An Artificial Intelligence Approach to Redistricting

Rohan Ramchand

October 2017

Abstract

Gerrymandering, or the use of the redistricting pen to effect a partisan advantage, has long hampered the ability of the American people to freely choose their Congressional representation. Instead, gerrymandering has created a system where, rather than voters choosing their representatives, representatives can choose their voters. In this paper, we discuss the history and practice of gerrymandering in the United States and propose a novel method of drawing nonpartisan legislative district lines. Our method allows for the creation of districts optimized along several axes, vastly improving on previous single-objective optimizers. Along the way, we give a novel proof that the balanced connected problem, a well-known NP-complete problem, remains NP-complete for planar input graphs.

Acknowledgements

I am eternally grateful to my thesis advisor, Dr. Scott Niekum, whose advice and guidance was invaluable to me throughout the process of writing this thesis, and without whom this project would not have been finished. Risto Miikkulainen, Daron Shaw, Andrew Blumberg, Anna Gal, Peter Stone, and William Press all provided helpful feedback and answered my (many, many) questions throughout this process. I'd also like to thank David Fong, Nathan Berkowitz, Mia DuBose, Pato Lankenau, and Arnav Sastry for their help and support. And of course, I'm indebted to my parents, without whom I wouldn't be where I am today.

Contents

1 Gerrymandering in the United States **7**

1.1 Redistricting 7

1.2 Gerrymandering 11

1.3 The Voting Rights Act of 1965 15

1.4 Redistricting and the Supreme Court 17

1.4.1 "One person, one vote": *Baker*, *Reynolds*, and *Wesberry* 17

1.4.2 Racial gerrymandering: *Gingles* and *Shaw* 18

1.4.3 Partisan gerrymandering: *Davis*, *Vieth*, and *LULAC* . 19

1.4.4 *Shelby County* and the future of the VRA 20

1.4.5 *Gill* and the future of gerrymandering 21

1.5 Combating Gerrymandering 21

1.5.1 Quantifying Gerrymandering 21

1.5.2 Public Redistricting 25

1.5.3 Algorithmic Redistricting 26

1.5.4 Alternatives to Redistricting 26

1.6 Conclusion 27

2 Gerrymandering as a Mathematical Problem **28**

2.1 Modeling Redistricting 29

2.2 The Block Graph 30

2.3 The Balanced Connected Partition Problem 32

2.3.1 Some Complexity Theory 34

2.3.2 The Complexity of BCP 36
2.3.3 Planar BCP is NP-Complete 37
2.3.4 Some Other Complexity Results 40
2.4 Moving Past NP-Completeness 41
2.4.1 Hill Climbing 41
2.4.2 Genetic Algorithms 44
2.4.3 Multi-Objective Optimization 46
2.5 Conclusion 48

3 Evolutionary Redistricting **49**
3.1 The Block Graph Revisited 49
3.2 The Algorithm 51
3.2.1 Genetic Operators 51
3.2.2 NSGA-II 52
3.2.3 Local Search 54
3.2.4 Handling Invalid States 55
3.3 Implementation 56
3.3.1 Data Sourcing 57
3.3.2 From Data to Graph 61
3.3.3 Evolution: Encoding the Chromosome 64
3.3.4 Optimizing Hill-Climbing 66
3.3.5 Tracking Connected Components 67
3.4 Conclusion 70

4 Results and Future Work **72**
4.1 Experimental Setup 72
4.2 Evaluation on Toy Graphs 73
4.2.1 Single-Objective Optimization 73
4.2.2 Multi-Objective Optimization 76
4.3 Evaluation at the State Level 78
4.4 Inefficiencies 80

4.5 Future Work 83
4.5.1 Finishing Our Work 83
4.5.2 Improving Our Algorithm 84
4.6 Conclusion 85

Introduction

In 2003, the state of Texas sent 17 Democrats and 15 Republicans to the United States Congress. That year, for the first time in modern history, the Republican Party gained a majority of seats in the Texas Senate, completing their takeover of the state government. Conveniently, this was around the same time the state's Congressional district lines needed to be redrawn. The Republican Party created, proposed, and passed their plan with little opposition. And so, in 2004, after the new plan was put into effect, Texas sent 11 Democrats and 21 Republicans to Congress.

There are lot of reasons why this might happen; indeed, the Republicans won four percentage points more of the vote share between 2002 and 2004. But that shift in vote share doesn't correlate with a six-seat shift, and the 2003 redistricting was widely criticized for granting the Republicans an unfair advantage. The redistricting was challenged in the Supreme Court, but while the Court did reject the 23rd District as being unconstitutionally unfair to minority voters, they ruled that the plaintiffs didn't present satisfactory evidence of partisan bias. (As a side note, the rejection of the TX-23 did nothing to harm the overwhelming Republican majority, which continues to this day.)

The practice of drawing legislative district lines to artificially magnify minorities or eliminate majorities is called gerrymandering. Gerrymandering has been used by partisan lawmakers interested in securing electoral victories throughout the history of the United States. With the advent of modern

technology, politicians have found it easier to create potent gerrymanders that, on the surface, appear fair, but profoundly limit the ability for voters to have their interests represented well.

But, if technology can be used to draw better gerrymanders, it can also be used to draw fairer, less partisan district maps. Some work has already been done in this field. This paper proposes a general framework for building districts optimized to maximize fairness. In order to do that, though, we first need a solid understanding of how redistricting works, what good (and bad) district maps look like, and what laws exist governing redistricting.

This paper is divided into four sections. The first section discusses gerrymandering in detail, covering the rules that govern the process and the ideals to which the process is held. The second section lays the foundation for the mathematical models used in this paper. The third section introduces a new evolutionary framework for redistricting algorithms. The fourth section discusses implementation challenges and results, as well as future directions for this work.

Chapter 1

Gerrymandering in the United States

The lower house of the United States Congress, the House of Representatives, is composed of 435 representatives, elected by the population of political regions called districts. Each state has a different number of representatives, depending on the population of the state; Wyoming, the least populous state, has one delegate, while California, the most populous, has 53. Every ten years, following the decennial Census, each state is assigned a number of seats in the House in a process called reapportionment. Generally speaking, sometime between one to three years after the reapportionment, the states redraw their district lines in a process called redistricting. This section discusses how redistricting is supposed to work and where it can go wrong.

1.1 Redistricting

Redistricting, broadly speaking, is the process by which states divide themselves into distinct political regions called districts.[1] Each district functions

[1]Districts are also used at the state level for elections to state legislatures. While this work focuses exclusively on Congressional elections, it is not difficult to conceive of an

in Congressional elections as independent political units, and the representatives they select are given the mandate of solving the problems of their district. The way in which the districts are laid out is therefore of tremendous importance, since at the federal level each district is treated as a bloc. The losing side in district elections—the side with less support, even if by a slim margin—might find itself completely ignored by their representatives, whose mandate is granted by the winning side.

Article I, Section 4 of the United States Constitution states that "The Times, Places and Manner of holding Elections for Senators and Representatives, shall be prescribed in each State by the Legislature thereof." Because state legislatures are granted all but total control over the redistricting process, each state has a different set of policies and practices that govern its redistricting. Seven states—Wyoming, Montana, North Dakota, South Dakota, Alaska, Delaware, and Vermont—don't have a population large enough to qualify for more than one district. The other 43, while occasionally differing from one another in particular areas, generally hew to a set of standard practices. There are two federal requirements, however, that bind every state—equal population and preservation of communities of interest—which are discussed in more detail in Sections 1.4 and 1.5.

Who draws the lines?

In 37 states, control of the redistricting process is left entirely up to the state legislature. Generally speaking, both chambers must agree to a redistricting plan, and in every state but one (North Carolina), the governor is empowered to veto plans proposed by the legislature. The remaining six states rely on commissions, generally either chosen by the leaders of the legislature or appointed by the governor. In two states, New Jersey and Hawaii, these commissions consist of elected officials; in the other four—Arizona, California, Washington, and Idaho—the committees consist of unelected citizens, gener-

application to state-level redistricting as well.

ally redistricting experts. Several of the other 37 states also rely on advisory commissions, although these bodies only draw district plans themselves when their state legislatures cannot agree to a plan. In almost every state, where a decision cannot be made, the state's Supreme Court is empowered to unilaterally draw new district lines. (For a more detailed state-by-state breakdown, see [36] or [39].)

What should districts look like?

Aside from the two federal requirements mentioned above, states generally follow a consistent set of requirements for district plans:

1. **Contiguity** requires that all districts be geographically contiguous; that is, except in the case of islands, any two points within a district should be reachable by a path entirely within the district.

2. **Compactness** requires that "constituents within a district should live as near to one another as practicable" [39]. Compactness can be difficult to enforce, as there are several standards of geographic compactness (discussed below in Section 1.6).

3. **Communities of interest** are defined as a "group of people in a geographical area, such as a specific region or neighborhood, who have common political, social or economic interests" (see [39]). It's generally accepted that such communities should be represented as a bloc, and several states require that those communities be contained in a single district where possible. (As with compactness, since defining what exactly a community of interest is is tricky, this is a hard standard to hold to.)

4. **Political boundaries** like city or county boundaries are generally required to be preserved by districts. The idea behind this requirement is

that cities, like communities of interest, share political goals and should be represented as a bloc.

5. **Population equality** requires all districts to be as equal in population as is possible. Unlike the previous requirements, this is a hard requirement imposed at the federal level. It's generally accepted [39] that district maps whose least and most populous districts differ in population by more than one percent are presumptively (that is, without further consideration) unconstitutional.

6. **Minority representation** refers to the idea that minorities, who historically were targeted by gerrymanders, should have the same voice in Congress as the majority. Formally encoded in Section 2 of the Voting Rights Act of 1965, a federal law, the requirement is strict, and can be violated both intentionally and otherwise.

A district map that has those six qualities is difficult to come by. It's also not clear that such districts are representative of their population. What it means for a population to be well-represented is a complex question, but while those requirements aren't guaranteed to encapsulate a good district map by themselves—not in the least because they are largely qualitative, not quantitative—they do provide a good framework for thinking about district design.

How are districts drawn?

As mentioned above, districts generally hew to political boundaries. Historically, districts were drawn by combining counties or cities, since more granular construction was intractable until the advent of computer-aided redistricting. (Tracking down the first computer-aided gerrymander is challenging, since states didn't generally discuss the methods they used, but William Vickrey's proposal in 1961 [1] seems to be regarded as the earliest

mention of computer-aided redistricting [18].) Since then, however, computers have introduced a level of sophistication into district-drawing that makes gerrymandering easier than ever.

When the Census is taken, the country is divided into tiny regions called Census blocks. The United States contains 11,078,297 Census blocks, each of which typically has a population of less than 1,000 people. These blocks are grouped into 217,740 block groups, which are grouped into 73,057 tracts, which make up the country's 3,143 counties. Blocks are the most granular level at which Census data are available, although demographic data collected in the American Community Survey (which taken more frequently than the Census) is only published at the block group level. Every geographic unit partitions the country—that is, no two units overlap, and the units as a whole cover the entirety of the landmass in the United States.

Even with modern technology, redistricting at the block level can be computationally intractable. On average, there are more than 217,000 blocks per state, which can be partitioned into more combinations than there are atoms in the Universe.[2] Software like Maptitude[3], which overlays Census and political data over a block-level map of the United States, rather than calculating over this enormous space, allow district planners to modify existing district maps by moving blocks from one district to another and seeing the effect on potential elections.

1.2 Gerrymandering

Partisans have long recognized the power of redistricting to create partisan advantages where none exist. Given sufficient latitude over the redistricting process, it's possible to fracture large blocs of opposition voters into multiple districts, making them the minority in each, or pack them into one district,

[2]In particular, the number of partitions of an n-element set into k sets is given by $S(n, k)$, the Stirling numbers of the second kind. For context, $S(1000, 20) = 4.4 \times 10^{1282}$.

[3]See www.caliper.com/maptovu.htm.

isolating their influence to one seat in Congress. The term *gerrymandering* originated in 1812, when the Federalist governor of Massachusetts, Elbridge Gerry, created a district in northeastern Massachusetts so convoluted that it resembled the mythical[4] salamander.[5]

Broadly speaking, gerrymandering describes any use of redistricting to achieve a particular goal. Its history in the United States broadly falls into two categories: gerrymandering to silence a minority demographic, or racial gerrymandering, and gerrymandering to silence a political group, or partisan gerrymandering. Racial gerrymandering, infamously used to silence newly enfranchised black voters during the Jim Crow era, was outlawed by the Voting Rights Act of 1965. Partisan gerrymandering—the subject of this paper—has, despite having a longer and broader history, remained legal. The Supreme Court, as described below, has had several opportunities to outlaw the practice, but has yet to do so.

Gerrymandering can be achieved in several ways:

1. **Packing** places a voter bloc into a single district, thus guaranteeing them a victory but isolating their voice to that district.

2. **Cracking** divides a voter bloc into several districts, thus guaranteeing them a voice in multiple elections, but with no chance of victory in any.

3. **Hijacking** places two elected officials from the same party into the same district, thus creating a more competitive primary, possibly end-

[4]Some readers may be confused by the adjective "mythical" to describe salamanders. It should be noted that salamanders are, in fact, a real creature, and that the mythical salamander is distinguished from the biological salamander by a number of properties, for example affinity to fire and, apparently, odd geography.

[5]Many believe that the first gerrymander came not from Elbridge Gerry, but from the Antifederalist Patrick Henry, who in 1788 used his influence over the Virginia House of Delegates to draw a district map placing James Madison, a prominent Federalist, in a district that would have made it difficult to achieve victory. However, some recent scholarship (see [20]) suggests that this might not actually be the case, illustrating the difficulty of identifying partisan gerrymanders.

ing the tenure of a powerful incumbent, and potentially creating a winnable district for the other party from the other original district.

4. **Kidnapping** places popular candidates in competitive districts, thus making it harder to win elections.

The aim of gerrymandering differs from state to state (and from district to district), but generally speaking, gerrymanders fall into a handful of categories:

1. **Packing and cracking.** Packed and cracked district maps combine the first two methods above to dilute a vote.

2. **Bipartisan gerrymandering.** Bipartisan, sweetheart, or *status quo ante* gerrymandering seeks to preserve electoral results across redistrictings.

3. **Natural gerrymandering.** Some authors [24, 15] point out that gerrymandering can happen unintentionally, as young liberals tend to cluster in urban areas while older conservatives tend to live in rural areas, making it difficult *not* to gerrymander the population.

A common misconception is that gerrymandering is done with the aim of maximizing the number of districts won by one party, but this is only one potential goal. Gerrymandering might be done with the goal of winning only as many seats as is necessary to achieve a majority in the House of Representatives (making, for example, the requirements on a liberal state like Massachusetts for number of seats won by the Republican Party less important than those for a more conservative state like Texas). It also might be done to preserve a particular incumbent, even at the cost of other seats. In Wisconsin, for example, the Republican legislature would likely value ensuring a victory in the First District, home of Speaker of the House Paul Ryan, more than winning additional seats in Congress.

Arguably the most common strategy centers not around winning seats but around ensuring the other party wastes as many votes as possible. A wasted vote is one that doesn't impact the results of an election. Any vote for the losing party[6], or any vote for the winning party above the bare minimum needed to achieve victory, is considered wasted. Packing and cracking, as demonstrated in the example below, is particularly effective in wasting votes. The votes diluted in both packed districts (where the bloc generally wins by a larger-than-necessary margin) and cracked districts (where the bloc is forced to lose) are almost all wasted.

Packing and cracking in practice

A natural question to ask is how bad gerrymandering can get. Consider an election in a state S with 11 districts, each of which has 11 voters. Every voter votes for either party A or party B. Suppose that party A is in control of the redistricting process, and their goal is to hold the majority of seats in the state's delegation. The minimum number of seats they'd need is six, and in each of those six elections party A would only need six votes—a total of 36 votes. Meanwhile, the other five votes in each of the six districts—a total of 30 votes—could go to party B with no impact on the outcome. The other five districts could also go to party B in their entirety—a total of 55 votes—thus making the statewide vote count 36–85 against party A. But the delegation would still go 6–5 for party A. (See Table 1.1 below.)

In this example, party B's vote in districts 1 through 6 are cracked—in none of those districts do they have enough of a vote share to win the seat. All 5 votes for B cast in those elections are wasted, since B loses. In districts 7 through 11, party B's vote is packed—in each district, they're guaranteed a win, even though their votes would be far more effective elsewhere. Since

[6]In theory, there could be more than one loser, for example in multiparty systems. In the American context, where two parties are the norm, this is less relevant, and isn't considered in this paper.

District	Votes for A	Votes for B	Outcome	Wasted Votes
1	6	5	A	5 (B)
2	6	5	A	5 (B)
3	6	5	A	5 (B)
4	6	5	A	5 (B)
5	6	5	A	5 (B)
6	6	5	A	5 (B)
7	0	11	B	5 (B)
8	0	11	B	5 (B)
9	0	11	B	5 (B)
10	0	11	B	5 (B)
11	0	11	B	5 (B)
Total	**36**	**85**	**A (6–5)**	**55–0 (B)**

Table 1.1: Outcomes in S. Here, even though votes for Party B outnumber votes for Party A by a more than 2-to-1 ratio, Party A still controls the majority of S's delegation.

B only needs 6 votes to win, the additional 5 votes cast for B are also wasted. In no election does A waste votes—in the first six districts, they only cast as many votes as are needed to win the district, and in the last five, they cast no votes. Thus, B wastes 55 votes—45.4% of the votes cast, and 64.7% of the votes cast for B—while A, the party in control of the redistricting, wastes no votes.

1.3 The Voting Rights Act of 1965

The Voting Rights Act of 1965, passed during the height of the Civil Rights Era, outlawed voter disenfranchisement on the basis of race. The Act created an enforcement mechanism for the voting rights guaranteed by the Fourteenth and Fifteenth Amendments. The VRA is divided into 19 sections, three of which are of particular importance here: Section 2, Section 4(b), and Section 5.

Section 2 reads in its entirety:

> "No voting qualifications or prerequisite to voting, or standard, practice, or procedure shall be imposed or applied by any State or political subdivision to deny or abridge the right of any citizen of the United States to vote on account of race or color."[7]

Section 2 specifically enjoins any behavior that can be used to make voting more difficult for racial minorities, protecting the right to vote regardless of race enshrined in the Fifteenth Amendment. That broad provision has been used, among other things, to strike down racially gerrymandered district maps.

Section 5, which was designed to ensure state compliance with Section 2, requires certain jurisdictions to submit any change to their voting practices—including, notably, any and all changes to their district plans—to the Department of Justice for approval. Which jurisdictions are subject to Section 5 was determined by the preclearance rule in Section 4(b), which, despite being amended periodically to reflect changes in racially discriminatory behavior, was struck down as unconstitutionally outdated in 2013. Now, the only way for a jurisdiction (generally states) to be subject to Section 5 is for a court to "bail in" that jurisdiction, which rarely happens.

Although the VRA is considered one of the most effective pieces of civil rights legislation in American history, its critics have noted that Section 5 has had the unintentional effect of worsening racial gerrymandering. In particular, it's been noted that Republican leaders have used a requirement that a certain percentage of every state's districts be majority-minority (to the extent possible)—a requirement created to prevent cracking of minority votes—as an excuse to pack their votes instead. (See, for example, [25] and [27] for a further discussion of how the GOP used the VRA to their advantage.)

[7]Voting Rights Act of 1965, Pub. L. No. 89-110, 79 Stat. 437 (1965).

1.4 Redistricting and the Supreme Court

For the majority of its history, the Supreme Court held that redistricting was a question of a "political nature"[8], thus rendering it outside of the realm of questions that could be answered by the Court.[9] But, starting in 1962, the Court reversed itself and started to hear districting-related cases. This section discusses three important "lines" of cases, as well as recent developments that may affect the future of gerrymandering.

1.4.1 "One person, one vote": *Baker*, *Reynolds*, and *Wesberry*

Colegrove (see the footnote above) decreed redistricting questions to be outside the purview of the Court. The Warren Court reversed that decision, however, in *Baker v. Carr*[10]. At the time, the Tennessee state district plan contained rural districts far more sparsely populated than the urban districts. The case hinged on an equal-protection argument derived from the Fourteenth Amendment; the Tennessee state legislature cited the political-question doctrine, claiming that the Court couldn't provide a remedy. Justice William Brennan's plurality opinion articulated a test of justiciability, which found against the district map[11].

Baker instituted the requirement that at least one chamber of the state legislature should have districts of equal population. *Reynolds v. Sims*[12] expanded on this, holding both chambers to what Chief Justice Warren termed

[8] *Colegrove v. Green*, 328 U.S. 549 (1946).

[9] The political question doctrine, set forth in *Marbury v. Madison*, 5 U.S. 137 (1803), holds that some questions are better decided by the people than by the Court, making them undecidable by the Court. See *Political Questions, Public Rights, and Sovereign Immunity*, 130 Harv. L. Rev. 723 (2016).

[10] 369 U.S. 186 (1962).

[11] However, because Brennan's opinion failed to attract a majority, the decision was simply sent back to the lower court.

[12] 377 U.S. 533 (1964).

"one man, one vote" in his opinion for the majority. Because the requirement was imposed on both chambers, whereas State Senates had typically functioned like the U.S. Senate and represented counties equally, they now functioned like their lower-house counterparts and also represented their districts proportional to their population. *Wesberry v. Sanders*[13] bound Congressional districts to "one person, one vote," but, unlike *Reynolds*, did not impose the same requirement on the Senate.

1.4.2 Racial gerrymandering: *Gingles* and *Shaw*

Baker declared redistricting questions to be justiciable. The Voting Rights Act added a significant avenue for claims to be brought before the Court against racial gerrymanders. *Thornburg v. Gingles*[14], which invalidated a North Carolina district map for violating Section 2 of the VRA, laid out a set of preconditions for future Section 2 claims, called the "*Gingles* factors". In particular, if a minority group is geographically compact and numerous enough to be able to form a majority in a reasonable district, votes as a bloc, and is opposed by a majority that also votes as a bloc against the interests of the minority, then that minority could have a claim against the state for diluting their vote.[15]

However, *Shaw v. Reno*[16] found that district plans that consider race as a factor must be held to the standard of strict scrutiny.[17] *Shaw* involved

[13] 376 U.S. 1 (1964).

[14] 478 U.S. 30 (1986).

[15] The *Gingles* factors are merely preconditions; that is, a minority could show the existence of all three factors without proving that a district map is discriminatory.

[16] 509 U.S. 630 (1993).

[17] The term originates from Footnote 4 of Chief Justice Stone's opinion in *United States v. Carolene Products, Co.*, 304 U.S. 144 (1938). The doctrine of strict scrutiny, broadly speaking, holds that laws that discriminate against a particular minority must be held to a higher standard to pass Constitutional muster. In particular, the footnote asks "whether prejudice against discrete and insular minorities may be a special condition, which tends seriously to curtail the operation of those political processes ordinarily to be relied upon to protect minorities, and which may call for a correspondingly more searching judicial inquiry."

another North Carolina district map, but unlike in *Gingles*, the plaintiffs were white and claimed reverse discrimination, arguing that black votes were unnecessarily magnified. *Shaw* forced states to walk the thin line between unnecessarily racially-motivated (unconstitutional under *Shaw*) and not representative enough of minorities (illegal under the VRA).

1.4.3 Partisan gerrymandering: *Davis*, *Vieth*, and *LULAC*

The picture is a little murkier for partisan gerrymandering. In *Davis v. Bandemer*[18], the Court declared partisan gerrymandering claims to be justiciable under the Fourteenth and First Amendments, but limited the cases in which those claims could be brought before the Court. Then, in *Vieth v. Jubilerer*[19], a plurality voted to reverse *Davis* and declare partisan gerrymandering claims non-justiciable. Justice Kennedy's concurrence, however, declined to do so, and left open the possibility of striking down a partisan gerrymander, but only if a reasonable standard were to be presented to the Court: "I would not foreclose all possibility of judicial relief if some limited and precise rationale were found to correct an established violation of the Constitution in some redistricting cases."[20]

Recently, in 2006, the Court heard a challenge to the 2003 Texas redistricting, in *League of United Latin American Citizens v. Perry*[21]. Prior to 2003, the state legislature failed to reach a decision on a district map, so a panel of judges drew the map in accordance with state policy. In 2003, however, after the Republican Party won control over the Texas Senate (see the Introduction to this paper), the map was redrawn, ostensibly to favor the Republicans (thus leading to the Democratic six-vote loss in the 2004

[18] 478 U.S. 109 (1986).
[19] 541 U.S. 267 (2004).
[20] *Vieth*, Kennedy, J., concurring.
[21] 548 U.S. 399 (2006).

elections). Though the map was not just highly favorable to Republicans but also drawn in the middle of the decade, after a plan had already been instituted, the Court in *LULAC* failed to find evidence of an unconstitutional partisan gerrymander, only striking down one district on a claim of unconstitutional racial vote dilution under the *Gingles* test. The decision in *LULAC*, however, was highly fractured, resulting in six different opinions. No decision was made (that is, no opinion attracted a majority) on partisan gerrymandering.

1.4.4 *Shelby County* and the future of the VRA

A year after the VRA was passed, the law was brought before the Supreme Court in *South Carolina v. Katzenbach*[22], where a unanimous Court, in an opinion authored by Chief Justice Warren, found that the majority of the VRA was constitutional. (Justice Hugo Black concurred in part, but would have struck down Section 5, the preclearance requirement, as unconstitutional.) Recently, however, in 2013, the Court once again faced a challenge to the VRA in *Shelby County v. Holder*[23]. This time, the 5-4 decision struck down Section 4(b), the coverage formula for preclearance, as unconstitutional, finding that the formula was fundamentally rooted in data from the 1960s and thus too outdated to be a good standard. Although *Shelby County* didn't explicitly strike down Section 5 as well, thus leaving the preclearance requirement intact, it is effectively toothless, since no jurisdiction is subject to it by default. (See [26] for a discussion of the future of the VRA after *Shelby County*.)

[22] 383 U.S. 301 (1966).
[23] 570 U.S. 2 (2013).

1.4.5 *Gill* and the future of gerrymandering

In October 2017, the Court heard arguments in the highly-anticipated *Gill v. Whitford*, a case centering around the constitutionality of partisan gerrymandering. At the core of the discussion in *Gill* is the efficiency gap, a proposed measurement of partisan gerrymandering (discussed in more detail below). The expectation is that Justice Kennedy will be the swing vote in the case; if the efficiency gap satisfies his challenge for a standard of partisan gerrymandering issued in *Vieth*, he may side with the liberal wing of the Court in finding the practice unconstitutional. (See [35] for a deeper discussion of the background of *Gill*.) The Court is expected to issue a decision in June 2018.

1.5 Combating Gerrymandering

In the absence of judicial action on partisan gerrymandering, several independent initiatives have sprung up to try and curtail the practice. This section discusses four of these: the attempt to establish a quantitative measure for gerrymandering, the attempt to democratize the redistricting process, the attempt to automate the redistricting process, and the attempt to modify the American system of legislative elections.

1.5.1 Quantifying Gerrymandering

Justice Kennedy's concurrence in *Vieth* left open the possibility for partisan gerrymanders to be struck down under a consistent standard (or set of standards). Three such standards are discussed here: compactness, the efficiency gap, and redistricting probability distributions.

Compactness

Texas' 35th district (see figure 1.1) is one of the most blatantly gerrymandered districts in the country. Snaking along a sparsely populated tract

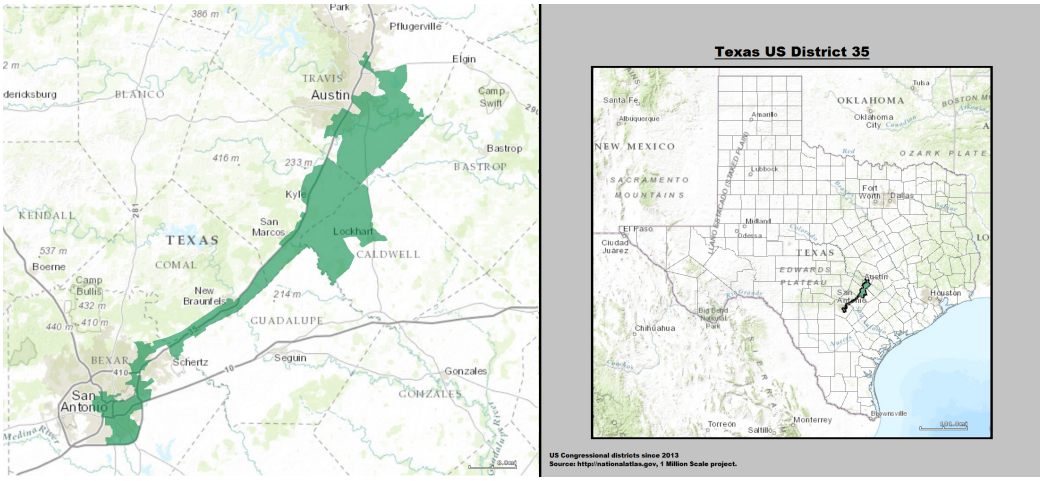


Figure 1.1: Texas' 35th District.

of Interstate 35 between Austin and San Antonio, connecting two largely Democratic areas of both cities, the district is guaranteed to be held by the Democratic Party. The district captures most of the Democratic vote in Austin, leaving the rest of the heavily Democratic city divided into four other districts (TX-10, 17, 21, and 25), all of which are strongly Republican.

It's not hard to look at the long, snaky shape of the district and conclude that a more normal district plan wouldn't include a district that looks like TX-35. The formal measurement of the oddness of the district is known as "compactness", and refers to any measure that attempts to quantify the regularity or irregularity of a geometric object. As mentioned above, 18 states require that their Congressional districts pass muster in a compactness test.

One of the most common compactness metrics is known as the *isoperimetric inequality*, and measures the ratio of the area of the shape to the area of a circle with the same perimeter. Formally, if L is the perimeter and A is the area, then the isoperimetric inequality is given by

$$\frac{4\pi A}{L^2} \leq 1,$$

where equality holds if the shape is a circle. This metric forms the basis of the Polsby-Popper test, which is cited as the most commonly used compactness test. (For a survey of other geographic measures, see [16].)

However, purely geometric analysis of partisan gerrymandering has been criticized as reducing the problem to one of drawing pretty shapes and ignoring the need to represent people well. (See [30] for a deeper discussion of this criticism.) Some authors (see, for example, [19]) have proposed metrics that measure the distance between voters and the geometric centroids of their districts. Those metrics allow for districts that respect population distribution.

The Efficiency Gap

In 2015, Eric McGhee and Nicholas Stephanopoulos proposed the "efficiency gap" standard (see [31]), which is now at the center of the claim in *Gill*. It measures the percentage of votes wasted in an election (see above for a discussion of wasted votes) relative to the total number of votes cast, and uses that percentage to calculate the number of seats which may have been decided as a result of partisan gerrymandering.

Suppose that party A wastes a votes, and party B wastes b votes. Then the efficiency gap is measured as

$$e = \frac{|a - b|}{a + b},$$

and is the percentage of seats that might have gone to the party with more wasted votes, had those votes not been wasted. So, for example, if $e = 0.3$ across elections in ten districts, then the party that wasted more votes might have won three more seats, had those votes not been wasted.

Because of its relation to the packing and cracking strategy—packed votes are wasted because they provide unnecessary support to the winner, and cracked votes are wasted because they go to the loser—the efficiency gap has

been proposed as a metric to measure partisan gerrymandering. McGhee and Stephanopoulos originally proposed that any district map with $e > 0.2$ in state-level races or more than two wasted seats in Congressional races could be considered presumptively unconstitutional. In the election described in table 1.1, for example, $a = 0$ and $b = 55$, so

$$e = -\frac{55}{85} = 0.647,$$

and thus, since 64.7% of 11 is 7.12, and $b > a$, party b might have won a staggering 7 more seats had they not been the victims of partisan gerrymandering.

The District Court for the Western District of Wisconsin found the efficiency gap justiciable in their decision. Their decision, however, only binds that district; the Supreme Court's decision will bind the entire country. If they uphold the Western District's decision, the Court will, for the first time, institute a formal, quantitative test of partisan gerrymandering, potentially paving the way for an overhaul of every state's redistricting practices.

Probability Distributions

Recently, a team at Duke University [29, 32] led by Jonathan Mattingly devised a new way of measuring the "goodness" of a district map. Any district map can be measured with some utility function (population equality among districts, compactness, etc.), but it's difficult to tell in isolation whether that score legitimately reflects the quality of the map or merely reflects the underlying space (that is, no better districting exists).

Mattingly's team instead uses the Metropolis-Hastings variant of the Monte Carlo algorithm to sample a probability distribution over the space of valid district maps. This approach can be used to find the probability of a district map occurring given a random selection from the space of district plans, and was used to show that the post-2010 census North Carolina dis-

trict map was highly unlikely, especially compared to a map proposed by an independent expert panel.

1.5.2 Public Redistricting

Four states grant control of the redistricting process to independent commissions, but these commissions are generally comprised of experts nominated by state party leaders. California has a much more involved process. The commission consists of 14 citizens—five Republicans, five Democrats, and four nonpartisans—semirandomly selected from a pool of 60 highly qualified experts, as determined by a state board. The commission has come under fire from politicians, whose incumbency may be challenged by the commissions' district maps (which, as is the norm, are not reviewable by the state legislature). A similar commission in Arizona was challenged on constitutional grounds (the petitioners claiming that Article I, Section 4's grant of redistricting authority to the state legislatures was violated by the commission's independence), but the Court voted to uphold the commission.[24]

The California commission raised an interesting question—can citizen commissions do a better job of drawing district maps, and, more broadly, can ordinary people, not necessarily gerrymandering experts, draw good districts? Recently, a number of open software packages—see, for example, [37]—have opened up the process to anyone who has access to a computer. A number of groups have run redistricting competitions—FixPhillyDistricts[25], for example, was aimed at drawing better city council districts.

The impact of these projects is yet to be seen (see [22], for example, for a discussion of the impact of public redistricting projects). It's not inconceivable, however, that more states will follow in the California model. And, although the chances of their being instituted as law are small, public redis-

[24]See *Arizona State Legislature v. Arizona Independent Redistricting Commission*, 576 U.S. ____ (2015).

[25]http://www.fixphillydistricts.com/.

tricting competitions could help to broaden public awareness of the problems and potential issues latent in redistricting.

1.5.3 Algorithmic Redistricting

As mentioned above, the idea of using computers as a tool in drawing districts has been around since the 1960s [18]. Computers have been used repeatedly as tools to assist in the creation of district maps—Maptitude and DistrictBuilder are both prominent software packages used in some capacity to make districts—but only recently have computers gained the capacity to draw districts entirely from scratch. Few such projects have gained significant notoriety, but as computers continue to scale and as data continues to become available, it's not unlikely that the redistricting process will involve less and less human influence. The remainder of this paper discusses this idea in far greater detail.

1.5.4 Alternatives to Redistricting

Finally, a number of proposed modifications to the American electoral system would render obsolete the practice of gerrymandering. Some have proposed alternative systems of voting altogether—the Fair Represention Act (H.R. 3050, 2017), proposed by Don Beyer (D-VA), would "establish the use of ranked choice voting in elections for Representatives" and "require States to conduct Congressional redistricting through independent commissions," among other provisions. While the Act could threaten the current majority in the House and is thus unlikely to even make it out of committee, Beyer's proposal and others like it could end gerrymandering as a tool to gain partisan advantages. (See [33] for a discussion of the proposal.)

Others have proposed a game-theoretic solution to gerrymandering. By approaching redistricting as a competition between two rational actors (namely, the two parties), Landau and Su [28] devised a system by which the two

parties would collaboratively draw district maps in a fashion similar to two-player cake-cutting (see, for example, [40]). Broadly speaking, one party would make a "cut" of the state, but the other party would choose a "slice" to gerrymander, thus creating a framework where each party could, through intelligent cutting, limit the potential of the other party to effect a gerrymandered advantage.

1.6 Conclusion

Redistricting is a difficult problem to solve, and even good-faith approaches can yield undesired results. Meanwhile, gerrymandering can subtly undermine the process in a way that's difficult to measure. The VRA, the only significant federal action addressing the problem, has enjoyed tremendous success at ending racial discrimination at the voting booth, but has yet to prove effective in combating discrimination at the mapmaking level. Furthermore, the Court has declined to act, despite being given many opportunities to do so, citing reasons from nonjusticiability to a lack of a justiciable metric. That may change in *Gill*, however.

A variety of approaches have been suggested to end gerrymandering, among them several computational solutions. Coupled with the right metric, a computational approach may prove to be the silver bullet in ending partisan gerrymandering. The next chapter further explores this idea of gerrymandering as a computational problem.

Chapter 2

Gerrymandering as a Mathematical Problem

Chapter 1 introduced the idea of gerrymandering as a political problem, one that in effect pits politicians against their own constituents. But gerrymandering is more than just a political issue. At its core, the problem of redistricting is a mathematical one, as follows. Suppose we have a bounded region of the Cartesian plane, $S \subset \mathbb{R}^2$, and a population function $p : 2^S \to \mathbb{Z}_+$ that takes subsets of S to integers. In particular, p is subject to the restrictions that for any two subsets A and B of S, $p(A \cup B) = p(A) + p(B) - p(A \cap B)$ and that $P(\varnothing) = 0$. A district map of S that divides S into d districts is equivalent to a partition $P = \{P_1, P_2, \ldots, P_d\}$ of S, such that $p(P_i) = p(P_j)$ for all i and j.[1]

This is, of course, a very simple definition of redistricting that only requires population equality. As we have seen, the requirements of a valid district map vary, and are considerably more complex. Moreover, in order for a practical solution to be reached, a more concrete model than the one sketched above is needed. This chapter details that model, and lays the

[1] A partition of a set X is a set $P = \{X_1, \ldots, X_n\}$, such that $X_i \subset X$ for all i, $X_i \cap X_j = \varnothing$ for all i and j, and $\bigcup_i X_i = X$.

groundwork for the discussion of the technical implementation in Chapter 3.

2.1 Modeling Redistricting

Given the problem as stated above, redistricting boils down to partitioning a region of space into non-overlapping sections. There are two ways of modeling this problem: the continuous model, where the space is assumed to be infinitely divisible, and the discrete model, where it's assumed to consist of indivisible atoms that can be arranged.

Redistricting within the continuous model is the domain of the *shortest-splitline algorithm*, developed by the Center for Range Voting. The algorithm takes as input a state S and the number of districts d, and works as follows:

1. If $d = 1$, return S.
2. Let $a = \lceil \frac{n}{2} \rceil$ and $b = \lfloor \frac{n}{2} \rfloor$, such that $n = a + b$.
3. Find L, the set of all line segments ℓ that divide the state into regions A_ℓ and B_ℓ with a population ratio of $a : b$.
4. Let ℓ^* be the shortest such line segment, and denote its resultant regions A_{ℓ^*} and B_{ℓ^*}. Recurse on each, with district counts a and b.

This algorithm will result in n districts of equal population, but suffers for two reasons. First, while population equality is certainly the most important requirement, being the only federal one, it is far from the only goal. Moreover, the assumption the algorithm makes is that any line segment that divides a state can function as a district boundary. Not only does this assumption make it all but impossible to maintain political boundaries, it makes it all but impossible to even maintain property boundaries. That is, a line drawn by the algorithm could easily end up cutting through a house, causing awkward issues around which district its residents should vote in.

This is more than just an issue with the shortest-splitline algorithm. It is, in fact, a problem with *any* districting algorithm that operates in the continuous domain. Whereas a continuous domain assumes an infinitely divisible search space, as noted above, a discrete domain allows for atoms. The smallest possible atom would be at the property level, therefore preventing houses from being split among districts. Aside from the fact that the number of distinct properties in the average state is incredibly large, population and demographic data is not available at the property level.

Instead, there is a slightly coarser level at which we can atomize the state—the Census block (see Section 1.1 above). Blocks are drawn to respect city, county, and state boundaries, as well as property boundaries. Districts comprised of census blocks will therefore respect property lines, avoiding the kind of divide possible in the continuous model. Drawing districts thus reduces to the relatively simpler problem of assigning blocks to districts, which, as discussed above, is computationally intractable. We can reduce the search space by imposing a graph structure on the block map. How this is done is described below.

2.2 The Block Graph

A graph is a collection of points V, with some (possibly empty) collection E of lines connecting the points in V. The points are called vertices (or nodes), and the lines are called edges. The study of graphs is called graph theory, and has extensive practical applications in computer science. Graphs are useful in that they provide a common language for discussion of a wide range of problems, from map traversal to social network analysis.

Here, we represent the block-level division of a state as a graph, which we refer to as a *block graph*. Each block is assigned a vertex, and edges connect vertices whose blocks share a linear border. That is, blocks that touch at a single point, like those diagonal to each other in a grid, are not

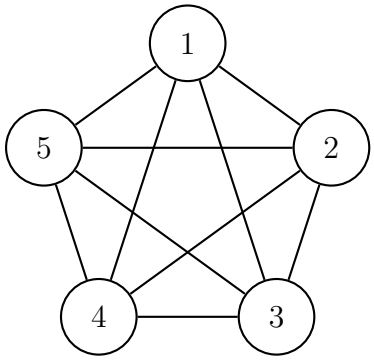


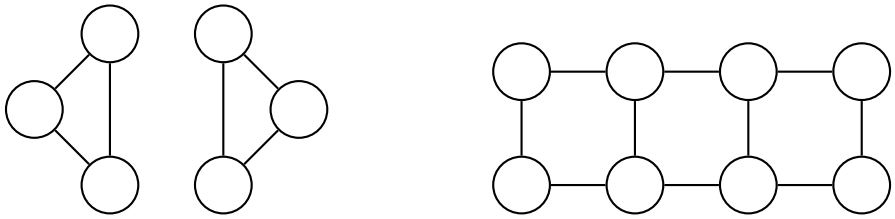
Figure 2.1: A graph with five vertices and ten edges. This graph is not planar, since it can't be drawn without edges crossing each other. It *is* connected, since there is a sequence of edges between every pair of vertices.

connected. A block on an island or that is otherwise disconnected from the rest of the graph is connected to the "mainland" (so to speak) by an edge connecting it to the block whose centroid is closest to its own. Formally, then, if the vertex set is V, the edge set is given by $E = A \cup B$, where $A = \{(u, v) \mid u, v \in V \text{ and blocks } u \text{ and } v \text{ share a border}\}$ and

$$B = \{(u, v) \mid \forall\, v' \in V\, (u, v') \notin A \text{ and } v = \arg\min_{v' \in V} \operatorname{dist}(u, v')\}.$$

In order to define the redistricting problem in the language of graphs, it's useful to introduce a few definitions. A *path* between two vertices u and v is a sequence of edges, $P = \{(x_1, x_2), (x_2, x_3), \ldots, (x_{n-1}, x_n)$, where $x_1 = u$ and $x_n = v$. A *connected* graph is one where every pair of vertices has a path between them. Finally, for some $V' \subseteq V$, the *induced subgraph* on V' is the graph $G' = (V', E')$, where E' contains only those edges that connect vertices in V' to each other (that is, $E' = \{(u, v) \in E \mid u, v \in V'\}$), and is denoted $G[V']$.

Finally, a *weighted* graph is one that is equipped with a weight function $w : V \to \mathbb{R}$ (that is, one that assigns to every vertex a real value), such that $w(v)$ is the weight of vertex v. (Unweighted graphs can be seen as



(a) A planar, disconnected graph. (b) A planar, connected graph.

Figure 2.2: Some examples of graphs.

having a weight function w that assigns to every vertex a constant weight.) Graphs can have multiple weight functions, and the block graph certainly does. Each block has a population, a number of votes cast for one party, a number of votes cast for the other, and so on, each of which can be realized as a weight function from vertices to the real numbers (or more specifically to the positive integers). Although w technically takes vertices to their weights, it's also convenient to introduce the notation $w(X) = \sum_{v \in X} w(x)$ for some set $X \subseteq V$ of vertices in the graph.

2.3 The Balanced Connected Partition Problem

The problem of dividing a state into d districts is equivalent to the problem of finding a d-partition[2] $P = \{V_1, V_2, \ldots, V_d\}$ (see footnote above) of the vertices of the block graph, such that the induced subgraphs on each V_i are connected. Furthermore, given a balance function that gives the total weight of the smallest partition, $B = \min_{V_i \in P} w(V_i)$, find the partition that maximizes B.

That is, find the partition whose subsets both induce connected subgraphs and are as equal in population as is possible. This problem is known as the balanced connected partition problem, or BCP, and has been discussed

[2]A partition consisting of d subsets.

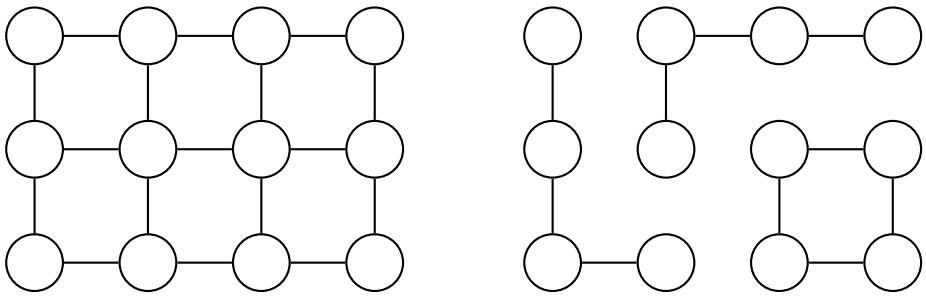


Figure 2.3: A planar graph (in this case, a grid graph) G, and a maximally balanced 3-partition of G.

extensively in the literature. When the graph is unweighted (which, as noted above, is equivalent to a graph with uniform weight), the problem is known as 1-BCP, and the balance function $B = \min_{V_i \in P} |V_i|$ gives the *size* of the smallest partition instead of the total weight.

The input to BCP generally contains the graph we wish to partition, as well as the number of partitions q into which the graph should be divided. An algorithm that solves BCP is expected to be able to do so for any value of q. A variant of BCP, called BCP_q, relaxes that restriction. An algorithm that solves BCP_q is only required to do so for the specific value of q given in the problem name; a solution to BCP_2, for example, is only required to partition graphs into two components. A solution to BCP can thus be seen as a solution to BCP_q for all q. (There is an equivalent dichotomy for 1-BCP.)

Figure 2.3 demonstrates a simple example of an unweighted balanced connected partition. The graph in figure 2.3a, which has 12 vertices, can be 3-partitioned $S(12, 3) = 86256$ ways, only a handful of which are connected. Of these, only a handful will be maximally balanced. One of these ways is demonstrated in figure 2.3b. Since all three connected components are of size 4, the balance function $B = \min_{V_i \in P} |V_i| = 4$ is maximized. Figure 2.4 demonstrates a similar example, but on weighted graphs.

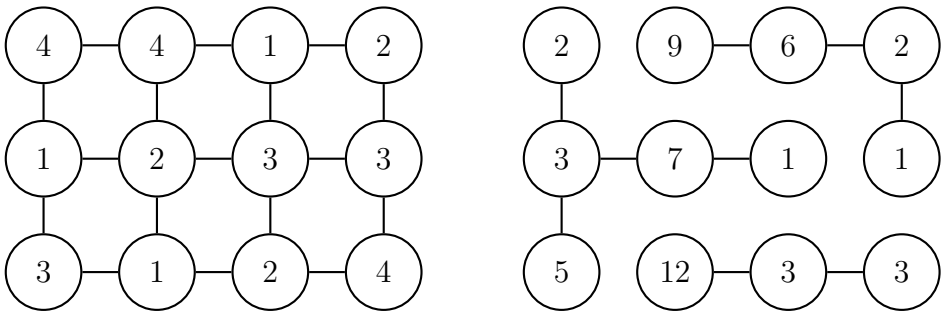


Figure 2.4: Here, the subgraphs are equal not in size, but in sum of the weights. Each component has weight sum 18.

2.3.1 Some Complexity Theory

In discussing problems like BCP, it's useful to define computational "hardness" in some standard way. Computer scientists say that a problem is "easy" if it can be solved efficiently, which generally means that if the size of the input is n, there is some algorithm that can solve it using no more than n^k operations for some k. The class of problems with this property is called P, or polynomial. (It's worth noting that since k is arbitrary, it's unclear that an algorithm that requires n^{100} operations for an input of size n is very fast. In the grand scheme of computational complexity, though, it turns out n^{100} isn't actually too bad.)

P stands for polynomial, so it's commonly assumed that NP stands for not polynomial. In fact, it stands for nondeterministic polynomial, and refers to the class of problems whose solutions can be verified in polynomial time. Any problem in P is obviously in NP, since checking that a solution is correct is as simple as running the solution-finding algorithm and checking equality. It's important to note, though, that the language of NP-ness can only be discussed for what are called *decision problems*, which ask a yes-or-no question (e.g., is n composite?), and not *optimization problems*, which ask for an answer that maximizes or minimizes some condition (e.g., what's the

shortest path from point a to point b?).

It's easy to reason about whether a solution to a decision problem can be checked quickly—for example, if we're asked if n is composite, and we say "yes, and here's a nontrivial factor q," our answer can be verified quickly (i.e., by making sure q divides n). On the other hand, optimization problems are harder to check—if we're asked what the shortest path between a and b is, and we say "here's a path p with length ℓ," while it can certainly be checked that p goes from a to b, it's not clear that there isn't a shorter path.

Luckily, it's easy to convert an optimization problem to a decision problem, and vice versa, as follows. Suppose we have an optimization problem that asks for a solution that minimizes some function f. The equivalent decision problem is as follows: "Is there a solution whose f-value is at most k?" It's possible, by asking the question of at most a logarithmic number of values of k, to find the solution with the minimal value of k.[3] The other direction is simpler: clearly, any solution to the optimization version of a problem is a solution to the decision version of the problem.

Suppose there were a fast (that is, in a polynomial number of operations) way of converting a solution to a problem A into a solution to a problem B. For example, converting a solution to the problem "Is n composite?" to a solution to the problem "Is n prime?" is trivial. Suppose further that B was known to be difficult. Then it must be true that A must also be difficult, since if a fast solution to A were possible, and there were a fast way of turning a solution to A into a solution to B, there would be a fast way of solving B as well. This method of finding what are called polynomial-time reductions is a common way of proving difficulty by comparison, and are used to define two additional complexity classes.

The first class is called NP-hard, and contains problems, informally speaking, at least as hard as the hardest problems in NP. More formally, H is NP-hard if, for every problem L in NP, there is a polynomial-time reduc-

[3]That is, performing binary search.

tion from L to H. NP-hard problems don't necessarily have to be in NP themselves, and those that are are referred to as NP-complete. NP-complete problems are special because a polynomial-time solution to any such problem is equivalent to a polynomial-time solution to *all* such problems, and indeed a fast solution to every problem in NP. If such an algorithm were found, then P and NP would be equal; the question of whether P equals NP is one of the most famous open problems today.

2.3.2 The Complexity of BCP

It's unclear when BCP was first mentioned in the literature, but one of the earliest and most important results about it is called the Győri-Lovász Theorem, named for two mathematicians who discovered it independently (see [3] and [2]). It concerns k-connected graphs, which are graphs that can "tolerate" the removal of up to $k-1$ vertices without becoming disconnected, and is as follows:

Theorem 1 (Győri-Lovász)**.** *Let $k \geq 2$ be an integer and suppose that $G = (V, E)$ is a k-connected graph. Let $v_1, v_2, \ldots, v_k$ be distinct vertices of G, and let $n_1, n_2, \ldots, n_k$ be integers whose sum is $|V|$. Then G has disjoint connected subgraphs $G_1 = G[V_1], G_2 = G[V_2], \ldots, G_k = G[V_k]$ such that $|V_i| = n_i$ and $v_i \in V_i$ for all $1 \leq i \leq k$.*

[14] puts the earliest polynomial-time algorithms for $q = 2$ and $q = 3$ in the mid-nineties, while [7] appears to be the earliest polynomial-time algorithm for all q. The Győri-Lovász Theorem is enough, however, to give the following complexity result:

Theorem 2. *q-connected 1-*BCP$_q$ *is in P.*

For general graphs, though, Dyer and Frieze (see [5]) showed that 1-BCP$_q$ is NP-hard, and noted that NP-hardness holds for planar graphs when the

	1-BCP		BCP	
Graph Type	1-BCP$_q$	1-BCP	BCP$_q$	BCP
general	NP-hard [5]	NP-hard	NP-hard	NP-hard
q-connected	P [3, 2]	P	NP-hard [14]	NP-hard
planar	NP-hard [5]	NP-hard	NP-hard	NP-hard
ladders	P [11]	P	P [11]	P
trees	P [4]	P	P [4]	P
grids	P	P	NP-hard [10]	NP-hard
bipartite	NP-hard [5]	NP-hard	NP-hard [5]	NP-hard

Table 2.1: Summary of complexity results for BCP.

size of the partition sets is more than 3.[4] Furthermore, Chataigner et al. (see [14]) showed that BCP$_q$ is NP-hard, even on q-connected graphs (thus negating the possibility of an extension of Győri-Lovász to BCP$_q$).

Table 2.1 gives a small subset of complexity results for BCP. Note that in general, if on some class of input graphs BCP$_q$ (respectively 1-BCP$_q$) for all $q \geq 2$ is NP-hard, then BCP (resp. 1-BCP) on that class must also be NP-hard. Recall that BCP includes q in the instance. Then, as long as $q \geq 2$, BCP can just call BCP$_q$ for that q-value, and the result of BCP$_q$ for that q is equivalent to the solution for BCP. (If $q = 1$, the partition is just the original graph, assuming it's connected.) Similarly, if 1-BCP$_q$ (resp. 1-BCP) is NP-hard, it's clear that BCP (resp. BCP$_q$) is NP-hard as well, since, as mentioned above, 1-BCP is simply an instance of BCP where the weight of every vertex is 1.

2.3.3 Planar BCP is NP-Complete

In the context of redistricting, though, block graphs are guaranteed to be *planar*—that is, they can be drawn in two dimensions without edges crossing

[4]Dyer and Frieze used slightly different notation, where rather than considering the optimization problem of maximizing the size of the smallest set, they tried to find partitions where each set has a fixed size k.

over each other[5]—which imposes a degree of simplicity on the structure of the graph. It's obvious that by implication, since it's known that planar 1-BCP is NP-hard (see [5]), planar BCP should be NP-hard as well. This paper contains a simpler proof of that fact, using a reduction from the subset sum problem.[6]

The subset-sum problem asks for, given a list of numbers X of size n and a desired sum $T \neq 0$, a subset of the list X', such that $\sum_{x \in X'} x = T$. Consider the following strategy for finding a solution to subset-sum:

1. Let G be a graph with two sets of vertices, one of size n and another of size 2, such that the nodes in each set are connected to every node not in their own set. Assign to the vertices in the size-n set weights $x_1, \ldots, x_n$. Let $S = \sum_{x \in X} x$, and assign weight $S - 2T$ to one node, denoted n, and weight 0 to the other node in the size-2 set. (See figure 2.5.) Note that G is planar.

2. Find a connected 2-partition of V, $P = \{V_1, V_2\}$, such that $w(V_1) = w(V_2) = S - T$.

3. Assume without loss of generality that $n \in V_1$, and note that V_1 cannot equal $\{n\}$ (since $w(\{n\}) = S - 2T$, not $S - T$)[7]. In particular, $V_1 = \{n\} \cup X'$, where $w(X') = (S - T) - (S - 2T) = T$. Thus, $V_1 \setminus \{n\}$ is a solution to subset-sum.

Note that step 2 is equivalent to solving BCP_2 on the planar graph G defined in step 1. Subset-sum is known to be NP-complete, however. Therefore, if planar BCP_2 were in P, subset-sum would also be in P, since the

[5]To see this, note that *any* map-like graph whose geographical equivalent is in two dimensions can be represented by a planar graph.

[6]To our knowledge, planar BCP has never been investigated directly. The reduction from subset sum only works for BCP and not 1-BCP, since unary subset sum is in P.

[7]This could work if $T = 0$, but a solution to subset-sum with $T = 0$ is trivially the empty set.

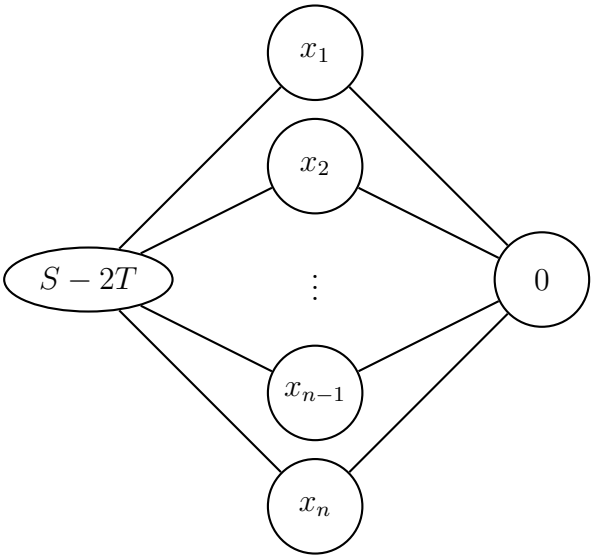


Figure 2.5: A 2-partition of this graph will necessarily contain one component with the node with weight $S - 2T$, along with some subset of the x_i's that sum to T.

strategy above reduces a solution to BCP_2 to a solution to subset-sum in polynomial time. We therefore arrive at a contradiction, and therefore:

Theorem 3. BCP_2*, even when restricted to planar graphs, is* NP*-complete.*

Extending this to planar BCP_q is straightforward. Note that the weight of node n is such that $S + w(n) = q(w(n) + T)$, where q is the number of partitions into which G is divided. Thus, for general q,

$$w(n) = \frac{S - qT}{q - 1}.$$

Whichever partition contains node n will necessarily also contain some subset of X that sums to T. This carries forward to BCP as well, since if BCP were not NP-complete, BCP_q could be solved quickly, simply by passing the input graph as well as q (which is known, not passed as input) to BCP. And thus:

Theorem 4. *BCP, even when restricted to planar graphs, is* NP*-complete.*

Thus, unfortunately, redistricting in the discrete graph model is NP-complete. This implies that any discrete model, in fact, cannot be districted quickly. There are approximation algorithsm for certain flavors of BCP, but it is known that BCP in general can't be approximated well.[8] There are no approximation algorithms in the literature for planar BCP, which is the problem we wish to solve, and it's conceivable that there may exist a reasonably accurate one that can be found. We study a different approach, as detailed in the next section.

2.3.4 Some Other Complexity Results

There are two other complexity results in redistricting that are worth mentioning before moving on. First, packing and cracking is NP-complete; see [17] for a discussion of this problem. Furthermore, gerrymandering in the Israeli context, which differs slightly from the American one, is also NP-complete. In this system, rather than being placed into districts, voters are permitted to vote at ballot boxes anywhere around the country, but are generally assumed to vote in the box closest to them. Politicians can choose to activate only a subset of ballot boxes, thus forcing voter behavior.

The question thus becomes: given knowledge of the voters' policy preferences (i.e., who they intend to vote for), is it possible to figure out which subset of the ballot boxes should be chosen, such that at each active box the plurality or majority of votes go to a desired candidate? The answer is, perhaps unsurprisingly, no; for more detailed discussion, as well as a study similar to this one on the potential of artificial intelligence in providing approximate solutions, see [34].

[8] That is, if α is the ratio of the balance function on the optimal partition to the balance function on the approximated partition, called the approximation ratio, it's known that no algorithm exists for which α is less than or equal to 6/5.

2.4 Moving Past NP-Completeness

NP-completeness, which (assuming $P \neq NP$) implies that BCP can't be solved quickly and deterministically by a computer, adds some difficulty to the problem of algorithmic redistricting. Thus, no algorithm can operate in polynomial time that guarantees a correct solution. This is, however, a very strict constraint. Instead, we can observe that the maximally balanced partition is one element of a (very, very large) set of connected partitions, some of which may not be balanced. This set in turn is a subset of the set of *all* d-partitions, not all of which are guaranteed to be connected. Thus, by enumerating all possible ways of partitioning a graph into d components, we are guaranteed to eventually find the maximally balanced partition.

This approach suffers from the obvious setback that to do so would take an unreasonable amount of time, because even for relatively small graphs, the space of all d-partitions is enormous. (Recall that in a graph with just 12 components, there there are over 86,000 ways of partitioning it into 3 components.) Finding elements in search spaces of this magnitude—sometimes referred to as combinatorial search spaces—is the domain of artificial intelligence, and it is from there that we draw our approach.

2.4.1 Hill Climbing

In the definition of BCP, we're given a balance function B that takes elements of the space of partitions of a graph G to the integers. The goal is to find the partition P that maximizes B. If B were differentiable, we could set the derivative equal to zero and solve for the maximizing function. It's clear, though, that B is not. In fact, B's behavior over the space of partitions is relatively unpredictable, making any kind of analysis of its maxima and minima impossible.

The goal of local search is, given such an unpredictable function, to find its extrema. Broadly, the approach is characterized by algorithms that take

a point (generally chosen randomly) in the space and investigating its neighbors, trying to find the direction with the steepest gradient and moving in that direction. In order to do so, the space must be endowed with some kind of locality, such that states have well-defined "neighbor" states.

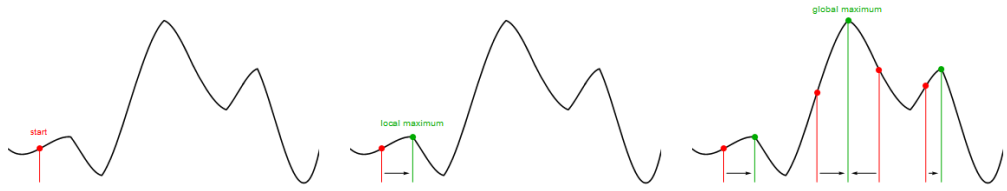
There are a variety of approaches to local search, but the simplest is known as hill-climbing. The algorithm is simple:

1. Choose a random state S.
2. Let $N(S)$ be the neighbors of S, and let $N^* = \arg\max_{N \in N(S)} B(N)$.
3. If $B(N^*) <= B(S)$, return S. Otherwise, set $S = N^*$ and repeat.

The algorithm terminates only when the best neighboring state is no better than the current state. (Some variants continue even if $B(N^*) = B(S)$ in step 3, but it's easy to see that unless every visited state is stored to ensure no states are repeated, which would take a lot of memory, this process can continue infinitely.)

The obvious problem with hill-climbing is that the algorithm returns states that are only better than all of their neighbors—but not necessarily the state that's better than *every* other state. That is, the algorithm can get "stuck" at local extrema, ignoring global ones. An approach called simulated annealing addresses this problem by allowing movement to lower-utility states with some probability p. As the algorithm continues, the probability decreases, favoring risk-taking at the beginning but becoming more risk-averse over time. The choice of p is important, though; for small values of p, the algorithm is not much better than hill-climbing (which, in effect, is simulated annealing with $p = 0$), and for large values of p, even global maxima will be skipped over.

Another approach is to run hill-climbing or simulated annealing multiple times, perhaps in parallel, and choosing the best value from all of the runs. The idea is that the more restarts happen, the better the chances will be that



(a) (b) (c)

Figure 2.6: Hill-climbing in action. The curve is a map from state space to utility, and the goal of the hill-climbing algorithm is to find the global maximum. The algorithm starts by choosing a random starting position (figure 2.6a), and moves in the direction that increases the score. Eventually, it gets to a point where it can't do so any more; this is a maximum of the curve (figure 2.6b). However, since this approach isn't guaranteed to find global maxima, the hill-climbing variant used in this work picks *several* random starting points, finds the closest maximum to each, and returns the maximum of those peaks (2.6c). As the number of random restarts increases, the probability of finding the global maximum increases as well.

a decent candidate will be found. This has some merit, but for search spaces as large as the set of partitions of a graph the number of random restarts needed can be intractable.

Although hill-climbing is simple in theory, the choice of the search space, and the definition of a valid move in the space, is an important one. Consider BCP as an example. The narrowest possible search space would only include partitions whose sets were equal in weight. The broadest possible search space would include any partition of the vertices, including those that wouldn't induce connected subgraphs. Finding initial elements in the former would be equivalent to solving BCP itself, but finding extrema in the latter would take a long time. Hill-climbing works best when the search space is small enough that solutions don't occur infrequently, but large enough that it isn't hard to find initial states or valid moves.

2.4.2 Genetic Algorithms

Another approach to local search takes inspiration from biological evolution. At a very high level, evolution can be seen as the interaction of DNA. Two members of a population mate and combine their DNA in some fashion, which yields another member of the population. Evolutionary algorithms work very similarly. Every element of a search space is mapped to a chromosome[9]. In the beginning (so to speak), these chromosomes are arbitrary, and over time experience growth in three stages:

1. Selection, the first stage, is the process by which members of the population are chosen for genetic combination. Selection methods vary, but generally members of the population are chosen based on their fitness. Some approaches, for example, choose parents randomly such that the chances that a member will be chosen is proportional to its fitness.

2. Crossover, the second stage, is the process by which two parents combine their chromosomes to yield children (generally two). Generally, some crossover point is chosen, such that one child will have parent A's genetic material up to that point and parent B's genetic material afterwards (and vice versa for the other child). Some algorithms choose a single crossover point, while others choose multiple.

3. Mutation, the final stage, randomly changes some part of the chromosome with some probability p. Generally speaking, p decreases over time, as the population becomes more stable (so to speak), although this differs from algorithm to algorithm.

Genetic algorithms work best when the problem has some degree of locality. In general genetics preserve sequences of base pairs (called genes) that have a high impact on the fitness of an individual. Locality, where two

[9]Chromosomes are most easily considered as arrays, which is analogous to the base-pair sequences of real DNA.

base pairs being close to each other on the chromosome implies some relation between them in the phenotype (that is, the member of the population generated by the chromosome), is thus more conducive to rapid discovery of maxima in the search space.

One major issue with genetic algorithms is that while the framework doesn't change much from problem to problem, it's only a framework, and requires the designer to make a number of choices along the way. There are a variety of methods of performing selection, crossover, and mutation, which have been documented extensively in the literature (see, for example, [8] for a survey). But perhaps the most important decision that has to be made is the choice of chromosome encoding.

Although chromosomes can technically be any arbitrary data structure, it's most common to represent chromosomes as arrays, which is similar to the way chromosomes are represented as sequences of base pairs in real-world evolution. The choice of array, though, and the meaning of each element, can have a dramatic effect on the efficacy of the algorithm, similar to how the choice of search space can make a big difference in hill-climbing.

Another major issue with genetic algorithms is that they can often get close to a good solution without ever reaching it. Because the relationship between parents and children is random, genetic algorithms are bad at hyper-local optimization. For this reason, some approaches, so-called "memetic" approaches, perform local search, like hill climbing or simulated annealing, after every generation. This comes at a performance cost, but has the advantage of restricting the population to local maxima only.

A further problem with genetic problems is that, depending on the choice of selection, crossover, and mutation operators, the population can often quickly converge to the best element in the current population, preventing any further optimization. After a certain point, of course, this is a good thing, but unless the algorithm happens to start with a very strong population that contains a global maximum, this can lead to premature convergence to a

non-optimal solution. This can be mitigated in several ways, for example by ensuring that the mutation rate is high enough to "force" generation of children whose genetic material differs from the rest of the population, or by allowing the selection operator to choose non-optimal, but genetically diverse, parents.

2.4.3 Multi-Objective Optimization

Up to this point, we've described local search algorithms that try and maximize a single objective function, like BCP's balance function. In many cases, though, it's desirable not just to maximize one function but to find the element of the space that maximizes multiple functions. This class of problems are generally known as multi-objective optimization.

The challenge of MOO is that it is very infrequently the case that a single element can maximize multiple functions at the same time. For example, consider a modification to BCP that combines the weighted and unweighted problems and seeks to find the partition of a weighted graph that doesn't just balance the total weights of the partitions but also the total sizes of the partition. It may be straightforward to find a solution that maximizes one of those two constraints, but finding one that balances both is difficult. It's also not guaranteed that any solution that maximizes one will maximize the other.

One way of solving this is by assigning weights to the score functions and taking their sum as the overall score function. For example, suppose the goal of a given MOO problem were to maximize a set $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ of n objective functions. Let $W = \{w_1, w_2, \ldots, w_n\}$ be real values. Then the instance of MOO can be converted into a single-objective optimization problem by trying to maximize $f^*(x) = w_1 * f_1(x) + w_2 * f_2(x) + \ldots + w_n * f_n(x)$.

This approach has the upside of reducing the complexity of the problem to a well-understood one, but has the downside of requiring manual selection of the weights. This can become difficult for non-normalized (or

non-normalizable) weights, where, for example, the upper bound might be 0 but the lower bound might be $-\infty$, making comparisons between objectives for the purposes of scoring difficult. This can be resolved with enough tweening of the weights, but may take a while and requires considerable manual labor.

The most frequent approach, however, is simply to redefine the concept of optimality. In the single-objective context, an optimal state is one in which no neighboring state has a higher objective value. In the multi-objective context, an optimal state is one where no neighbor is better along every single objective axis. In other words, if $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is the set of objective functions, then a state S is optimal in the multi-objective context iff, for all neighboring states N,

$$\forall\, f \in \mathcal{F}\, f(N) \leq f(S).$$

States that obey this property are called Pareto optimal.[10] So, whereas single-objective hill climbing accepts any move for which $f(N) > f(S)$, multi-objective hill climbing can only accept moves where two conditions hold:

1. For all objective functions $f \in \mathcal{F}$, $f(N) \geq f(S)$.
2. For at least one objective function $f \in \mathcal{F}$, $f(N) > f(S)$.

A state N that satisfies both of these properties is said to dominate S. Dominance is a partial order, rather than a total order, and so it's possible for a set of states $\mathcal{S}$ to be divided into subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ such that for all $i < j$, every element of $\mathcal{S}_i$ dominates every element of $\mathcal{S}_j$, but no two elements in the same set dominate each other. $\mathcal{S}_1$, the subset whose elements

[10] In fact, states for which $f(N)$ doesn't equal $f(S)$ for all f are called strongly Pareto optimal, and states for which $f(N) = f(S)$ for all f are called weakly Pareto optimal. For reasons similar to those discussed above, though, strong Pareto optimality is generally preferred in the computational context.

dominate every other element in the state space, is also called the Pareto frontier.

The goal of MOO algorithms is often to return this Pareto frontier in an efficient manner, but doing so can be difficult. Nearly every single-objective approach, including genetic algorithms, can be adapted to the multi-objective context.

2.5 Conclusion

Gerrymandering is more feasibly interpreted as a computational problem in a discrete domain, where block-sized atoms preclude the possibility of districts that split houses. The abstraction of a graph equips us with a mathematical language to describe the problem, which is called the balanced connected partition problem. Unfortunately, BCP is NP-complete, which makes finding the best district map of a state hard to do quickly and deterministically. Search approaches, in particular hill climbing and genetic evolution, can nevertheless yield good solutions, although with a speed-quality tradeoff.

This chapter laid out at a very general level the ideas necessary to discuss our approach. The next chapter discusses our solution in detail, and assumes familiarity with both this chapter and the previous one.

Chapter 3

Evolutionary Redistricting

Chapter 1 discussed the problem of gerrymandering in detail. Chapter 2 laid out a theoretical foundation for a mathematical approach to redistricting. The rest of this paper focuses on our main technical contribution: a new system of drawing districts, rooted in graph theory and evolutionary algorithms. The model and the theory are briefly laid out in the first section, after which the various algorithms used are described in considerable detail. We also discuss some issues encountered during the implementation of our algorithm.

3.1 The Block Graph Revisited

Chapter 2 discussed the block graph in some detail, but it's useful to redefine the graph here. Formally, let B be a collection of blocks in a state, which can be realized as closed shapes that partition a polygon (the state) on a plane. Blocks b_1 and b_2 are neighbors iff their boundaries intersect at a line (such that blocks that touch only at a point are not interpreted as neighboring). A block with no direct neighbor b is interpreted as the indirect neighbor of the block b' whose centroid is closest to its own.

The block graph $G = (V, E)$ is defined as follows. Each block $b \in B$ is

assigned a vertex $v_b \in V$, and

$$E = \{(v_{b_1}, v_{b_2}) \mid b_1 \text{ and } b_2 \text{ are neighbors}\}.$$

A valid district map is a d-partition of V into d sets $P = \{V_1, V_2, \ldots, V_d\}$ such that $G[V_i]$ is connected for all i. In this paper, we define three functions that should be optimized:

1. **Maximize population equality.** Let $w(v_b)$ be the population of block b. Then for any district map $P = \{V_1, \ldots, V_d\}$,

$$PE(P) = \min_{V_i \in P} w(V_i).$$

2. **Minimize the efficiency gap.** Let $w_A(v_b)$ and $w_B(v_b)$ be votes cast for party A and party B, respectively, in block b. Let $w_T(v_b) = w_A(v_b) + w_B(v_b)$ (that is, the total number of votes cast). Assume without loss of generality that A is the winning party. Then for any district map $P = \{V_1, \ldots, V_d\}$,

$$EG(P) = \left| \sum_{V_i \in P} \sum_{v_b \in V_i} \left(w_A(v_b) - \left(\frac{w_T(v_b)}{2} + 1 \right) \right) - w_B(v_b) \right|.$$

   (Note that this assumes that elections are majority-based, and the winner needs half the votes plus one to win.)

3. **Minimize counties split.** Let $c(b)$ be the county of a block. Then for any district map $P = \{V_1, \ldots, V_d\}$,

$$CS(P) = \sum_{V_i \in P} |\{c(b) \mid b \in V_i\}|.$$

While there are other factors of district map quality that could be considered like compactness, we believe that the three factors above are sufficient

to create districts of high quality. Adding more factors would dramatically increase the difficulty of the problem, since the search space would increase by an order of magnitude.

We can realize the problem of redistricting as solving BCP for each factor. But, since it's unlikely that a maximally efficient map would have much overlap with a map maximally equal in population, for example, it would be difficult to reconcile three separate solutions. Instead, we can use local search techniques, and in particular multi-objective optimization methods, to find a map that has high scores for each balance function above. Our approach is discussed below.

3.2 The Algorithm

The framework of our algorithm is straightforward. Using memetic evolution, we can quickly find maxima for each of several balance functions. The search space is broad, consisting of every partition of vertices (regardless of whether the partition induces connected subgraphs, and in fact regardless of whether the partition is a d-partition). This enables us to encode elements of the search space as chromosomes intuitively.

3.2.1 Genetic Operators

The chromosome is a $|V|$-length array C, where $C[i]$ is the index of the subset containing vertex i. This presents the issue where multiple chromosomes can encode the same partition—for example, $C_1 = [1, 2, 3]$ represents the same 3-partition of 3 vertices as $C_2 = [3, 2, 1]$—so chromosomes are normalized, where $i < j$ implies $C[i] < C[j]$. Thus, both C_1 and C_2 would be represented as $[1, 2, 3]$.

Selection is performed using 3-way tournament selection, where each parent is chosen as the best of three randomly sampled elements of the population. This approach doesn't guarantee that the best individuals in a pop-

ulation are chosen, but this is done to ensure diversity of genetic material, as discussed above. Crossover takes two parents, p_1, and p_2, picks a random split point in the chromosome, splitting the parent chromosomes into two, such that $p_1 = \alpha\beta$ and $p_2 = \gamma\delta$, and yields children $c_1 = \alpha\delta$ and $c_2 = \gamma\beta$. Finally, mutation randomly (and with diminishing probability in successive generations) chooses a random element in the chromosome and assigns it a random district. (After mutation, the chromosome must be renormalized.)

3.2.2 NSGA-II

These operators determine how elements of the population interact with each other. In addition to this, how the population is determined between generations must be chosen carefully. The simplest approach is to replace every generation with their children wholesale, but that could remove particularly optimal parents merely for the sake of having new genes in the gene pool. In real life, that's unavoidable, but in computational evolution, we can choose to hold on to particularly good members of the population for an arbitrary number of generations.

This technique of preserving good parents at the cost of including suboptimal children is called elitism, but opens up the possibility of premature convergence. It's also computationally expensive, since comparing every element of the population to every other element for the sake of elitism takes a while. Moreover, if we add the additional complication of multi-objective optimization, the runtime can blow up quickly.

Balancing elitism with genetic diversity in the multi-objective context, without compromising runtime, is a tricky problem, and a lot of research in evolutionary algorithms focuses on this problem. Arguably, the current gold standard in the literature is NSGA-II, which stands for Nondominated Sorting Genetic Algorithm (version 2), introduced in 2002 by Deb *et al.* (see [13]). A more technical introduction to the subject is available in the original paper, but broadly speaking, the algorithm operates as follows:

1. Let P_{t-1} be the current set of parents, and let Q_{t-1} be the children of those parents. Denote $n = |P_{t-1}|$. Define the current population as $P_{t-1} \cup Q_{t-1}$.

2. Sort the current population into frontiers $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$. Define the rank of a state S as i if $S \in F_i$.[1]

3. Initialize P_t to the empty set. While $|P_t| < n$, let $P = P \cup F_i$ for $i \in [1..m]$. (Note that this may result in $|P_t| > n$.)

4. For each objective function f, sort F_k under f.[2] Calculate the distance from each state $F_k[i]$ to its neighbors, as follows:

$$d_f(S) = \begin{cases} \infty & i = 0 \text{ or } i = |F_k| - 1 \\ \dfrac{|f(F_k[i+1]) - f(F_k[i-1])|}{f^{\max} - f^{\min}} & \text{otherwise} \end{cases}$$

   ($f^{\max}$ and $f^{\min}$ are the maximal and minimal values, respectively, of f. Let $d(S) = \sum_{f \in \mathcal{F}} d_f(S)$.

5. Let $\prec$ be the partial order defined as follows: $S_1 \prec S_2$ if $\text{rank}(S_1) < \text{rank}(S_2)$, or $\text{rank}(S_1) = \text{rank}(S_2)$ and $d(S_1) > d(S_2)$. Sort P_t according to $\prec$, and let $P_t = P_t[:n]$ (that is, select the best n elements).

6. Perform selection, crossover, and mutation as normal to find Q_t.

At a very high level, what this is intended to do is choose the best elements from the parent and child populations taken together to form the new parent populations, thus in theory creating the best pool of genetic material from which to create children. Step 4 also guarantees that solution diversity is maintained, which is critical in guaranteeing that genetic material doesn't

[1] In the single-objective sense, this is equivalent to ordinary sorting.
[2] That is, order the states in F_k by their value under f.

become redundant over time. The overall runtime is $O(MN^2)$, where M is the number of objective functions and N is the size of the population.[3]

3.2.3 Local Search

After each generation is created, each child is optimized before the population is sorted by performing hill-climbing. The conversion between a genetic candidate C and a search state S is simple: let $G = (V, E)$ be the original graph (that is, before any partitioning is performed). Then the induced graph $G_C = (V, E_C)$, where $E_C = \{(i, j) \in E \mid C[i] == C[j]\}$. The set of edges removed by the partition, $H_C = E \setminus E_C$, are termed "hypothetical" edges, for reasons that will become clear.

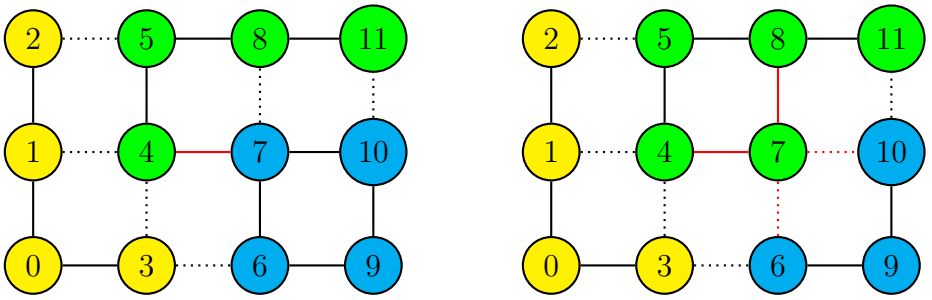
The search space is necessarily the same as the phenotypic domain (that is, the set of possible partitions chromosomes could represent). Steps in the domain can be visualized as moving vertices that lie on the border between districts from their district to the other one. The set of vertices that lie on the border are, of course, precisely the set of hypothetical edges—termed hypothetical because they represent the set of possible moves.

Steps must maintain the following invariants:

- **Invariant One.** For every intermediate graph partition G', H' must only contain edges not in G', but in G.
- **Invariant Two.** For every intermediate graph partition G' and chromosome C', for all vertex pairs i and j, $C[i] \neq C[j]$ implies (i, j) is not in G'.

Steps are made as follows:

[3] An algorithm having runtime $O(f(n))$, where n is the size of the input, means the algorithm takes a number of operations bounded by $c * f(n)$, for some $c > 0$. This is called *big-O notation*.



(a) Chromosome: [111 122 332 332] (b) Chromosome: [111 122 322 332]

Figure 3.1: Taking a step. Each vertex color is a separate district. Hypothetical edges are shown in black dotted lines. Here, the hypothetical edge $(4, 7)$ is realized by adding 7 to 4's district. 7 is disconnected from its component (thus adding $(6, 7)$ and $(7, 10)$, shown in dotted red, to the hypothetical edge set and removing them from the graph) and added to 4's component (thus adding $(4, 7)$ and $(7, 8)$, shown in solid red, to the graph and removing them from the hypothetical edge set).

1. Let (u, v) be a hypothetical edge. Then, by construction, u and v are in different districts. Let $N(i) = \{v' \in V \mid (i, v') \in E_C\}$ be the set of neighbors of a vertex i in G_C.

2. Set $C[v] = C[u]$.

3. For all vertices $w \in N(v)$, if $C[w] == C[u]$, add (w, v) to the graph; if $C[w] == C[v]$, remove (w, v) from the graph. The added edges are removed from H_C, and vice versa.

3.2.4 Handling Invalid States

Chromosomes in the evolutionary phase and moves in the hill-climbing phase aren't guaranteed to yield valid partitions—that is, d-partitions that induce d connected subgraphs. For example, in the graph in figure 3.1, if the edge $(1, 4)$ were realized, although the second district (in green) would be connected, the first district (in yellow) wouldn't. It's also possible that the number of

districts could decrease (but not increase) arbitrarily. For example, if one subset of vertices in the partition contained a single vertex, and that vertex were added to a different district, that partition would cease to exist.

To account for this, district scores receive a penalty modifier for having more or fewer than d connected components, and more or fewer than d partitions, such that if $B(S)$ is the score of a partition with c connected components and p partitions, the adjusted score $B'(S)$ becomes

$$B'(S) = B(S) - 100 * |B(S)| * |c - d| - 1000 * |B(S)| * |p - d|.$$

Since $B(S) = B'(S)$ when $c = p = d$, this both hurts invalid states and allows local search to find a better, valid alternative (which wouldn't be possible if, for example, invalid states received a score of $-\infty$).

3.3 Implementation

At a high level, our software operates as follows for each state:

1. **Getting the data.**
   (a) The state's block and block group shapefiles and demographic data are collected from the Census Bureau.
   (b) If available, the state's precinct shapefiles and voting results are collected from the Office of the Secretary of State for that state.
   (c) The state's precinct data are disaggregated to the block level by examining the geographic relationships between blocks and precincts.
2. **Constructing the block graph.**
   (a) The block graph is built by examining the geographic relationships between blocks and constructing the appropriate graph.

(b) The block graph is populated with with demographic data from the Census Bureau.

(c) The block graph is populated with electoral data from the disaggregated precinct data.

3. **Evolution.**

(a) The initial population (of size p) is generated by generating random chromosomes. Selection, crossover, and mutation are performed as normal on the first generation.

(b) The entire population is optimized via local search.

(c) The population is sorted into frontiers and ranked, and the best p elements are chosen to create the new parent population, via NSGA-II. Selection, crossover, and mutation are performed, and the resulting child population is optimized via local search.

(d) The last step is repeated until the maximum number of generations is reached.

The theory behind third step, evolution, was discussed in detail above. In implementing the system, though, we encountered several major hurdles, from data availability to computational intractability. This section focuses on those issues, detailing the sources of these problems and our solutions to each of them. We also discuss several problems to which we don't yet have a good solution and potential solutions. In addition, we discuss some results, both in test cases and in real-world examples. It should be noted that these results are incomplete, and future work will see these approaches taken to larger-scale problems than those solved within this paper.

3.3.1 Data Sourcing

The United States Census Bureau makes all data collected during the course of the decennial census available online. While the data are expansive in

their scope, they are also presented in a fragmented and somewhat confusing manner. The Bureau takes the approach that complexity is important, and that it's better to present all the data confusingly than it is to present only some of the data simply. It's worth discussing, though, how data for this project was sourced.[4]

Working with Shapefiles

The geography of the United States is described by the Census Bureau using a data format called a shapefile. Shapefiles detail polygons by their coordinates (which, in the real world, corresponds to latitude-longitude pairs), and can contain auxiliary data (like population or average rainfall) corresponding to the region described by the file. For each geographic entity tracked by the Census Bureau (see figure 3.2), shapefiles going back several years are freely available online.[5]

The Bureau also maintains a separate database of CSV (comma-separated value) files that store population data, but these data are stored in several places at once, with small differences between each source. One source[6] maintains year-2015 shapefiles joined with data from the 2011-2015 American Community Survey, which is a secondary survey performed by the Bureau on an annual basis (as opposed to the Census, which is taken every ten years). These files only track data down to the block group level, though. The same source also maintains joined data at the block level, but using data from the 2010 Census (and, although unspecified, presumably shapefiles from the same year).

Luckily, once the correct data are found, working with the files is generally fairly simple. Third-party software exists that allows users to view and analyze the shapefiles. In addition, shapefile manipulation libraries exist in

[4]It's also worth noting that this section will likely go out of date very soon, as the Decennial Census will be released in fewer than five years from the time of writing.

[5]See https://www.census.gov/cgi-bin/geo/shapefiles/index.php.

[6]See https://www.census.gov/geo/maps-data/data/tiger-data.html.

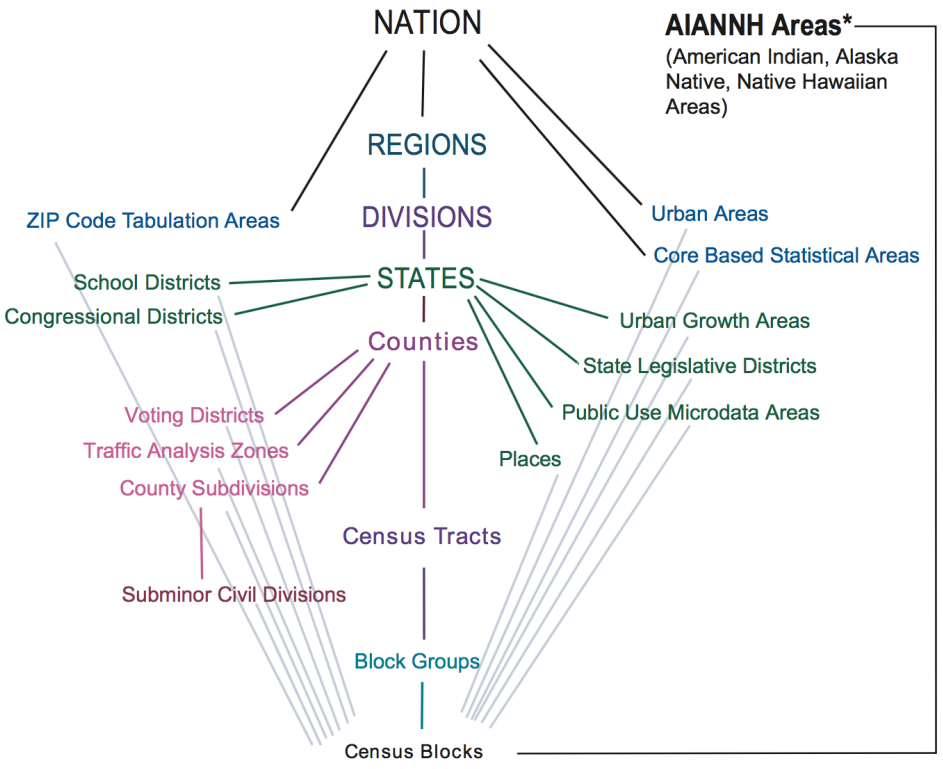


Figure 3.2: The hierarchy of Census geographic entities. Taken from the U.S. Census Bureau website.

many popular programming languages. For this project, we use Fiona[7] to read and write shapefiles, Shapely[8] to work with the spatial data (specifically, to build the block graph, as detailed below), and Descartes[9] to plot shapefile data.

[7]See https://pypi.python.org/pypi/Fiona.
[8]See https://pypi.python.org/pypi/Shapely.
[9]See https://pypi.python.org/pypi/Descartes.

Merging Voting Data

Voting data comes in several forms, each of which is safeguarded by the strongest privacy protections. All of these data are maintained not at the federal level, though, but at the state level, and often at even more granular levels than that. Every state is divided into voting precincts (also called VTDs), each of which maintains election results independently. These results are sent to the Secretary of State for each state and are tabulated to declare winners in statewide elections.

This presents two problems. For one thing, states are under no obligation to maintain or provide to the public precinct-by-precinct breakdowns of voting results. This occasionally means that data has to be collected from each precinct, many of which don't maintain simple electronic records. This can also create potential data standardization issues, since what one precinct describes as a vote for party A might not be what another uses to describe that vote. Some states do maintain state-level records, but finding these is often non-trivial.

The other problem is slightly more challenging. Precincts are arbitrary, and aside from obeying county lines don't have to obey any other geographic lines. The process of disaggregating precinct-level data to the block level is a nontrivial one. The goal is to figure out how many votes each block contributes to the statewide total, but because precincts aren't guaranteed to snap to block boundaries, blocks are sometimes contained in multiple precincts at once. Assigning those blocks a share of the vote in every precinct in which they're contained is a nontrivial problem. How this issue is resolved is discussed in the next section.

3.3.2 From Data to Graph

Building the Block Graph

Once all of the data has been found, the block graph must be constructed. Recall that the block graph contains a vertex for every block in the graph, and an edge between every pair of vertices whose blocks intersect nontrivially (that is, along more than just a point). The simplest way to do this, of course, is to compare every block to every other block and see if they intersect. But this requires $O(n^2)$ operations for n blocks, which at an average of 200,000 blocks, this would require 40 billion intersection checks, which are expensive.

Instead, consider the following process:

1. Place the n blocks into m collections, such that the blocks in each collection are connected.

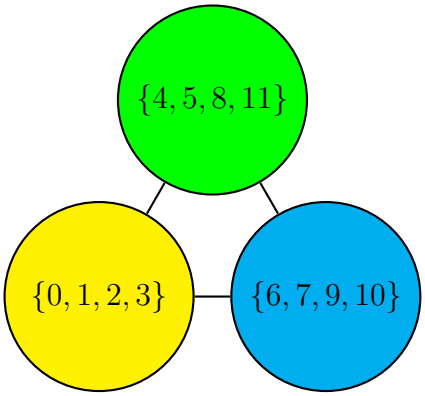
2. Build a graph of the m collections using the naïve approach, which for small m isn't too expensive.

3. For each collection, build a graph of the (on average) n/m blocks naïvely.

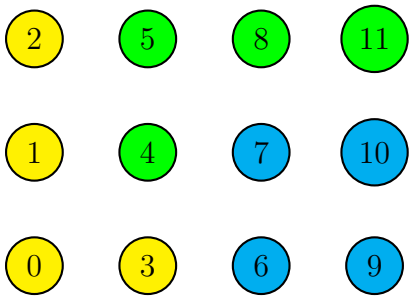
4. For each edge in the collection graph (u, v) naïvely build the edges between the blocks in u's collection and the blocks in v's collection.

(See figure 3.3.)

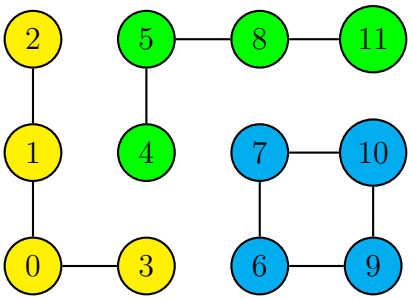
Constructing the collection graph takes $m + m^2$ operations to create the vertices and build the edges. Similarly, the block graph within each of the m collections takes $n/m+(n/m)^2$ operations to create. Finally, the comparisons between collection subgraphs takes $(n/m)^2$ operations. The number of these comparisons is equal to the number of edges in the collection graph, which, since the graph is planar, is at most $3m - 6$ by Euler's Theorem. Thus, the



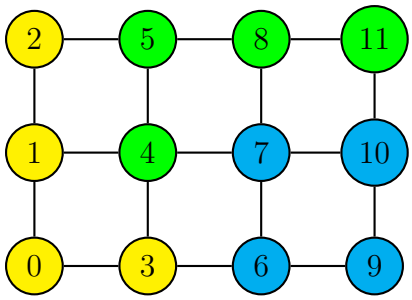
(a) Start by constructing the graph of block groups.



(b) Then, initialize the vertices of the block graph and assign them to block groups.



(c) Then, construct each block group's subgraph.



(d) Finish by constructing the edges between connected block groups.

Figure 3.3: Building the block graph.

total number of operations is

$$m + m^2 + m\left(\frac{n}{m} + \frac{n^2}{m^2}\right) + (3m - 6)\left(\frac{n^2}{m^2}\right) = m + m^2 + n + (4m - 6)\frac{n^2}{m^2}.$$

By taking the derivative of this equation with respect to m, setting it equal to 0, and solving for m, for $n = 200{,}000$, we get that the optimal value of m is 4307. As it happens, there are around 4270 Census block groups per state in the United States. Furthermore, finding the block group that contains a block is trivial. Every block is assigned an ID number (called a

GEOID), which is the ID of its block group, plus the index of the block in the block group, so going from block to block group is a matter of removing the last three digits from the ID of the block. Thus, by building a block group graph first, then using the above algorithm to build the block graph, we can reduce the number of operations from 40 billion to 55 million.

Mapping Precincts to Blocks

A few different approaches can be used to resolve the problem of precinct disaggregation, introduced in the previous section. (See [38] for a breakdown of each.) The simplest method, which only requires the boundaries of the precincts in a state, takes the following approach. Suppose that a precinct p with area a and v votes cast intersects n blocks $b_1, \ldots, b_n$, in regions with area $a_1, \ldots, a_n$. Then the fraction of v assigned to block b_i is given by a_i/a. This makes the assumption that populations in each precinct are uniformly distributed, which is unlikely to be the case. In the absence of more specific data, however, this is the best possible option.[10]

Even with this approach, disaggregation is an expensive process. Naïvely comparing every precinct to every block requires a number of operations proportional to the product of the number of precincts and the number of blocks. A state with 200,000 blocks and 5,000 precincts (although these numbers can vary widely) requires a billion intersection checks, which can be expensive. We can improve on this approach somewhat by borrowing ideas from our construction of the block graph. Our disaggregation approach finds every block group-precinct overlap, thus reducing the number of block-precinct comparisons by only having to compare blocks in a block group to precincts that overlap that block group.

[10] [38] suggests an alternative approach that involves using a voter registration database, geolocating for each voter their address and assigning the block in which that address is contained to the precinct assigned to that voter in the database. However, voter registration data aren't always easily available, and geolocation of the millions of addresses contained in the average database is infeasible.

3.3.3 Evolution: Encoding the Chromosome

The theory behind the evolutionary algorithms discussed above is not particularly complicated, but while most of the genetic operators are very straightforward, the choice of chromosome encoding is not, and merits further discussion. Graph partitions are represented very simply by the chromosome in the evolutionary phase. Each partition is assigned a value between 1 and d, and if vertex i is in partition j, then the chromosome C has $C[i] = j$. Reconstructing the graph (i.e., the phenotype corresponding to the genotype expressed in the chromosome) is straightforward, as the graph is merely the composition of the subgraphs induced by each partition. These partitions can be found in $O(|V|)$ time, and so the graph can be built in $O(|V| + |E|)$ total time. (Since, as mentioned above, the graph is planar and therefore $|E| \leq 3|V| - 6$, this is actually $O(|V|)$.)

As mentioned above, though, this chromosome allows for *any* vertex partition to be a "valid" element of the search space, even if the subsets don't induce connected subgraphs. In addition, although as $|V|$ increases this becomes more unlikely, it's possible for a chromosome to encode fewer than d partitions if, for example, on initialization the random integer array contains fewer than d unique elements, or during crossover and mutation some partition indices disappear.

Both of these issues can be handled by penalizing partitions as discussed above, but penalization is expensive; in particular, finding the number of connected components in a graph takes $O(|V| + |E|)$ time naïvely. There are ways to speed this time up, as discussed below. We initially avoided this issue, however, by changing the representation of the partition.

In our initial attempt, the chromosome was an $|E|$-length array. Each edge e was assigned a priority $p_e \in [0, 1]$. To reconstruct the graph, we added edges into graph in order of their priority. By using the union-find data structure, which can be used to track connected components in effec-

tively constant time[11], we could add edges until the number of connected components equaled d. Thus, the graph was guaranteed to have d connected components.

There were two issues with this approach. The first was that while the chromosome could easily be converted into a graph, the reverse was not easily doable. Since a graph has no implicit ordering on the edges it contains, it's not immediately clear that there's a deterministic conversion from edge into priority. Genetic algorithms function best when as much is deterministic as possible, so optimizing a chromosome via hill-climbing may have actually made the process less likely to reach an optimal candidate.

When we removed optimization, we found that the algorithm converged very quickly to a suboptimal candidate. (The utility of the final output was less than half the utility of the final output of a very simple and inefficient implementation of the vertex-based chromosome approach.) There are many possible reasons for this, and substantive discussion of the suboptimality of one approach of evolution compared to another is beyond the scope of this paper.

However, we conjecture that the reason behind the inefficiency is simple. In human DNA, a nontrivial percentage of the base pairs has no direct impact on protein construction. Instead, it's theorized that these pairs have significance (so to speak) in epigenetic interactions and in ways that aren't immediately clear in the phenotype. This DNA is called "junk DNA". Put simply, we believe that the reason the edge-based encoding as described above failed to act efficiently is because most of it is junk. (See, for example, [23] for a discussion of junk DNA.)

Every edge that connects two disconnected components decreases the number of connected components by one. To get from $|V|$ to d connected

[11]The disjoint-set data structure takes $O(\alpha(n))$ time, where $\alpha(n)$ is the inverse of the Ackermann function. The Ackermann function $A(n)$ grows so fast that $A(5)$ is a number larger than can be expressed by the physical universe. Thus, $\alpha(n) < 5$ for all practical input sizes, giving the disjoint-set data structure near-constant time.

components, then, only $|V| - d$ edges are needed (at the very minimum). In a planar graph, though, there can be as many as $3|V| - 6$ edges, which means that in the very worst case there are $3|V| - 6 - (|V| - d) = 2|V| + (d - 6)$ edges aren't included in the graph (and thus in the fitness evaluation), and are thus effectively wasted.

It's worth studying why this approach failed to work in more detail, and one potential avenue of future work is exploring alternative chromosome encodings that make the graph operations simpler (as the edge encoding would have done). For now, though, districts are still drawn using the vertex encoding.

3.3.4 Optimizing Hill-Climbing

Several important optimizations are required to make hill-climbing efficient enough to succeed at scale. Generally, hill-climbing continues until there are no more neighboring states that dominate the current state. However, given that optimization occurs in every generation for every child, and given that for large graphs the number of possible neighbor states that must be evaluated is significant, this quickly becomes intractable. This process can be made more efficient, albeit at a cost to performance of the algorithm, by capping two parameters: the number of neighbors evaluated in each step (n), and the number of steps taken (s).

We also cache states during evaluation, as follows. Let $\{S_1, \ldots, S_s\}$ be the states encountered during s-step hill-climbing from a starting state S_0. Each S_i, $0 \leq i \leq s$, is mapped to S_s, such that if during the optimization of a different state S' some S_i is encountered, the algorithm can immediately terminate by returning S_s in $O(1)$ time. (Note that this approach doesn't guarantee that s steps are *always* taken. If $S' = S_i$ for some i—that is, the output of the genetic phase is S_i—then only one "step" will be taken and S_s will be returned.)

3.3.5 Tracking Connected Components

As mentioned above, counting the number of connected components in a graph is done naïvely using breadth-first search in $O(|V| + |E|)$. Without getting into too much detail, the simple algorithm enumerates connected components by starting at a random point, adding all of its neighbors to its connected component, going to each of their neighbors and adding them to the same connected component, and so on until there are no more neighboring vertices. If there are still unexplored vertices in the graph, the algorithm repeats the process at one of them, and so on, until there are no more unexplored vertices.

This isn't too bad for a single query. But, since the number of connected components must be calculated every time a new candidate is created in evolution and every time a step is evaluated (regardless of whether or not it's taken) in hill-climbing, redoing this operation each time can get very expensive. (Indeed, calculating the number of connected components takes up almost 99.9% of the total computation time in the naïve implementation.) However, there has been a significant amount of work done in tracking the number of connected components as edges are inserted and deleted in a graph, yielding significantly faster algorithms.

As mentioned below, this work is beyond the scope of this paper. Our approach simply caps the number of steps taken for each child candidate at 20, and the number of moves examined at 50. This reduces runtime, albeit at a tradeoff with solution quality.

Related Work

For general graphs, Holm *et al.* (see [12]) use a structure called Euler-tour (or ET) trees, created by Henzinger and King (see [9]), to support arbitrary edge insertions and deletions. These operations can be performed in $O(\log^2 n)$ time. Furthermore, connectivity queries (i.e., "is there a path between u and v?") can be answered in $O(\log n / \log \log n)$ time. Eppstein *et al.* (see [6])

give a structure that supports insert, delete, and connectivity operations in $O(\log n)$ time for *plane* graphs, which are planar graphs whose embeddings in the plane are fixed (for example by specifying Cartesian coordinates to each vertex).

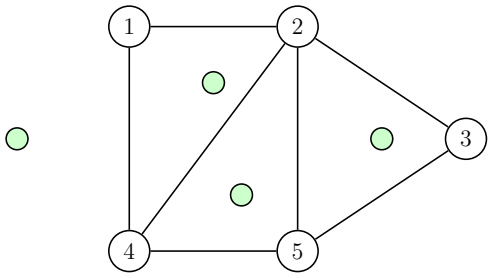
Here, though, we have a simpler problem to solve than the one answered by Eppstein *et al.* Their work supports arbitrary edge insertion and deletion; that is, so long as the graph remains planar, any two vertices can be connected in the graph. Here, though, we know at the beginning the total structure of the graph—in other words, we know every edge that will ever be added or removed from the graph. We conjecture that it's possible to improve on the $O(\log n)$ result of Eppstein *et al.* by exploiting this fact to do some precomputation on the graph; what exactly that precomputation may be is unknown.

Some Computational Geometry

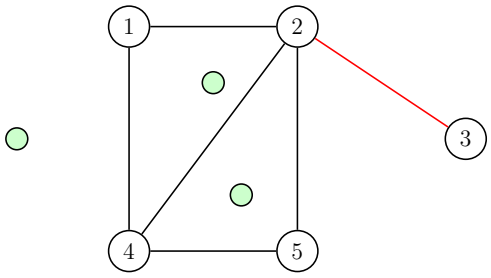
There's another approach that's possible here, similar to the one developed by Eppstein *et al.*, that exploits the planarity of the graph to track connected components. Plane graphs (but not planar graphs) induce *faces*, which are regions on the plane bounded by edges in the graph. Plane graphs also contain a single unbounded face f^* that consists of the plane, minus the closed faces. (See figure 3.4a.) Faces have a convenient relationship to connected components. A *bridge* edge in a graph is an edge whose removal would increase the number of connected components in a graph. In other words, (u, v) is a bridge in a graph if, in the graph $G = (V, E)$, u and v are connected, but in $G' = (V, E \setminus \{(u, v)\})$ (i.e., the graph without the edge (u, v)), u and v are no longer connected. (See figure 3.4b.)

Every edge in a plane graph is incident to (i.e., borders) at most two faces, one on each side of the edge. These faces don't have to be unique, so that both sides of the edge border the same face. The edges for which both incident faces are the same are precisely the set of bridge edges. To see this,

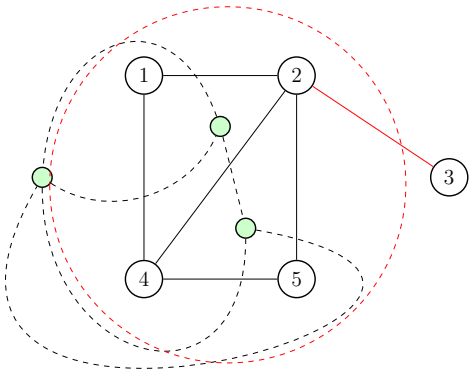
suppose there were an edge $e = (u, v)$, both of whose sides were incident to the same face, and assume by contradiction that u and v are still connected after removing e. Then there is some path from u to v that doesn't contain e. It can be seen that adding e to this path creates a cycle (since the path would then travel from u to v). This cycle creates a closed face, on one side of e, separate from the face incident to the other side of e. This is a contradiction, however, and thus e must be a bridge. (See [21].)



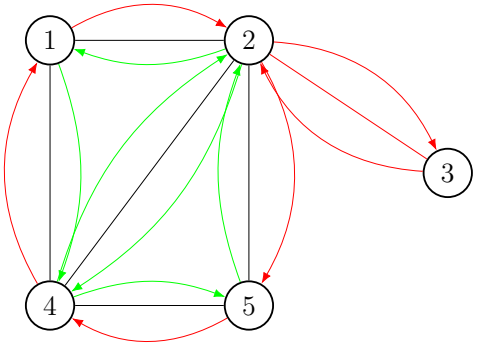
(a) This plane graph has four faces, the vertices in green.



(b) Here, the red edge is a bridge edge, since its removal would disconnect vertex 4 from the other three vertices.



(c) The dual graph. Note that the dual of $(2, 3)$ is a self-loop, which implies that $(2, 3)$ is a bridge.



(d) The half-edge data structure. Red half-edges enclose the open face, and green edges enclose the closed face. Note that the bridge edge has two red half-edges.

Figure 3.4: Plane graphs, the dual graph, and the half-edge data structure.

Bridge edges increase the number of connected components by one on deletion, and decrease the number by one upon insertion. We can thus track the number of connected components in a dynamic plane graph by testing if added and deleted edges are bridges. Whether these edges are bridges or not can be determined by testing if the edge is incident to the same face on both sides. There are several ways of doing this. One way is via the dual graph (see figure 3.4c), which is the graph $G^* = (V^*, E^*)$, where each vertex $v^* \in V^*$ represents a face in G, and each edge $e^* = (u^*, v^*)$ connects two faces when separated by the edge e (so that $u^* = v^*$ if an edge is incident to the same face on both sides). Another way is via the half-edge data structure (see figure 3.4d), where every edge $e = (u, v)$ is assigned two directed half-edges, $u \to v$ and $v \to u$, such that each half-edge is incident to only one face, and such that the half-edges incident to a face form a cycle that delineates that face.

It's possible that by initializing one (or both) of these data structures upon building the block graph, we can precompute which edges function as bridges under different graph partitions. This work, however, is beyond the scope of this work. As discussed below, however, counting connected components is among the most expensive parts of the entire algorithm, exploding the runtime. Future work will address this by trying to use one of the algorithms mentioned above to reduce the complexity of local search.

3.4 Conclusion

This section introduced a novel framework for solving the balanced connected partition problem with multiple balance functions. The approach is theoretically simple, encoding graph partitions as vertex-to-component index mappings. Genetic evolution is performed using these mappings as chromosomes and operating under the NSGA-II framework with 3-way tournament selection, one-point crossover, and one-point mutation. Children in each gen-

eration are optimized via 20-step hill-climbing. The next chapter discusses our results in detail, as well as future directions for this work.

Chapter 4

Results and Future Work

...intro comments go here...

4.1 Experimental Setup

All code was written in Python 3.5.2. Evaluation code (see Section 4.2) was evaluated on an Intel(R) Xeon(R) CPU E3-1270 v3 @ 3.50GHz running Ubuntu 16.04.3 LTS (Xenial Xerus). State-level code (see Section 4.3) was evaluated on Mastodon, the University of Texas Department of Computer Science high-throughput computing cluster[1]. Shapefiles were read in with `fiona` and analyzed with `shapely`. Graphs were processed entirely in `networkx`. `matplotlib` was used for graphs, and `descartes` was used to plot shapefiles.

[1]More information on specific machine hardware is available at `https://www.cs.utexas.edu/facilities/documentation/condor`.

4.2 Evaluation on Toy Graphs

4.2.1 Single-Objective Optimization

We first tested our implementation on $n \times n$ grid graphs. These graphs were small ($5 \leq n \leq 15$) and were initialized with randomly generated weights between 1 and 50. For all n, the number of partitions, d, was set to 5. The number of generations, g, was set to 100, and for $n > 6$, the population size p was set to 50.

The first round of evaluation was performed with a single weight on each vertex, reducing the problem to single-objective optimization. For each n, the algorithm was run once without, and once with, hill-climbing optimization, to compare performance. The number of steps taken was fixed at 20, and the number of moves sampled was set to 50. The utility function, or B-score, calculated the minimum weight sum among districts in the partition.

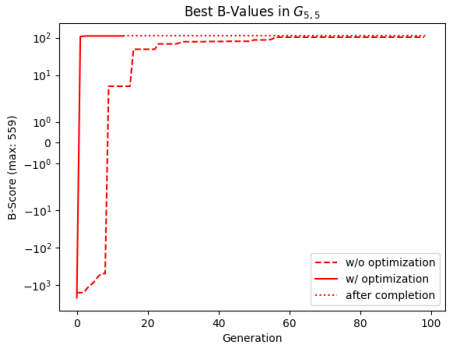
			Unoptimized		Optimized	
Graph	T	B^*	Best Score	Total Time	Best Score	Total Time
$G_{5,5}$	559	111	103	n/a	111	n/a
$G_{6,6}$	868	173	136	n/a	173	n/a
$G_{7,7}$	1346	269	-798	0:16	268	6:05
$G_{8,8}$	1521	304	-2396	0:20	302	11:28
$G_{9,9}$	2300	460	-3992	0:24	458	16:06
$G_{10,10}$	2424	484	-5394	0:28	484	18:19
$G_{11,11}$	2922	584	-5397	0:32	584	27:01[2]

Table 4.1: Best score and total runtime after 100 generations. Runtime data wasn't collected for $n = 5$ or $n = 6$, as the population size was larger (300 instead of 50). Note that the best possible score, B^*, may not always be achievable, since not all partitions are valid.

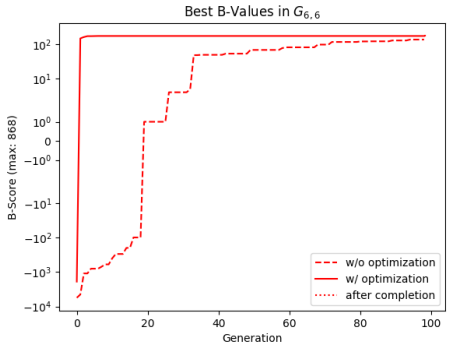
The difference in performance is staggering: for most graphs, after 100 generations, the best unoptimized solution had a negative B-score (that is, the number of connected components differed from the goal), while typically the optimized algorithm reached a nearly-optimal solution after fewer than

10 generations. Table 4.1 contains the results of this computation for different n. Figure 4.1 contains graphs of the B-score versus generation for the unoptimized (dashed line) and optimized (solid line) algorithms. Moreover, the optimized algorithm generally achieves the maximum possible score.

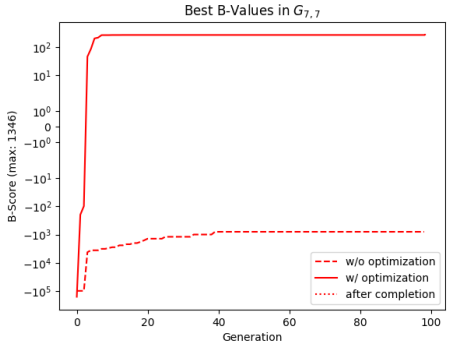
Of course, as Table 4.1 shows, the efficiency dropoff, even for small graphs, is steep. The optimized algorithm took at least an order of magnitude more time to perform hill-climbing, even with the severe limits on number of steps taken and number of potential moves evaluated. On the evaluation machine, each step in $G_{11,11}$ took 60 milliseconds to run, but for 75 generations, with a population size of 50, and a maximum of 20 steps per child, hill-climbing took a total of 26.2 minutes.



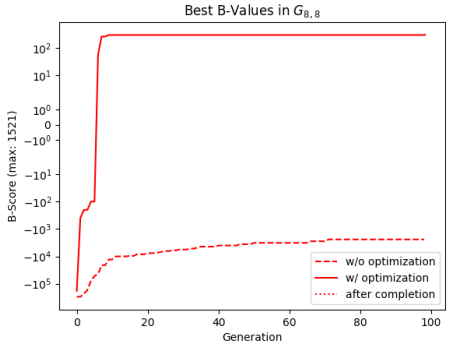
(a) $n = 5, T = 559$



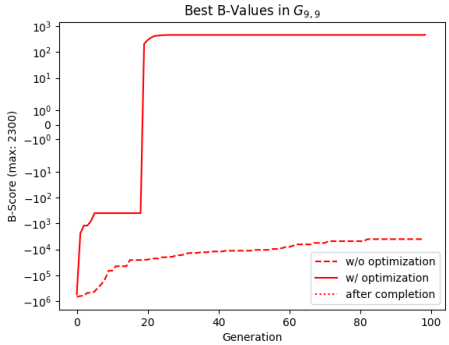
(b) $n = 6, T = 868$



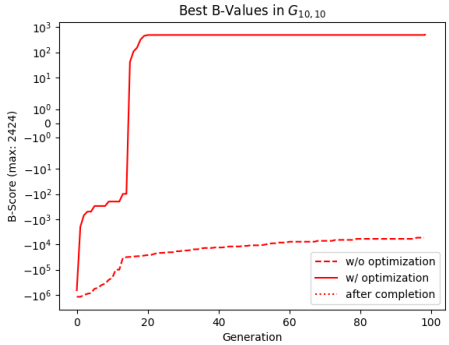
(c) $n = 7, T = 1346$



(d) $n = 8, T = 1521$



(e) $n = 9, T = 2300$



(f) $n = 10, T = 2424$

Figure 4.1: Running on graphs of size $n \times n$ for 100 generations. The total weight sum T is also given, such that the maximum possible score is $T/5$.

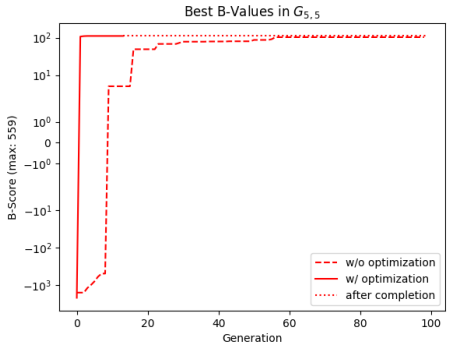
4.2.2 Multi-Objective Optimization

The evaluation framework doesn't differ much in the multi-objective model, except that every vertex has two random weights instead of one. The balance functions become B_a and B_b, where B_i gives the minimum weight sum among subsets in the partition. The major change in evaluation comes from the fact that it is in general not guaranteed that a solution that maximizes both objective functions exists, and so instead the best possible result from evaluation is the Pareto frontier of solutions that are all "as good" as each other. The number of generations was still fixed at $g = 100$, and the population size remained at $p = 50$. The number of districts also remained fixed at $d = 5$.

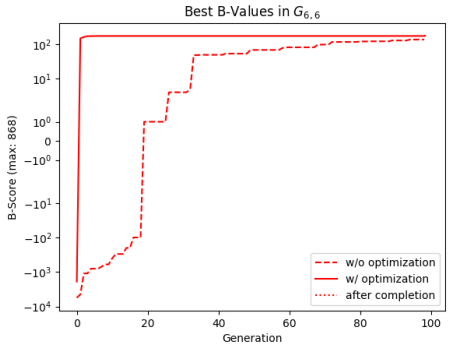
			Unoptimized		Optimized	
Graph	T	B^*	Best Score	Total Time	Best Score	Total Time
$G_{5,5}$	559	111	103	n/a	111	n/a
$G_{6,6}$	868	173	136	n/a	173	n/a
$G_{7,7}$	1346	269	-798	0:16	268	6:05
$G_{8,8}$	1521	304	-2396	0:20	302	11:28
$G_{9,9}$	2300	460	-3992	0:24	458	16:06
$G_{10,10}$	2424	484	-5394	0:28	484	18:19
$G_{11,11}$	2922	584	-5397	0:32	584	27:01[3]

Table 4.2: Best score and total runtime after 100 generations. Runtime data wasn't collected for $n = 5$ or $n = 6$, as the population size was larger (300 instead of 50). Note that the best possible score, B^*, may not always be achievable, since not all partitions are valid.

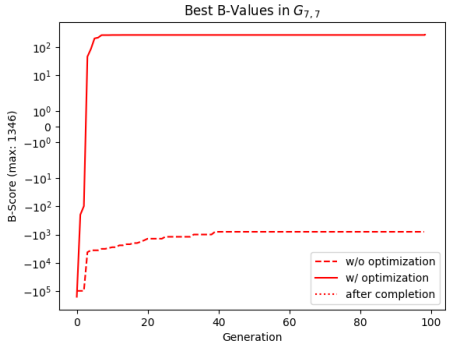
Figure 4.2 shows the Pareto frontier after 100 generations with a population size of 50. Here again, the performance of the unoptimized algorithm was far outstripped by the performance of the optimized one, albeit at the same cost to efficiency. (See Table 4.2 for details.)



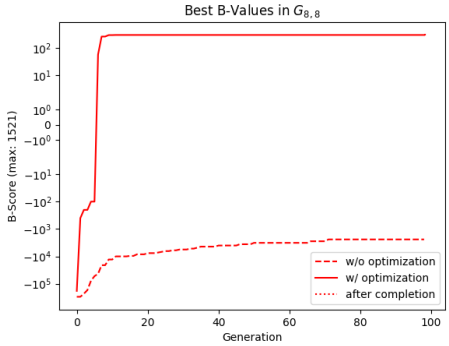
(a) $n = 5, T = 559$



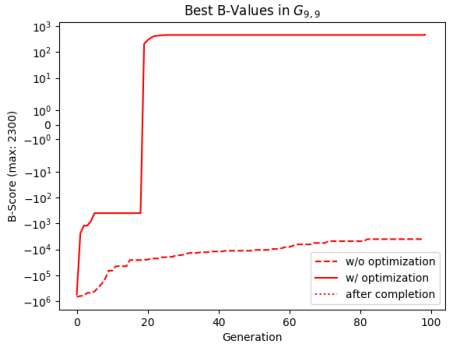
(b) $n = 6, T = 868$



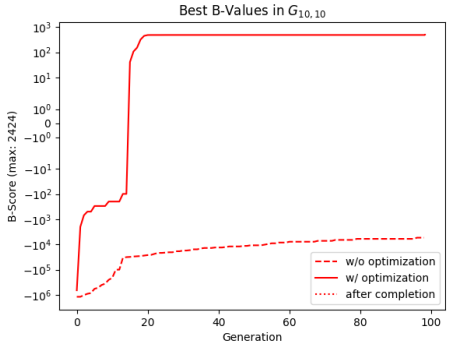
(c) $n = 7, T = 1346$



(d) $n = 8, T = 1521$



(e) $n = 9, T = 2300$



(f) $n = 10, T = 2424$

Figure 4.2: Running on graphs of size $n \times n$ for 100 generations. The total weight sum T is also given, such that the maximum possible score is $T/5$.

4.3 Evaluation at the State Level

The real test of our algorithm is, of course, in partitioning a state into districts, a far more involved calculation than evaluation on toy graphs. We tested our solution on the state of Washington, for two reasons: quality of data and size. As mentioned above, data availability can vary widely from state to state, as different Secretaries of State maintain different standards of election data collection. Washington maintains extraordinarily complete and easily accessible election returns, broken down at the county and precinct level, for use in research. Furthermore, Washington is reasonably sized, consisting of 195,000 blocks, which is fairly close to the national average of 217,000.

Unfortunately, Washington confronts the would-be gerrymanderer with a handful of problems. Washington is geographically diverse, with mountains and bodies of water that can divide districts into multiple parts, none of which is accessible to any of the others. Accounting for geography is in general nontrivial, since shapefiles don't discriminate between neighboring blocks whose border can be crossed and "neighboring" blocks that are in fact separated by impassable terrain. Moreover, Washington, although admittedly to no further a degree than many other states, contains a population that's spread out widely through the state. Almost a tenth of the population lives in Seattle alone, for example.[4]

Level	$\|V\|$	$\|E\|$	Pop. Mean (Std. Dev.)
County	39	92	186872 (379608.4)
Block Group	4783	13338	1461 (676.7)
Block	195574	480271	34 (82.3)

Table 4.3: The size of the adjacency graph at each granularity, along with the mean and standard deviation of the population in each unit.

[4]Seattle is home to 704,000 people, whereas Washington state is home to 7.3 million, according to the Census Bureau.

Evaluation at the state level is possible at a number of granularities: at the county level, at the block group level, and at the block level. The size of the graph varies significantly between levels. (See Table 4.3.) Because the standard deviation of population decreases at each granularity, the optimality of the best solution should increase as the input graph becomes more specific. (That is, the more similarly-populated units in each graph are, the easier it is to create partitions of equal size, since partitioning a graph where the population of each unit is the same is equivalent to the easier problem of 1-BCP.)

We started with the single-objective case, where the only score maximized was population equality. The county graph was easily partitioned, given its small size. After 100 generations, the optimal solution had a population equality score of 109,398, less than a third of the goal score. (See Figure 4.3.) This is because in Washington, the desired district population is 364,400, but four counties have more than that number of residents. In fact, the largest county has 2.2 million residents, making creating district maps with a population difference of less than 1% between the smallest and largest districts impossible.

The *denouement* of most papers on computational gerrymandering tends to be a picture of a state redistricted with the algorithm proposed in the paper. Unfortunately, except for the highly suboptimal county-level graph, we were unable to create such an image, as expanding to larger graphs proved intractable. The expected runtime for the optimized algorithm, with $p = 50$ and $g = 100$ as before, was on the order of a week for the block group graph. (We weren't able to calculate expected runtime for the block graph, which is two orders of magnitude larger in size than the block group graph.)

Multi-objective optimization at the state level is currently in progress, with the primary roadblock being data quality. The precinct shapefiles, which are standardized at the state level, are not by default valid, which means some additional computation is required to convert the files as they are to valid

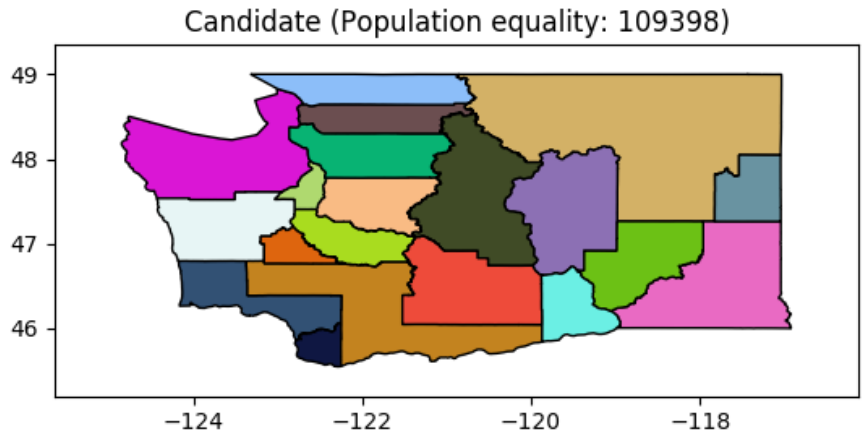


Figure 4.3: The optimal partition of Washington state at the county level.

input data for unit to precinct association.

4.4 Inefficiencies

In the context of high-performance computing, even a graph with 200,000 vertices is not a particularly large input graph. The fact that evaluation takes minutes on toy graphs is disconcerting, and suggests that there are deeper ineffeciencies that need to be addressed. These inefficiencies are varied, but probably the most serious and costly one is that Python itself is a poor language for performance-intensive computing.

Python is a great language for rapid deployment. That is, it doesn't take much time to go from an idea to functioning Python code. The language paradigms are so simple that Python is often compared to pseudocode. Of course, all of this comes at a hit to performance. The programmer doesn't have to specify types, which makes coding easier than in a language where the types of variables have to be specified, but also makes looking up a variable (the under-the-hood process of matching a variable name to the data it holds)

more expensive.

There are, of course, ways of optimizing Python to improve performance somewhat. One approach, called Cython, allows programs in Python to call expensive methods in C, a much more performant language, thus saving on overall runtime. Another is to abandon the standard interpreter, CPython, in favor of an interpreter with better performance, like PyPy. (CPython was built as a reference implementation of a Python interpreter, and wasn't necessarily built with speed as the primary goal.)

Another solution to this would be to rewrite the entire project in a different, more performant language like C++ or Golang. But the advantage of Python isn't just in the fact that it's easy to deploy applications—it's in its excellent third-party library database. We made extensive use of Python libraries in our work, as described above—the bulk of the graph algorithms were handled by the `networkx` library—some of which don't have clear analogues on other languages.

Of course, using libraries comes with the downside that it's fairly nontrivial to make library code faster; for example, `networkx` would be able to count connected components (one of the most time-consuming parts of the algorithm) faster if that particular method were written in C and called via Cython. But, aside from manually editing the source code, which can get messy, there isn't an easy way to resolve this.

Python remains, however, good for small input sizes, and a graph with 5,000 vertices is, although perhaps not trivially small, not exceptionally large either. In theory, Python should be able to be somewhat performant on graphs of that scale—and yet it doesn't. Part of this is in the implementation. Python's simplicity is deceptive, and straightforward-looking code can often be dangerously non-performant. All the line profiling[5] in the world wouldn't reveal subtle inefficiencies.

[5]Line profiling refers to the process of timing each line of code with the purpose of determining which lines take up the most computation time.

Dividing the workload between multiple processors would also, in theory, serve to reduce runtime. Anecdotally, however, switching to pool-based execution, using all available cores, didn't help that much with improving runtime, and in many cases actually ended up taking more time (perhaps because of the overhead of having to spin up new processes and combine the results from each subprocess).

Moreover, as discussed briefly above, there are several parts of the algorithm itself that are inefficient. The most obvious example is in counting the number of connected components in the graph during utility evaluation, but there are others. For example, maintaining the set of hypothetical edges efficiently is in practice if not in theory complicated, as is caching previously-encountered chromosomes during evolution and states during hill-climbing.

Some of the practical complexity comes from the opacity of the language, which renders innocuous dangerously inefficient processes, but that is only part of it. Some processes are not merely practically but also theoretically inefficient (like, as mentioned above, counting connected components, which is $O(|V| + |E|)$). Finding better ways of solving those problems is a far more difficult hurdle to cross.

Finally, the performance tradeoff only became really apparent during the last few weeks of work, as the majority of our testing was on small graphs for debugging purposes. This hindered our ability to explore any of the potential fixes to the inefficiencies in our codebase, but more importantly it prevented us from exploring potential ways of taking advantage of alternative computing environments, like the systems available through the Texas Advanced Computing Cluster[6] that might have improved our computational speed. The overhead of adapting our code for execution on those machines was too significant to be manageable given our limited timeframe.

As discussed below, fixing the latent inefficiency of our code is of crucial importance to any and all future work. The advantage is that these fixes

[6]See `tacc.utexas.edu`.

are relatively simple—except perhaps those which require a large amount of theory—and many could be fixed merely by switching to a different language. The theory remains sound, and although the implementation still has fault lines, these are on the whole manageable.

4.5 Future Work

Throughout this paper, we've discussed various potential ways of improving on and continuing our work. These avenues fall into two categories: finishing execution of our algorithm, and improving on the algorithm itself.

4.5.1 Finishing Our Work

Our algorithm is structurally sound—the optimality of evaluation on toy graphs proves that point. So, for that matter, is the majority of our codebase. Two things need to happen for this work to be truly completed, however:

1. **Finish execution.** At time of writing, our algorithm is partly through partitioning the block group graph of Washington state, with the goal of maximizing population equality alone. This process needs to be completed, and the results should be made available.

2. **Fix and merge voting data.** As mentioned above, there are some issues with the precinct data that require some work to fix. For example, the shapefiles provided by the Office of the Washington Secretary of State aren't completely valid, in that some of the shapes they describe contain self-intersections. Furthermore, precincts are specified by the county in which they're drawn, but precincts often don't completely overlap with the county. (It's still relatively high, with the lowest area of overlap encountered at 94% of the precinct area.) These anomalies will require careful thought as to how they can be fixed.

Once both of these tasks are completed, we can attempt execution on the larger block graph, to get an optimal, granular district map.

4.5.2 Improving Our Algorithm

There are several ways to improve the algorithm and our implementation:

1. **Switch to a more performant language.** As mentioned above, Python isn't very efficient, and this inefficiency has led to the long computation times we've highlighted in this paper. Switching to a more performant language like C++ or Golang would solve this.

2. **Consider alternative genetic operators.** The genetic operators used in this paper—three-way tournament selection, one-point crossover, and one-point mutation—were chosen because of their simplicity and because of a seeming consensus in the literature of their optimality for graph algorithms. There are, however, alternative, more complicated approaches, that might yield better performance—perhaps even removing the need for local-search optimization.

3. **Choose alternative chromosome encoding schemes.** As described above, we experimented with an edge-ordering chromosome that yielded highly suboptimal performance. Our transition to a vertex-based chromosome was done mostly because that encoding was straightforward to implement. But there is a vast array of ways in which graph partitions can be represented. Future work could focus on testing different encoding schemes for performance.

4. **Implement more efficient graph algorithms.** Our work uses the default implementations of most graph algorithms provided by the *networkx* library. These are in general fairly performant, but because the graph in our work is fixed, there are optimizations that can be taken advantage of—for example, as mentioned above, we can exploit the

planarity of the graph to use more efficient algorithms for tracking connected components.

5. **Explore alternative local search strategies.** Genetic evolution and hill-climbing are not the only ways of exploring a combinatoric search space. There are others, like simulated annealing, which allows for occasional downward movement with diminishing probability in hill-climbing, that may yield more optimal solutions. Our approach to memetic evolution also performs evolution every generation, but it may be more efficient to perform local search only once every 10 generations, for example. There are also alternatives to NSGA-II for population management that might yield more diverse solutions.

6. **Experiment with different utility functions.** Our work uses three fairly basic utility functions, but there are many more that should be considered. For example, it may be useful to consider the compactness of districts, which, although difficult to achieve while balancing minority interests when districts are drawn by hand, could be achievable using a computer. Other possible utility functions include diversity of population in each district (by some combination of demographic factors), or a more complex efficiency gap measurement that measures partisan lean by primary voting.

These are a few potential approaches to improving our work. There are others. Much of the work that needs to be done to improve solution will require additional background research into genetic theory and artificial intelligence beyond the scope of this paper.

4.6 Conclusion

...concluding remarks go here...

Conclusion

Gerrymandering is, without a doubt, one of the most dangerous practices carried on in the United States today. Because of its subtlety, it's hard to measure, hard to attack, and hard to fix. The Supreme Court has declined to act against it, and in fact has weakened protections against the practice. In 2004, they came the closest yet to outlawing the practice, but Kennedy decided not to side with the liberals and instead, as is his practice, chart out a middle ground that left open the possibility of a future challenge, if a standard for measuring gerrymandering could be found. In June of next year, when the Court hands down their opinion in *Gill v. Whitford*, Kennedy may decide that the efficiency gap is precisely that standard, and strike down gerrymandering as unconstitutional once and for all.

If he does so, the vast majority of states will have to rethink the way they handle redistricting. There are several possibilities for how they may do so, but none is more promising than by computer, taking advantage of the unprecedented scale at which data are available that would enable drawing fairer, more representative voting districts. And, while there are a handful of examples scattered throughout the literature of algorithms that draw districts by computer, they mostly focus on the simplest challenge—creating districts equal in population.

This paper goes beyond that. Our algorithm enables drawing districts that are optimal with respect to an arbitrary number of objective functions. Although execution is still pending at the state level, the results are promising

on small inputs—in fact, within only a handful of generations, our algorithm finds partitions that are optimal along every objective function. While there is much that can be done to improve on these results, the results described in this paper provide a solid foundation for future work.

It remains to be seen if the Court will strike down gerrymandering and end a practice that has subverted the will of political minorities for over two hundred years. Even if they don't, the rise of independent, expert redistricting commissions around the country is promising. But the only reliable solution to redistricting is an algorithmic one. Perhaps with such an approach, coupled with leadership at the state and federal level, we may find ourselves at the end of the era of gerrymandering, and the beginning of an era of truly fair representation.

Bibliography

[1] William Vickrey. "On the Prevention of Gerrymandering". In: *Political Science Quarterly* 76.1 (1961), pp. 105–110. ISSN: 00323195. URL: http://www.jstor.org/stable/2145973.

[2] L. Lovász. "A homology theory for spanning tress of a graph". In: *Acta Mathematica Hungarica* 30.3-4 (1977), pp. 241–251. DOI: 10.1007/BF01896190.

[3] E. Győri. "On division of graphs to connected subgraphs". In: *Combinatorics*. Ed. by A. Hajnal and VT. Sós. Amsterdam ; Oxford ; New York: North-Holland Publ. Comp, 1978, pp. 485–494. ISBN: 9638021020.

[4] Yehoshua Perl and Stephen R. Schach. "Max-Min Tree Partitioning". In: *J. ACM* 28.1 (Jan. 1981), pp. 5–15. ISSN: 0004-5411. DOI: 10.1145/322234.322236. URL: http://doi.acm.org/10.1145/322234.322236.

[5] M.E. Dyer and A.M. Frieze. "On the complexity of partitioning graphs into connected subgraphs". In: *Discrete Applied Mathematics* 10.2 (1985), pp. 139–153. ISSN: 0166-218X. DOI: http://dx.doi.org/10.1016/0166-218X(85)90008-3. URL: http://www.sciencedirect.com/science/article/pii/0166218X85900083.

[6] David Eppstein et al. "Maintenance of a minimum spanning forest in a dynamic plane graph". In: *Journal of Algorithms* 13.1 (1992), pp. 33–54. ISSN: 0196-6774. DOI: https://doi.org/10.1016/0196-

6774(92)90004-V. URL: http://www.sciencedirect.com/science/article/pii/019667749290004V.

[7] Jun Ma and Shaohan Ma. "An $O(k^2n^2)$ algorithm to find a k-partition in a k-connected graph". In: *Journal of Computer Science and Technology* 9.1 (1994), pp. 86–91. ISSN: 1860-4749. DOI: 10.1007/BF02939489. URL: http://dx.doi.org/10.1007/BF02939489.

[8] M. Srinivas and L. M. Patnaik. "Genetic algorithms: a survey". In: *Computer* 27.6 (June 1994), pp. 17–26. ISSN: 0018-9162. DOI: 10.1109/2.294849.

[9] Monika Rauch Henzinger and Valerie King. "Randomized Dynamic Graph Algorithms with Polylogarithmic Time Per Operation". In: *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*. STOC '95. Las Vegas, Nevada, USA: ACM, 1995, pp. 519–527. ISBN: 0-89791-718-9. DOI: 10.1145/225058.225269. URL: http://doi.acm.org/10.1145/225058.225269.

[10] Ronald Becker et al. "Max-min partitioning of grid graphs into connected components". In: *Networks* 32.2 (1998), pp. 115–125. ISSN: 1097-0037. DOI: 10.1002/(SICI)1097-0037(199809)32:2<115::AID-NET4>3.0.CO;2-E. URL: http://dx.doi.org/10.1002/(SICI)1097-0037(199809)32:2%3C115::AID-NET4%3E3.0.CO;2-E.

[11] R. Becker et al. "A Polynomial-Time Algorithm for Max-Min Partitioning of Ladders". In: *Theory of Computing Systems* 34.4 (Aug. 2001), pp. 353–374. ISSN: 1433-0490. DOI: 10.1007/s00224-001-0008-8. URL: https://doi.org/10.1007/s00224-001-0008-8.

[12] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-logarithmic Deterministic Fully-dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-edge, and Biconnectivity". In: *J. ACM* 48.4 (July 2001), pp. 723–760. ISSN: 0004-5411. DOI: 10.1145/502090.502095. URL: http://doi.acm.org/10.1145/502090.502095.

[13] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. ISSN: 1089-778X. DOI: 10.1109/4235.996017.

[14] Frédéric Chataigner, Liliane R.B. Salgado, and Yoshiko Wakabayashi. “Approximation and Inaproximability Results on Balanced Connected Partitions of Graphs”. In: *Discrete Mathematics and Theoretical Computer Science* 9 (2007), pp. 177–192. ISSN: 1365-8050.

[15] Bill Bishop. *The Big Sort: Why the Clustering of Like-Minded America is Tearing Us Apart*. Houghton Mifflin, 2008.

[16] Azavea. *Redrawing the Map on Redistricting 2010: A National Study*. 2009. URL: https://cdn.azavea.com/com.redistrictingthenation/pdfs/Redistricting_The_Nation_White_Paper_2010.pdf.

[17] Clemens Puppe and Attila Tasnádi. “Optimal redistricting under geographical constraints: Why “pack and crack” does not work”. In: *Economics Letters* 105.1 (2009), pp. 93–96. ISSN: 0165-1765. DOI: https://doi.org/10.1016/j.econlet.2009.06.008. URL: http://www.sciencedirect.com/science/article/pii/S0165176509001967.

[18] Micah Altman and Michael P McDonald. “The Promise and Perils of Computers in Redistricting”. In: *Duke Journal of Constitutional Law and Public Policy* 5 (2010), 69–159. URL: http://www.law.duke.edu/journals/djclpp/?action=downloadarticle%5C&id=167.

[19] Roland G. Fryer and Richard Holden. “Measuring the Compactness of Political Districting Plans”. In: *The Journal of Law and Economics* 54.3 (2011), pp. 493–535. ISSN: 00222186, 15375285. URL: http://www.jstor.org/stable/10.1086/661511.

[20] T. R. Hunter. “The First Gerrymander?: Patrick Henry, James Madison, James Monroe, and Virginia’s 1788 Congressional Districting”.

In: *Early American Studies: An Interdisciplinary Journal* 9.3 (2011), pp. 781–820. DOI: doi:10.1353/eam.2011.0023.

[21] Tommy R Jensen and Bjarne Toft. *Graph coloring problems*. Vol. 39. John Wiley & Sons, 2011.

[22] Justin Levitt. "Weighing the Potential of Citizen Redistricting". In: *Loyola of Los Angeles Law Review* (2011). URL: https://ssrn.com/abstract=1710191.

[23] Elizabeth Pennisi. "ENCODE Project Writes Eulogy for Junk DNA". In: *Science* 337.6099 (2012), pp. 1159–1161. ISSN: 0036-8075. DOI: 10.1126/science.337.6099.1159. eprint: http://science.sciencemag.org/content/337/6099/1159.full.pdf. URL: http://science.sciencemag.org/content/337/6099/1159.

[24] Jowei Chen and Jonathan Rodden. "Unintentional Gerrymandering: Political Geography and Electoral Bias in Legislatures". In: *Quarterly Journal of Political Science* 8.3 (2013), pp. 239–269. ISSN: 1554-0626. DOI: 10.1561/100.00012033. URL: http://dx.doi.org/10.1561/100.00012033.

[25] Steven Hill. *How the Voting Rights Act Hurts Democrats and Minorities*. June 2013. URL: https://www.theatlantic.com/politics/archive/2013/06/how-the-voting-rights-act-hurts-democrats-and-minorities/276893/.

[26] Nathaniel Persily and Thomas Mann. "*Shelby County v. Holder* and the Future of the Voting Rights Act". In: *Governance Studies at Brookings* (2013). URL: https://www.brookings.edu/wp-content/uploads/2016/06/Persily_Mann_Shelby-County-v-Holder-Policy-Brief_v9.pdf.

[27] David Weigel. *The Voting Rights Act and the GOP's Gerrymandering Advantage*. Jan. 2013. URL: http://www.slate.com/blogs/

weigel/2013/01/17/the_voting_rights_act_and_the_gop_s_gerrymandering_advantage.html.

[28] Zeph Landau and Francis Edward Su. "Fair Division and Redistricting". In: *CoRR* abs/1402.0862 (2014). URL: http://arxiv.org/abs/1402.0862.

[29] J. C. Mattingly and C. Vaughn. "Redistricting and the Will of the People". In: *ArXiv e-prints* (Oct. 2014). arXiv: 1410.8796 [physics.soc-ph].

[30] *Compactness is a terrible standard for redistricting and determining if maps were gerrymandered*. May 2015. URL: https://www.dailykos.com/stories/2015/5/12/1384062/-Compactness-is-a-terrible-standard-for-redistricting-and-determining-if-maps-were-gerrymandered.

[31] Nicholas O. Stephanopoulos and Eric M. McGhee. "Partisan Gerrymandering and the Efficiency Gap". In: *University of Chicago Law Review* 82 (2015), pp. 831–900. URL: https://lawreview.uchicago.edu/publication/partisan-gerrymandering-and-efficiency-gap-0.

[32] S. Bangia et al. "Redistricting: Drawing the Line". In: *ArXiv e-prints* (Apr. 2017). arXiv: 1704.03360 [stat.AP].

[33] Anita Earls. *Could This Put an End to Gerrymandering?* June 2017. URL: https://www.thenation.com/article/could-this-put-an-end-to-gerrymandering/.

[34] Yoad Lewenberg, Omer Lev, and Jeffrey S. Rosenschein. "Divide and Conquer: Using Geographic Manipulation to Win District-Based Elections". In: *AAMAS*. 2017.

[35] Amy Howe. *Argument preview: The justices tackle partisan gerrymandering again*. URL: http://www.scotusblog.com/2017/09/argument-preview-justices-tackle-partisan-gerrymandering/.

[36] Justin Levitt. *Who draws the lines?* URL: http://redistricting.lls.edu/who.php.

[37] Michael P. McDonald and Micah Altman. *DistrictBuilder*. hhttp://www.districtbuilder.org/.

[38] Michael P. McDonald and Micah Altman. *The Public Mapping Project*. http://www.publicmapping.org/resources/data/.

[39] *State-by-state redistricting procedures*. URL: https://ballotpedia.org/State-by-state_redistricting_procedures.

[40] Eric W. Weisstein. *Cake Cutting*. URL: http://mathworld.wolfram.com/CakeCutting.html.