to_folium('az', districts[0])
```

This process enables you to generate maps, evaluate their fairness, compare them to current districts, and convert them for visualization. With these steps, you can utilize the full functionality of the FairyMander package to create and analyze fair district maps.