

RESEARCH ARTICLE

A Clustering Approach for NSGA-II in Political Redistricting

Blake Splitter¹  | Matthew J. Saltzman²  | Mary E. Kurz³ 

¹Mathematics Department, Tennessee Wesleyan University, Athens, Tennessee, USA | ²School of Mathematical and Statistical Sciences, Clemson University, Clemson, South Carolina, USA | ³Department of Industrial Engineering, Clemson University, Clemson, South Carolina, USA

Correspondence: Blake Splitter (blakesplitter@gmail.com)

Received: 12 December 2025 | **Revised:** 9 July 2026 | **Accepted:** 12 July 2026

Keywords: crossover | genetic algorithms | multiobjective | NSGA-II | political redistricting | redistricting

ABSTRACT

Political redistricting is a complex, multi-objective problem that has been extensively studied. One approach to solve this geographical partitioning problem is to use multiobjective genetic algorithms applied to the adjacency graph of precincts. Following the work of Vanneschi et al., we develop an algorithm using a Non-Dominated Sorting Genetic Algorithm (NSGA-II) to build electoral plans in the state of South Carolina. Our work (which we name Cluster-NSGA-II) utilizes a novel clustering approach to breed children rather than the standard crossover step. We find that this approach is competitive with other similar algorithms in terms of both solution diversity and Pareto optimality.

1 | Introduction

Political Redistricting is a well-studied problem in which several small geographic units (GUs) are partitioned into contiguous groups, known as districts. Each of these districts then elects one or more representatives to advocate for their constituents in their respective legislatures. The way these districts are drawn has a significant impact on the final election outcome. It is therefore imperative that districting *plans* be drawn in a way that does not unduly disadvantage a group of people.

The goal of this project is to develop an algorithm that generates districting plans that consider multiple competing objectives. The objectives we have selected for this project are:

1. Population equinumerosity
2. District compactness
3. Minimizing county splits

Further details about these objectives can be found in Subsection 1.2. To account for these objectives, we utilize the Non-Dominated Sorting Genetic Algorithm (NSGA-II) framework. This genetic algorithm breeds good solutions with one another to produce other high-quality solutions [1]. Conducting this procedure over many generations has been demonstrated to be an effective metaheuristic strategy for solving various complex problems. Our work in this paper follows many of the ideas introduced in Vanneschi et al. [2]. Our contribution emerges from the novel clustering approach, which we find to be better tailored for the context of genetic algorithms. We also increase the number of objectives considered in [2] from two to three.

This paper is divided into the following sections. In Section 1, we introduce the problem and the objectives used in the paper. Section 2 provides an overview for multiobjective optimization. Section 3 describes the algorithm used in this paper and the experimental design employed. Section 4 then displays the results of the experiments, and Section 5 concludes the paper and discusses potential future work.

Blake Splitter is the primary author.

© 2026 Wiley Periodicals LLC.

All rights reserved, including rights for text and data mining and training of artificial intelligence technologies or similar technologies.

1.1 | Problem Overview

In the United States, nearly every state’s congressional map must be redrawn every decade to account for changes in population. Gerrymandering is the process of creating voting districts that unfairly advantage one group of people over another. The problem of creating voting districts that fairly represent their populations is one of the most well-studied problems in political science, and the impulse to remove human bias from redistricting seems a natural one. In fact, the desire to automate the redistricting process dates back to at least the early 1960s when Vickrey wrote, “[T]he [redistricting] process should be completely mechanical, so that once set up, there is no room at all for human choice” [3]. Since then, numerous attempts have been made to utilize computers in an effort to automate district drawing.

It is important to distinguish our work as a project meant to *create* district plans; many other authors study the important, but very different problem of assessing district plans after they have been proposed. Early work on this front involved quantifying the compactness of legislative districts, arguing that noncompact districts may be evidence of gerrymandering. In view of this, many authors developed “compactness tests” to pinpoint gerrymandering (e.g., [4–6]). Other authors have sought to uncover gerrymandering with political fairness tests. These tests use previous voting data as an estimate for future voting patterns. Based on these predictions, authors develop objectives to determine how ‘fair’ a district plan is. Tests such as median-mean [7], efficiency gap [8], and partisan asymmetry [9] all fall into this category of work. Ultimately, many significant studies aim to assess the fairness of a district plan by comparing it to a range of reasonable alternative plans. Ideally, we would enumerate every possible districting plan and compare the proposal to the universe of options, but the number of legal districting plans is incomprehensibly large, making any enumerative strategy effectively impossible.

In contrast to studying gerrymandering *ex post facto*, our work focuses primarily on eliminating the gerrymander before it can emerge. Many works before us have sought to create district plans using computer automation in some way or another. Several authors utilize Voronoi approaches (a technique designed to divide many data points into clusters) to create compact districts of relatively equal populations (e.g., [10–12]). A variety of metaheuristic techniques have been attempted, including simulated annealing, tabu search, and old bachelor acceptance, among others. Most of these approaches optimize a single weighted-sum objective function that consists of one or more objectives intended to indicate a good districting plan. See for example [13–15].

Other authors have sought direct integer programming (IP) approaches. This work began in 1965 with Hess et al. [16], who worked to optimize compactness through geometric calculations of moment of inertia. This work modeled a hypothetical IP to solve this problem without actually solving it. In 1983, Birge formulated a quadratic program to minimize the number of county splits [17]. This work was later eclipsed by Shahmizad and Buchanan [18] in 2023 when they found the optimal county split

value for every state. Recently, Swamy et al. utilized a multilevel clustering approach to improve political metrics and compactness [19]. Finally, Belotti et al. optimized Polsby-Popper compactness by utilizing advances in mixed-integer programming software using counties as geographic units in 2025 [20].

Our work focuses more closely on genetic algorithms, particularly NSGA-II. Several authors have utilized this algorithm on the redistricting problem, including [2, 21–23]. We follow the work done by Vanneschi et al. in 2017 [2] somewhat closely as we build our algorithm.

The general idea behind genetic algorithms is to combine (or breed) the qualities of two feasible solutions into a third solution (a child), with the hope that the child produces better objectives than either parent. Some children undergo “mutations” to better explore the state space.

1.2 | Objectives

The objectives used in this project are described below. We utilize some subset of these objectives when performing experiments. Every objective is minimized, and every objective is assigned a user-specified upper bound; our algorithm rejects any plan that has an objective exceeding its upper bound.

Population equinumerosity is the principle that all districts in a plan should have equal populations. This is a well-established standard, as the Supreme Court of the United States has declared in numerous cases that the “one person, one vote” doctrine applies to both congressional plans and state legislative plans [24–26]. In this project, we achieve population equinumerosity by minimizing the sum of each district’s population deviation (PD) from the ideal population. Equation (1) details the calculation for this objective.

$$PD = \sum_{i \in D} |p_i - I| \quad (1)$$

Here, D is the set of districts, p_i is the population of the i th district, and I is the ideal population of a district and can be calculated as $I = \frac{1}{|D|} \sum_{i \in D} p_i$.

District compactness is generally acknowledged as a necessary constraint to discourage gerrymandering. Broadly speaking, a district is compact if it is tightly packed, meaning it is not spread out far geographically. There are various ways to measure a district’s compactness, including those that consider the district’s geometry, those that examine the distribution of people, and those that rely on the underlying graph. We direct the reader to [27] for a more comprehensive view of several compactness measures. In this project, we utilize Polsby-Popper compactness, a geometric measure that calculates the ratio of a district’s area to the area of a circle with the same perimeter as the district. However, any other compactness measure should also work well in our algorithm. Because we minimize the various objectives, we calculate the *shifted Polsby-Popper* (PP_s) score, detailed in Equation (2).

$$PP_s = \frac{1}{|D|} \sum_{i \in D} \left(1 - \frac{4\pi A_i}{r_i^2} \right) \quad (2)$$

Here, A_i is the area of the i th district and r_i is the perimeter of the i th district.

A county split (*CS*) occurs when a county is split between two or more districts. Because counties often share a sense of community (due to shared laws for their citizens), it is desirable to avoid splitting counties into multiple districts. Moreover, many legislative bodies make this constraint a high priority to avoid voter confusion. In this paper, we calculate the number of county splits by subtracting the minimum of the number of counties C and districts D from the number of county-district intersections (CDI), as detailed in Equation (3). We seek to minimize this quantity.

$$CS = CDI - \min(|C|, |D|) \quad (3)$$

2 | Multiobjective Optimization

Multiobjective Optimization is the study of simultaneously optimizing two or more objectives subject to one or more constraints. Formally, without loss of generality, a multiobjective problem can be modeled as:

$$\begin{aligned} \min \quad & [f_1, f_2, \dots, f_n] \\ \text{s.t.} \quad & [\text{constraints}] \end{aligned} \quad (4)$$

where $\{f_i\}_{i=1}^n$ are the objective functions. In contrast to single-objective problems, multiobjective problems do not generally have a single solution $\mathbf{x}^*$ that optimizes all objectives simultaneously. Instead, a set of solutions is considered optimal if there is no solution that can improve at least one objective value without worsening any other objective. For instance, consider a two-objective (also known as biobjective) minimization problem, with objectives f_1 and f_2 . The optimal set of solutions is known as the *Pareto front* and is deemed to be *Pareto optimal*. Figure 1 shows a sample objective function space for such a problem.

We say that an outcome $\mathbf{y}^1$ *dominates* outcome $\mathbf{y}^2$ if $y_i^1 \leq y_i^2$ for all objectives $i \in [n]$ and that for at least one objective i^* , $y_{i^*}^1 < y_{i^*}^2$. We further say that $\mathbf{y}^2$ is *dominated* by $\mathbf{y}^1$. If neither $\mathbf{y}^1$ nor $\mathbf{y}^2$ dominates the other, then we say that the two outcomes are

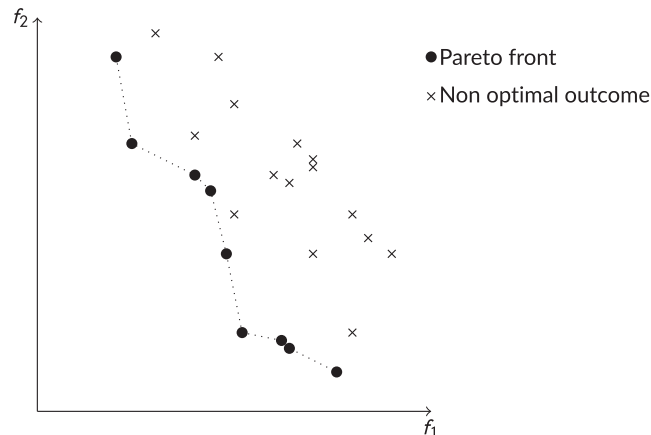


FIGURE 1 | Biobjective optimization problem objective function space.

nondominated with respect to each other. The Pareto front will be the set of outcomes that are nondominated with respect to each other and are not dominated by any other outcome.

Obtaining exact solutions for multiobjective problems is both possible and well-studied (see, e.g., [28]). In this project, however, due to the sheer quantity of feasible solutions, we focus on metaheuristic techniques. To assess the efficacy of the approach, we utilize a hypervolume metric.

Hypervolume is a widely used metric for assessing the performance of a set of multiobjective outcomes. The hypervolume indicator measures the volume of the objective space dominated by a given set of outcomes, relative to a reference point.

Consider Figure 2. The area of the region bounded “northeast” of all outcome points but “southwest” of the reference point represents the hypervolume of this set of outcomes. If a new outcome is found ($\mathbf{y}^*$ in the figure) that dominates (or is nondominated with respect to) one or more outcomes previously in the set, then the hypervolume increases, since the area grows. As such, any algorithm that gradually finds more nondominated outcomes should have a monotonically increasing hypervolume function.

There are several important notes to make about this calculation:

- This can be easily extended into higher dimensions if there are more than two objectives.
- All objectives are minimized in this example. It is therefore important that there is an upper bound for each objective; for without one, no reference point could be reliably plotted. One way to ensure this is to reject any outcomes with objectives exceeding a predetermined upper bound.
- It is wise to scale each objective so that they are all of similar magnitude. For instance, if f_1 has an upper bound of 10^6 and f_2 has an upper bound of 1, then improvements to f_2 would have minimal impact on the hypervolume.
- The choice of a reference point can be impactful. Choosing a reference point far northeast of the region would produce

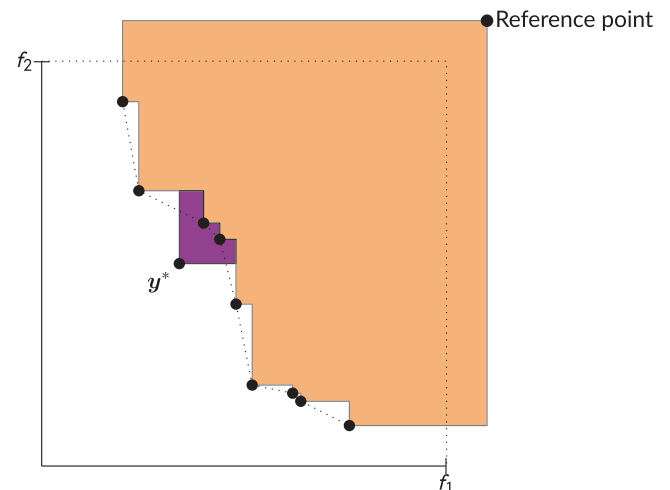


FIGURE 2 | Hypervolume increases after finding a new nondominated outcome $\mathbf{y}^*$.

minimal improvements to hypervolume compared to the current area of the region. A standard choice for the reference point is $(1.1, \dots, 1.1)$ if each objective has an upper bound of 1. Ishibuchi discusses additional considerations for choosing a proper reference point [29].

3 | Methods

3.1 | Previous Approaches

Vanneschi et al.'s algorithm uses NSGA-II and a Variable Neighborhood Search to perform political redistricting [2]. For brevity, we will refer to Vanneschi's algorithm as VNSGA-II from now on. VNSGA-II is summarized in Algorithm 1.

In general, the philosophy behind genetic algorithms is that if you select two solutions to be parents, combining the "genes" of the parents often produces even better children. In Algorithm 1 (and all other NSGA-II redistricting works to our knowledge), we notice that there is nothing inherently valuable about the genes of the parents. In the example detailed in Algorithm 1's step 2, there is no reason to believe that the 225th GU being assigned to district 2 influences (positively or negatively) the quality of the solution. However, we argue that there *is* inherent value in determining which GUs should be clustered together in the same districts.

To illustrate our point, consider the example graphs in Figure 3. Suppose each node represents a GU, and an edge is present if the GUs have a nonzero-length boundary. The color (and pattern)

ALGORITHM 1 | VNSGA-II for political redistricting [2].

Input:

```

 $n$ : population size
 $g$ : number of generations to run algorithm
 $G = (V, E)$ : Geographic Unit (GU) graph with vertex set  $V$  and edge set  $E$ 
 $d$ : number of districts to create
 $t$ : number of plans to choose for tournament selection
 $\mu$ : mutation probability
 $p_{VNS}$ : probability to use variable neighborhood search
1: Randomly generate  $n$  feasible plans and add them to a set  $P$ . This is achieved by selecting  $d$  random nonadjacent nodes as the centers of the districts. Then choose a random district  $D$  to expand. Add all unassigned nodes surrounding  $D$  to  $D$ . Repeat until all nodes are assigned.
2: Encode each plan as an array  $s$  with length equal to the number of GUs. The  $i$ th element of this array describes the district to which the  $i$ th GU is assigned. For example,  $s[225] = 2$  would mean that the 225th GU is assigned to district 2.
3: while  $gen < g$  do
4:   Create an empty set  $P'$  which will contain the children of this generation.
5:   while  $|P'| < n$  do
6:     if  $|P'| < 0.6n$  then perform crossover:
7:       Select two plans  $s_1$  and  $s_2$  in  $P$  to be the parents using tournament selection (described in lines 8–12).
8:       for each parent do
9:         Randomly select  $t$  plans from  $P$ .
10:        Among those  $t$  plans, pick the plan(s) with the lowest rank (closest to the best front)
11:        If more than one plan is selected, choose the one closest to the median plan of the front.
12:      end for
13:      Breed these two parents together to create two children  $s'_1$  and  $s'_2$ .
14:      For some GU  $i$ , let  $s_1[i] = D_1$  and  $s_2[i] = D_2$ .
15:      Pick a random crossover point  $c$  (i.e., pick an integer between 1 and  $|V|$ )
16:      Breed child 1:  $s'_1 := s_1$  except for each GU  $i \leq c$ ,  $s'_1[i] := D_2$  if contiguity in  $s'_1$  is not violated.
17:      Breed child 2:  $s'_2 := s_2$  except for each GU  $i > c$ ,  $s'_2[i] := D_1$  if contiguity in  $s'_2$  is not violated.
18:      Add  $s'_1$  and  $s'_2$  to  $P'$ .
19:     else if  $|P'| < 0.9n$  then perform mutation:
20:       Randomly select a plan  $s \in P$ . Set  $s' := s$ .
21:       In  $s'$ , randomly swap up to  $\mu |V|$  GUs to neighboring districts, ensuring contiguity isn't violated.
22:       Add  $s'$  to  $|P'|$ .
23:     else perform copying:
24:       Randomly select a plan  $s \in P$ . Add  $s$  to  $|P'|$ .
25:     end if
26:   end while
27:   Perform a variable neighborhood search (VNS) on  $p_{VNS} \cdot n$  plans.
28:   Among all plans in  $P \cup P'$ , build Pareto fronts according to the objective evaluations.
29:   Calculate the crowding distance for each of the plans.
30:   Add the best  $n$  plans to a set  $P''$ . Empty  $P$  and  $P'$ .  $P \leftarrow P''$ . Empty  $P''$ .
31:    $gen \leftarrow gen + 1$ 
32: end while

```

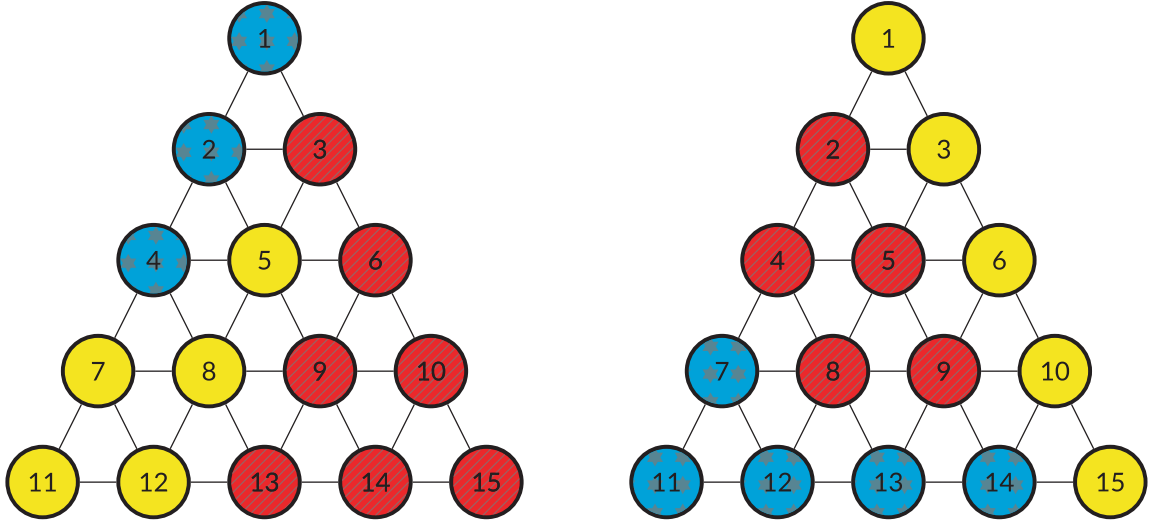


FIGURE 3 | An example of two potential parents. Each node represents a GU, and each edge signifies that those GUs are adjacent. Node colors correspond to different districts.

of the nodes represents the districts to which they are assigned. Letting the blue (starred) nodes correspond to district 1, red (diagonally lined) nodes correspond to district 2, and yellow (pattern-less) nodes correspond to district 3, previous NSGA-II approaches encode the parents as arrays:

$$s_1 = \begin{bmatrix} 1 & 1 & 2 & 1 & 3 & 2 & 3 & 3 & 2 & 2 & 3 & 3 & 2 & 2 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

$$s_2 = \begin{bmatrix} 3 & 2 & 3 & 2 & 2 & 3 & 1 & 2 & 2 & 3 & 1 & 1 & 1 & 1 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

If we arbitrarily set the crossover point to be 3, then to create child 1 s'_1 , we iteratively check through the indices $i = 1, 2, 3$ from s_1 and see if they can change district to $s_2[i]$ without violating contiguity. More precisely, we observe that:

- $s_1[1] = 1$ currently. We check to see if changing $s_1[1]$ to 3 causes any discontinuity. From Figure 3, we notice that node 1 cannot turn yellow, else the yellow district would be discontinuous.
- $s_1[2] = 1$ currently. We check to see if changing $s_1[2]$ to 2 causes any discontinuity. From Figure 3, we notice that node 2 cannot turn red, as the blue district would be discontinuous.
- $s_1[3] = 2$ currently. We check to see if changing $s_1[3]$ to 3 causes any discontinuity. From Figure 3, it does not create a discontinuity, so s'_1 is identical to s_1 except $s'_1[3] = 3$.

The issue here is that s'_1 is not “inheriting” any traits from s_2 . Most NSGA-II efforts effectively implement arbitrary GU flips to generate new plans. Even if the objectives evaluated for both parents are strong, there is no reason to believe that this crossover generates children with strong objectives themselves.

VNSGA-II utilizes a variable neighborhood search (VNS) in step 27 of Algorithm 1. This is where most of the plan improvement

occurs. To summarize, VNS seeks to optimize a plan by repeatedly applying hill climbing with respect to different neighborhood structures $\{N_i\}_{i=1}^4$. Each structure defines the type of perturbation that can be performed to get from one plan to another. The neighborhood structures are:

- N_1 : GU flips¹ that improve population balance
- N_2 : GU flips that improve compactness
- N_3 : GU swaps² that improve population balance
- N_4 : GU swaps that improve compactness

All neighborhood structures further require that all districts remain contiguous after the perturbation. With predetermined probability p_{VNS} , VNS performs the following steps on a plan s for each neighborhood structure N_i :

1. Select a solution s from the current population P .
2. Generate a solution s' by making a legal perturbation according to N_i . This perturbation need not improve the plan’s objectives.
3. Apply hill climbing³ on s' using legal perturbations according to N_i to create s'' .
4. If s'' improves upon N_i ’s objective, then set $s := s''$.

It is important to note that VNSGA-II does not explicitly aim to minimize the number of county splits. As we will see in Subsection 3.2.4, Cluster-NSGA-II improves upon VNSGA-II by considering this third objective.

3.2 | Our Approach

In contrast to previous works, we employ a novel clustering approach that more effectively models the theme of genetic algorithms. Broadly speaking, we group several neighboring GUs together into “clusters” and use those clusters as building blocks

Input:

g : Number of generations
 μ : Mutation probability
 n : Population size
 d : Number of districts
 U : Upper bounds for each objective
 $G = (V, E)$: GU graph with vertex set V and edge set E

- 1: Build n random *cluster graphs* on the graph G . ▷ See Subsection 3.2.1
- 2: Build a district graph on each cluster graph
- 3: $gen \leftarrow 0$
- 4: **while** $gen < g$ **do**
- 5: Crossover. Breed n child cluster graphs using a *crossover path*. ▷ See Subsection 3.2.2
- 6: Mutation. Mutate $\mu \cdot n$ children. ▷ See Subsection 3.2.3
- 7: Create district plans on these child cluster graphs. ▷ See Subsection 3.2.4
- 8: Evaluate the objectives on all $2n$ plans in generation gen .
- 9: Sort the $2n$ plans into ranked fronts, with the rank 1 front being the current approximated Pareto Front.
- 10: Assign the best n plans (according to front rank) to be the potential parents of the next generation. Use the crowding distance to determine the worst entrants to the next generation. Both the district graph and cluster graph are maintained through the next generation. ▷ See Subsection 3.2.5
- 11: $gen \leftarrow gen + 1$
- 12: **end while**

for the districts. The fundamental premise of this approach is that if a cluster of GUs helps create a “good” district, then it survives from generation to generation. A rough sketch of our approach can be found in Algorithm 2.

3.2.1 | Clustering

Our approach utilizes a novel clustering approach, where we cluster several nearby GUs together to form the building blocks for districts. If this grouping of GUs helps create quality districts, these clusters are preserved generation to generation. The motivation for this approach emerges from the observation that it is much easier to create compact districts if the underlying building blocks (i.e., the clusters) are compact as well.

To demonstrate a cluster graph example, consider the graph in Figure 4. There are six clusters in Figure 4, each represented by a unique color. In our code, clusters are constructed with anywhere between two and six GUs. The range of two to six GUs was determined experimentally. It was found that modifying this range did not significantly impact the outcomes. The districts are then built using these clusters. For example, if the cluster graph in Figure 4 has three districts, then those districts may contain the following GUs:

- District 1: GUs 1, 2, 3, 4, 5, 8, 9
- District 2: GUs 6, 10, 14, 15, 19, 20, 21
- District 3: GUs 7, 11, 12, 13, 16, 17, 18

In this algorithm, cluster graphs are created using the procedure outlined in Algorithm 3. The process by which districts are created on the underlying clusters is explained further in Subsection 3.2.4.

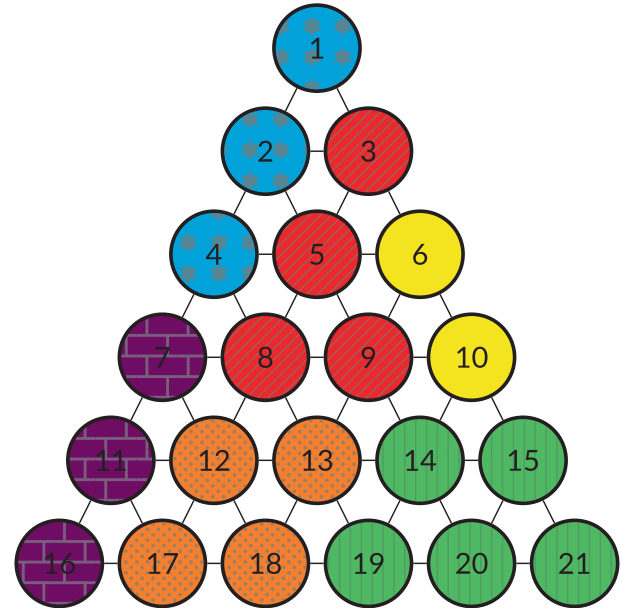


FIGURE 4 | An example of a cluster graph.

3.2.2 | Crossover

When performing the crossover step of NSGA-II, we start by randomly selecting two parents s_1 and s_2 from the current generation. Each parent is a cluster graph—that is, an assignment of GUs to clusters (as explained in Subsection 3.2.1). Even if the parent has a district plan imbued into it, that district plan is not considered during crossover. Then, on the underlying graph G , we split G into two subgraphs H_1 and H_2 in the following way:

1. Choose a random boundary GU u (a GU that has a nonzero-length boundary with the border of the state).

Input:

GU graph $G = (V, E)$ with vertex set V and edge set E

- 1: **while** there exists an unassigned GU $\in V$ **do**
- 2: Choose a random unassigned GU to serve as the center for a new cluster C .
- 3: Assign this cluster C a random maximum size M_C between 2 and 6.
- 4: **while** the number of nodes in C is less than M_C **do**
- 5: Find all unassigned neighboring GUs of C and choose one randomly to add to this cluster. We disallow clusters from crossing county lines.
- 6: If there are no unassigned neighboring GUs of C , adjust M_C to be the current size of the cluster.
- 7: **end while**
- 8: **end while**

2. Determine the shortest graph distance ℓ_{uv} between u and every other boundary GU v .
3. Assign a weight w_{uv} to each (u, v) pair, where $w_{uv} = (\ell_{uv})^2$.
4. Using a random weighted selection,⁴ pick v . Let the shortest path between u and v be described as the *crossover path*.
5. Let H_1 and H_2 be the subgraphs of G on either side of the crossover path. Randomly choose either H_1 or H_2 to include all nodes in the crossover path. If the number of subgraphs is not exactly two, reselect u and v from step 1. If the number of nodes in H_1 or H_2 is less than 20% of the total number of nodes, reselect u and v from step 1.

The weighting performed in step 3 is designed to incentivize the creation of larger subgraphs. If one subgraph is significantly smaller than the other, then the genes of the parents mix less, which is what we seek to avoid.

Now, for each cluster C in s_1 , if C is a subgraph of H_1 (i.e., $C \subseteq H_1$), then the child s' adopts cluster C . Similarly, for each cluster C in s_2 , if $C \subseteq H_2$, then s' adopts cluster C . For any GUs that remain unclustered in s' , assign them as singleton clusters. To demonstrate this concept, consider Figure 5. The parents s_1 and s_2 breed a child s' by splitting their clusters along the crossover path. H_2 was randomly chosen to contain all nodes in and above the crossover path, while H_1 contains all nodes below the crossover path. In any given generation, n children are created from $2n$ randomly chosen parents in the generation, where n is the population size. Therefore, in any given generation, there are $2n$ cluster graphs (n parents and n children).

3.2.3 | Mutation

In our algorithm, mutation occurs in several ways. Denote the mutation probability as μ and the population size as n . After the n children are bred using crossover, we mutate μn children (rounding to the nearest integer). There are three operations that are performed on the children:

1. Merging a GU into a neighboring cluster that it is not currently a part of.

2. Splitting a GU that is currently part of a cluster into a singleton cluster.
3. Using recombination to recreate two districts.

Regarding the first two operations, if a child has been designated as one to be mutated, we then randomly pick $\lfloor \mu |V| \rfloor$ of the GUs to mutate (where V is the set of GUs in the graph G). For each GU designated for mutation, we randomly decide whether it will be merged into a neighboring cluster or whether it will be split into a singleton cluster. If the GU selected is already a singleton cluster, then it will be merged into a neighboring cluster. Each time a GU is selected, we first must check whether removing the GU from the cluster causes the cluster to become discontinuous. If so, then we skip that mutation. Further, merges that create clusters crossing county lines are prohibited.

The third mutation operation occurs only after districts are created. See Subsection 3.2.4 for details.

3.2.4 | District Plans

After the mutation step, we then create district plans for the n children. The process by which districts are constructed emphasizes population equality, compactness, and preservation of county boundaries. This procedure also rejects district plans where the objectives are too large. Those upper bounds can be specified by the user. For each child, d districts are created using the procedure outlined in Algorithm 4. Each district would ideally have a population of $I := \frac{P}{d}$, where P is the total population of the state. The code forces each district to have a population in the interval $[\mathcal{L}, \mathcal{U}]$, where $\mathcal{L} = 0.5 * I * \left(\frac{0.99}{0.5}\right)^{gen/g}$, $\mathcal{U} = 1.5 * I * \left(\frac{1.01}{1.5}\right)^{gen/g}$, and gen represents the current generation number. Effectively, the population bounds at generation 0 will be $[0.5I, 1.5I]$ while the population bounds at the final generation g will be $[0.99I, 1.01I]$. The values of $\mathcal{L}$ and $\mathcal{U}$ follow geometric sequences and are inspired by temperature reduction schemes in the metaheuristic approach of simulated annealing.

Population equality is encouraged because, as the districts grow, those with lower populations have a higher probability of being chosen to expand. Compactness is emphasized by weighting the surrounding clusters proportionally to their boundary length with the district. This procedure reduces the likelihood of small offshoots emerging from the district. The code discourages counties from splitting into multiple districts by allowing only a 10% chance of a district “breaking into” a new county, as described in step 6. Further, if in this step, the percentage of clusters assigned has exceeded 50%, 60%, 70%, 80%, 90%, or 95% for the first time, the code will examine all subgraphs of unassigned clusters to determine whether any of those subgraphs are enclaves⁵ of a single district. If so, then all of those unassigned nodes are assigned to that district. This effectively fills in the unassigned holes. Example enclaves and a nonenclave are shown in Figure 6.

After districts are created, on any plan that was previously mutated, we perform another step of mutation by *recombining* two districts in the plan. The concept of recombination (ReCom), first introduced by DeFord et al., is a way to perturb a district

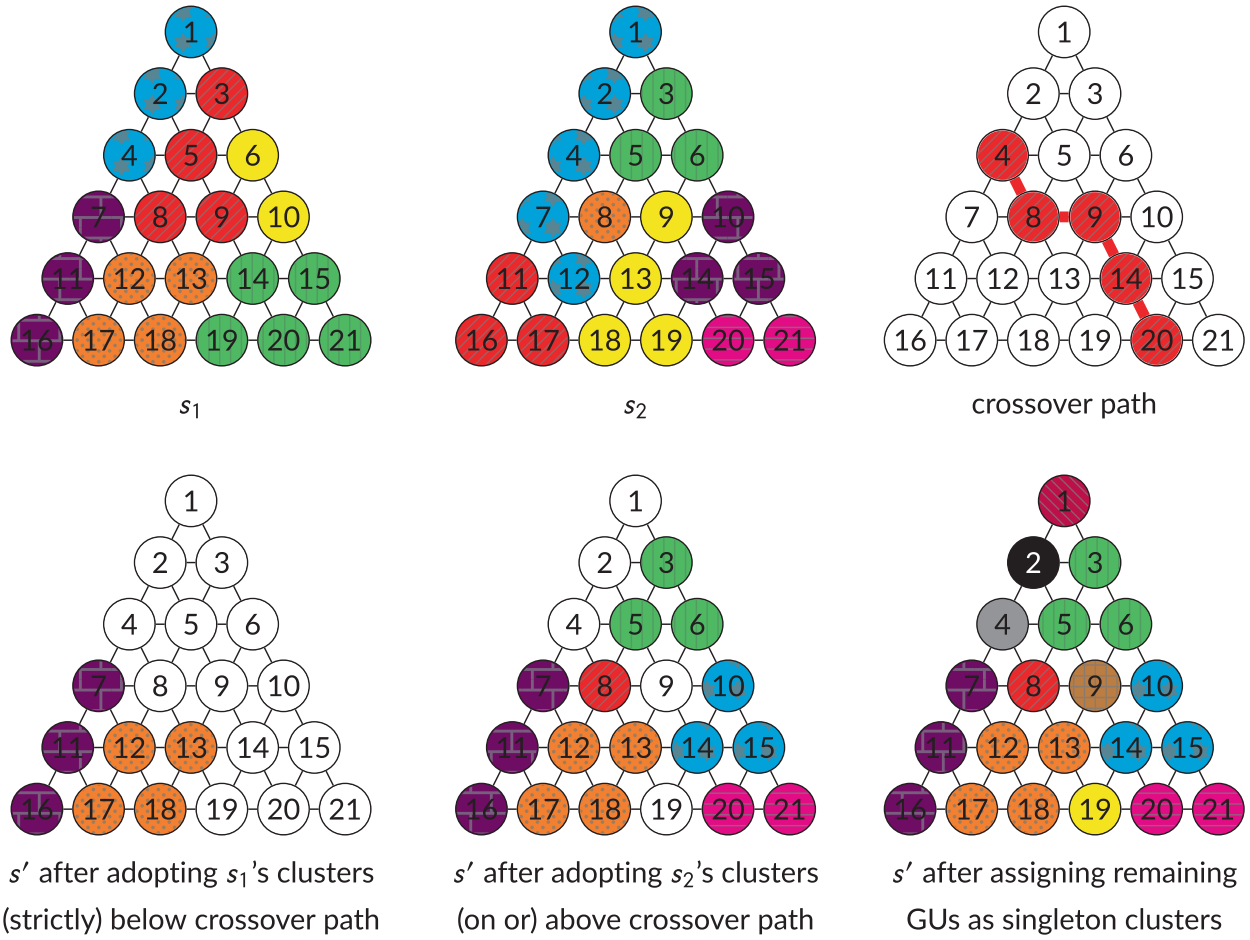


FIGURE 5 | Crossover steps.

ALGORITHM 4 | District building.

Choose d clusters to be the centers for district creation:

- 1: Choose ten sets of d clusters randomly. Each set is a candidate to become the set of centers. These centers will be the seeds from which the d districts will grow.
- 2: Select the set of centers that maximizes the minimum distance between any two centers. This helps ensure that districts are spread apart.

Choose a district to grow and append a cluster to it:

- 1: **while** Any cluster is unassigned and at least one district is eligible to grow **do**
- 2: Assign each district a weight that is inversely proportional to the current population of the district.
- 3: Based on these weights, use a weighted random selection to designate the district D that will grow.
- 4: Assign a new weight to each unassigned neighboring cluster of D that is proportional to its boundary length with district D .
- 5: Use a weighted random selection to designate the cluster C that might be appended to district D .
- 6: Let the set of counties occupied by district D be D_C . Let the county for cluster C be C_C . If $C_C \in D_C$, then append cluster C to district D . If $C_C \notin D_C$, then append cluster C to district D with 10% probability. If this cluster is not appended, then restart from step 5.
- 7: If $p_D \geq U$, then remove district D from the list of districts that are eligible to grow.
- 8: If it is found that no unassigned clusters exist around district D , remove D from the list of districts that are eligible to grow.
- 9: If in this step, the percentage of clusters assigned has exceeded 50%, 60%, 70%, 80%, 90%, or 95% for the first time, then determine whether any of the partially-built districts have unassigned enclaves in them. If so, then assign all clusters in the enclave to that district.
- 10: **end while**
- 11: If this approach failed to assign every cluster to a district or if any district had a population less than L , restart this procedure by choosing ten new sets of centers.
- 12: Calculate all objective values for the plan.
- 13: If any objective exceeds the upper bound defined by the user, reject the plan and restart the procedure by choosing ten new sets of centers.

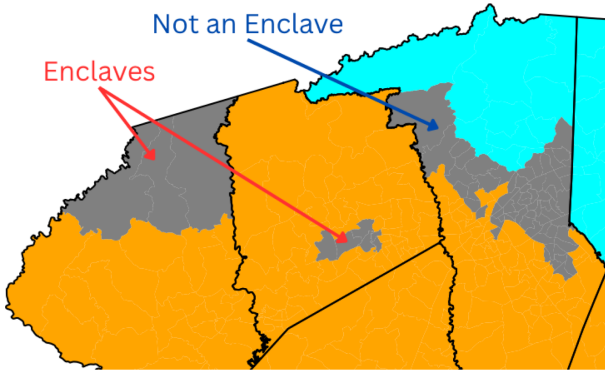


FIGURE 6 | An example of enclaves and nonenclaves. Gray nodes are unassigned.

graph by combining two districts together and then redistributing the nodes of this graph into two new districts [31]. Our algorithm utilizes ReCom in the following way:

1. On a plan designated for mutation, find two neighboring districts, where one has a population above the ideal population and the other has a population below the ideal population.
2. Perform ReCom on these two districts using clusters as the nodes, ensuring that the resultant assignment of districts forces both districts to be within 5% of the mean population of the two districts.

This procedure helps to decrease the population deviation of the plan.

3.2.5 | Nondominated Sorting and Crowding Distance

The remaining steps in our algorithm follow the standard NSGA-II procedure closely. After district plans are created for the n children, we then decide which n of the $2n$ candidate plans are brought to the next generation. Let f_{ij} correspond to the i th objective value from the j th plan. We say that plan s_{j_1} *dominates* plan s_{j_2} if $f_{ij_1} \leq f_{ij_2}$ for all objectives i , and that for at least one objective i^* , we have $f_{i^*j_1} < f_{i^*j_2}$. We say that a plan s is *nondominated* if there is no other plan that dominates it.

Utilizing some subset of the objectives mentioned in Subsection 1.2, we rank the $2n$ plans into fronts based on their status of domination. All of the nondominated plans are placed in rank 1; all plans that are dominated *only* by plans in rank 1 are then placed in rank 2, and so on until all plans have a rank.

Then the best n plans (i.e., the n plans with the best rank) are designated to survive to the next generation. If a front must be split up to determine which plans survive, then we choose the plans that are the most diverse. To accomplish this, we calculate the *crowding distance* (cd) for all plans in the front that must be split up, as shown in Algorithm 5. Then, if x plans must be chosen from front F , we choose the x plans with the highest crowding distance to survive to the next generation. This ensures that the population is as diverse as possible.

ALGORITHM 5 | Crowding distance.

```

1: Initialize  $cd_j = 0$  for each plan  $j$  in front  $F$ 
2: for each objective  $i$  do
3:   Sort the  $j$  plans in ascending order based on objective  $i$ 
4:    $cd_0, cd_{|F|} \leftarrow \infty$ 
5:    $r \leftarrow f_{i|F|} - f_{i0}$  ▷ This is the range of
objective  $i$ 
6:   if  $r = 0$  then
7:      $r \leftarrow 1$ 
8:   end if
9:   for  $j = 1 \rightarrow |F| - 1$  do
10:     $cd_j \leftarrow cd_j + \frac{f_{i(j+1)} - f_{i(j-1)}}{r}$ 
11:   end for
12: end for

```

Once the n plans are designated to survive to the next generation, we maintain their clustering assignments and their district assignments. When breeding children, the parents' clustering assignments are used in the crossover step, but the parents' district assignments are not considered when breeding children, as explained in Subsection 3.2.2.

This overall algorithm repeats until we have reached the predetermined number of generations.

4 | Results

4.1 | Experimental Design

To assess the efficacy of this program, we perform three experiments with 30 trials each. Those experiments are as follows:

1. Cluster-NSGA-II
2. VNSGA-II with Vanneschi's initialization strategy (VNSGA-II VI)
3. VNSGA-II with our initialization strategy (VNSGA-II SI)

To be precise, Experiment 2 performs VNSGA-II as defined in Algorithm 1. Experiment 3 modifies Algorithm 1 by replacing the district initialization step with the district building algorithm we present in Algorithm 4.⁶ Each trial runs the respective algorithm once.

Our code seeks to find Pareto-optimal solutions using population deviation (PD), shifted Polsby-Popper compactness (PP_s), and county splits (CS). Meanwhile, VNSGA-II seeks to optimize population deviation and compactness. See Section 1.2 for a precise calculation of each objective. For the sake of comparison, county splits are tracked in both experiments of VNSGA-II, even though the code makes no explicit effort to optimize them. In each experiment, every trial will contain a nondominated list of plans with respect to its optimized objectives. Then, among the amalgamated list of plans from the 30 trials, we find all nondominated plans. The final list of nondominated plans for each of the

experiments can be found in Tables A1–A3 in Appendix A. The amalgamated Pareto sets for the three experiments will henceforth be referred to as P_N , $P_{V,VI}$, and $P_{V,SI}$.

All trials run for 200 generations ($g = 200$). The mutation probability μ is set to 0.3 for all experiments. We have found a population size of $n = 75$ ensures that no nondominated solutions are lost over the course of the algorithm.

Each objective has an upper bound to ensure that absurd plans aren't being considered at any point during the algorithms. These upper bounds are also essential for the sake of calculating hypervolume. The upper bounds used in this experiment are:

- Population Deviation: $PD \leq 0.4P$, where P is the population of the state. In our data, $P = 5,118,425$.
- Shifted Polsby-Popper Compactness: $PP_s \leq 0.90909$ ($\Rightarrow$ inverse Polsby-Popper score ≤ 10).
- County Splits: $CS \leq 50$.

For each generation, all objectives are scaled by their respective upper bounds so that they have the same relative impact on the hypervolume. The reference point is then placed at the coordinate (1.1, 1.1, ...). We also compare our results to the actual 2024 South Carolina congressional plan, pictured in Figure 9.

4.2 | Experimental Results

We find that the best metric for comparing the algorithms is hypervolume, with respect to all three primary objectives: population deviation, compactness, and county splits. The hypervolume for the amalgamated Pareto set in each experiment is displayed in Table 1. Cluster-NSGA-II's hypervolume far exceeds either instance of VNSGA-II, showcasing the improved results.

While Vanneschi's code produces high-quality plans with respect to population deviation and compactness, it fails to produce plans with small values of county splits, which is why the hypervolume metric falters for VNSGA-II. Two sample plans are displayed for Vanneschi's work in Figure 9 and their objective values are recorded in Table 4.

While performing VNSGA-II in experiments 2 and 3, we noticed that the algorithm produces many similar-looking plans. In light of this observation, we chose to quantify the similarity of the plans to argue that Cluster-NSGA-II produced a larger diversity of outcomes.

TABLE 1 | Hypervolume of each experiment's amalgamated Pareto set.

Experiment	Hypervolume
1: Cluster-NSGA-II	0.39
2: VNSGA-II VI	0.25
3: VNSGA-II SI	0.25

The similarity between two plans is measured in the following way: Two plans are said to be *identical* if all GUs are assigned to the same districts in each plan, after an optimal district relabeling.⁷ For instance, two plans with the same GU–district assignment except with districts 1 and 2 swapped are considered identical. If, after an optimal district relabeling, two plans with $|V|$ GUs differ by d GU assignments, then the *similarity score* between the two plans is given by $\frac{d}{|V|}$. The *similarity score for a trial* represents the average similarity score among all pairs of plans in the Pareto set of the trial. We argue that it is desirable that an algorithm have a low similarity score for each trial, to ensure that the plans are sufficiently diverse.

For each experiment, we then average the similarity score for the 30 trials. The similarity score for each experiment can then be found in Table 2. We argue that Cluster-NSGA-II produces a larger diversity of plans compared to that of VNSGA-II VI or VNSGA-II SI, providing potential policymakers with better options for redistricting. Tables detailing the similarity scores for each trial in each experiment can be found in Tables A4–A6 in Appendix B. Subjecting these data to a Friedman's test results in rejection of the null hypothesis that the algorithms perform the same, at the 0.001 level. Furthermore, pair-wise Wilcoxon tests result in rejecting the null hypothesis that each algorithm performs the same, again at the 0.001 level. The box plots of the similarity data are consistent with these statistical test results (See Figure 7) supporting our general interpretation that the Cluster-NSGA-II performs best of these algorithms.

It is notable that the standard deviation for Cluster-NSGA-II is remarkably low. This suggests that each trial in Cluster-NSGA-II has a similarity score very close to 62.31%. We speculate that this

TABLE 2 | Similarity scores among all pairs of plans in every trial for each experiment.

Experiment	Mean similarity score	Std. dev.
1: Cluster-NSGA-II	62.31%	0.70%
2: VNSGA-II VI	82.34%	9.73%
3: VNSGA-II SI	92.49%	8.57%

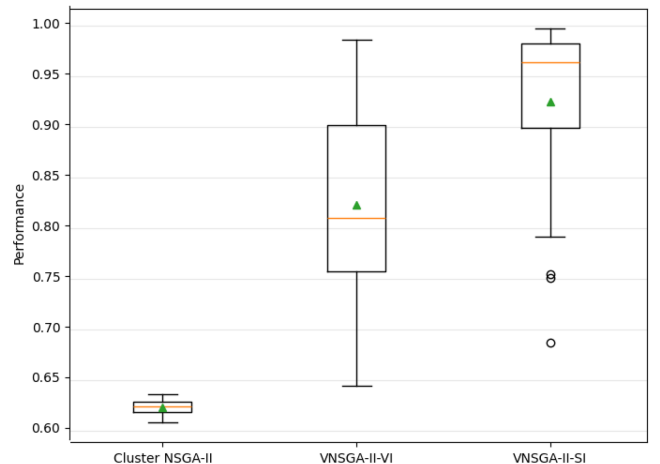
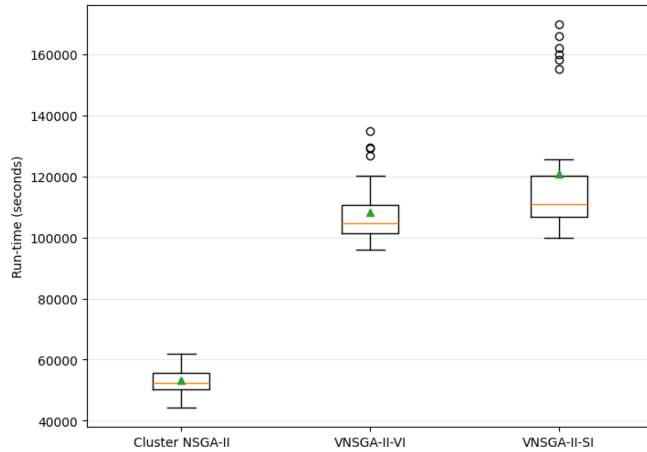


FIGURE 7 | Boxplots of the similarity data for each algorithm.

TABLE 3 | Mean run times for each experiment.

Experiment	Mean run time (s)	Std. dev. (s)
1: Cluster-NSGA-II	53 372	4445
2: VNSGA-II VI	108 419	10 465
3: VNSGA-II SI	120 701	21 842

**FIGURE 8** | Boxplots of the run-time data for each algorithm.

consistency is due to the large Pareto set size of Cluster-NSGA-II, which ranges from 16 to 48, resulting in between 120 and 1128 plan comparisons. When comparing large sets of diverse plans, it seems that the average similarity among those plans converges to around 62.31%. In the VNSGA experiments, the small Pareto set sizes (ranging from 4 to 33) cause wide variety in the similarity scores, explaining their larger deviation from the mean.

The mean run times for each experiment are reported in Table 3. The precise run times for each trial are also reported in Tables A4–A6 in Appendix B. We find that Cluster-NSGA-II runs significantly faster than either version of VNSGA-II. Subjecting these data to a Friedman’s test results in rejection of the null hypothesis that the algorithms use the same computational time, at the 0.001 level. Furthermore, pair-wise Wilcoxon tests result in rejecting the null hypothesis that each algorithm uses the same computational time, at the 0.001 level for the tests involving the Cluster-NSGA-II, while the p value for the test comparing the two variants of the VNSGA. The box plots of the run-time data are consistent with these statistical test results (See Figure 8).

Profiling the VNSGA-II approach reveals that the variable neighborhood search expended the most time while running. We leave open the possibility that this code could be more efficient with better implementation. Together, the similarity score findings and the run-time findings support the interpretation that the Cluster-NSGA-II algorithm performs best of these algorithms on the South Carolina data.

We now display three high quality plans produced by our approach and compare them to the 2024 implemented congressional plan in South Carolina. We also provide one representative plan from VNSGA-II VI (experiment 2) and one representative plan from VNSGA-II SI (experiment 3). This comparison can be found in Figure 9. Table 4 details the objectives for the six plans. The 2024 congressional plan is nondominated with respect to the Pareto set of plans for Cluster-NSGA-II P_N . This suggests our method can produce plans that are as good as real-world plans created by the South Carolina legislature. It is worth mentioning that all plans pictured here have a deviation from ideal population less than 2%.

5 | Conclusion and Future Work

Overall, we conclude that utilizing genetic algorithms, particularly NSGA-II, has promise for optimizing various objectives when making districting plans. We show that utilizing a novel clustering technique can be an effective tool for building districts, and that it follows the theme of genetic algorithms closely. Our work also expands the set of objectives considered in [2] by incorporating county splits in addition to population deviation and compactness.

Future work for this project may include the following improvements or changes:

- It remains to be seen whether the objectives could be lowered further with more generations. Due to time constraints, this code needed to terminate at generation 200, but the overall trend of the hypervolume graphs suggests that local optima were not yet reached.
- Other objectives, such as similarity to previous plans, grouping communities of interest, and adherence to the Voting Rights Act, could still be included and measured as part of a multiobjective framework.
- We used South Carolina as a test case in these examples; future work could include testing on larger states with more districts.

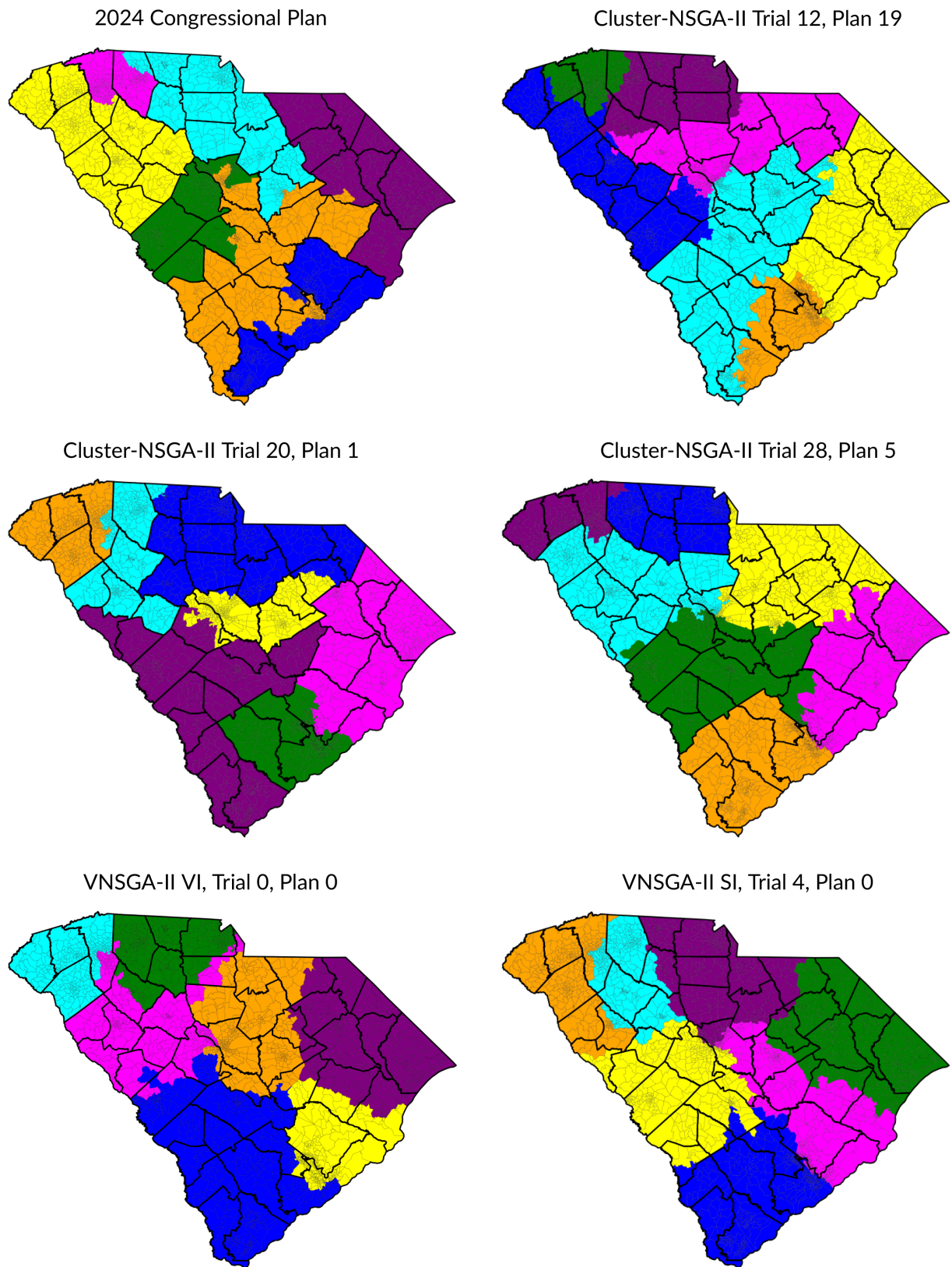


FIGURE 9 | Comparison of 2024 Congressional Plan to Plans Found From Cluster-NSGA-II and Plans Produced by VNSGA-II.

TABLE 4 | Comparison of objectives between Current Congressional Plan, NSGA-II Plans, and VNSGA-II Plans.

Plan	PD	PP _s	CS
2024 Congressional Plan	14 136	0.785	10
Cluster-NSGA-II, Trial 12, Plan 19	7689	0.758	19
Cluster-NSGA-II, Trial 20, Plan 1	56 751	0.725	8
Cluster-NSGA-II, Trial 28, Plan 5	77 869	0.691	11
VNSGA-II VI, Trial 0, Plan 0	7901	0.7	33
VNSGA-II SI, Trial 4, Plan 0	3839	0.708	29

Author Contributions

Blake Splitter: writing – original draft, writing – review and editing, software, conceptualization, investigation, validation, methodology, formal analysis, visualization, data curation, project administration. **Matthew J. Saltzman:** writing – review and editing, resources, conceptualization, methodology, project administration. **Mary E. Kurz:** writing – review and editing, conceptualization, methodology, project administration, formal analysis, writing – original draft.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Endnotes

- ¹ A *flip* is defined as a perturbation that changes the district number of a GU on a district boundary to the neighboring district's number.
- ² A *swap* is defined as a perturbation that exchanges the district numbers D_1 and D_2 of two GUs, each on the $D_1 - D_2$ district boundary.
- ³ Vanneschi et al. did not specify how hill climbing should be done, so we opted to perform no more than 250 perturbations.
- ⁴ Details on the weighted random selection can be found in [30].
- ⁵ A subgraph is an *enclave* of district D if the subgraph is completely surrounded by clusters assigned to district D or if the subgraph partially comprises the boundary of the state but still only borders clusters assigned to district D .
- ⁶ In Algorithm 4, clusters are used for the district building blocks. The modification for VNSGA-II SI uses GUs.
- ⁷ An *optimal district relabeling* relabels the district numbers to maximize the number of overlapping GUs in the two plans.

References

1. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation* 6 (2002): 182–197.
2. L. Vanneschi, R. Henriques, and M. Castelli, "Multi-Objective Genetic Algorithm With Variable Neighbourhood Search for the Electoral Redistricting Problem," *Swarm and Evolutionary Computation* 36 (2017): 37–51.
3. W. Vickrey, "On the Prevention of Gerrymandering," *Political Science Quarterly* 76 (1961): 105–110.
4. M. Dube and J. Clark, "Beyond the Circle: Measuring District Compactness Using Graph Theory," (2016), Northeast Political Science Association Conference.
5. R. G. Niemi, B. Grofman, C. Carlucci, and T. Hofeller, "Measuring Compactness and the Role of a Compactness Standard in a Test

for Partisan and Racial Gerrymandering," *Journal of Politics* 52 (1990): 1155–1181.

6. D. D. Polsby and R. D. Popper, "The Third Criterion: Compactness as a Procedural Safeguard Against Partisan Gerrymandering," *Yale Law and Policy Review* 9 (1991): 301.
7. M. D. McDonald and R. E. Best, "Unfair Partisan Gerrymanders in Politics and Law: A Diagnostic Applied to Six Cases," *Election Law Journal: Rules, Politics, and Policy* 14 (2015): 312–330.
8. N. O. Stephanopoulos and E. M. McGhee, "Partisan Gerrymandering and the Efficiency Gap," *University of Chicago Law Review* 82 (2015): 831–900.
9. S. S. H. Wang, "Three Tests for Practical Evaluation of Partisan Gerrymandering," *Stanford Law Review* 68 (2016): 1263.
10. H. A. Levin and S. A. Friedler, "Automated Congressional Redistricting," *ACM Journal of Experimental Algorithmics* 24 (2019): 1–24.
11. A. G. Novaes, J. S. de Cursi, A. C. da Silva, and J. C. Souza, "Solving Continuous Location–Districting Problems With Voronoi Diagrams," *Computers and Operations Research* 36 (2009): 40–59.
12. L. Svec, S. Burden, and A. Dilley, "Applying Voronoi Diagrams to the Redistricting Problem," *UMAP Journal* 28 (2007): 313–329.
13. M. H. Browdy, "Simulated Annealing: An Improved Computer Model for Political Redistricting," *Yale Law and Policy Review* 8 (1990): 163–179.
14. Y. Y. Liu, W. K. T. Cho, and S. Wang, "PEAR: A Massively Parallel Evolutionary Computation Approach for Political Redistricting Optimization and Analysis," *Swarm and Evolutionary Computation* 30 (2016): 78–92.
15. F. Ricca and B. Simeone, "Local Search Algorithms for Political Districting," *European Journal of Operational Research* 189 (2008): 1409–1426.
16. S. W. Hess, J. Weaver, H. Siegfeldt, J. Whelan, and P. Zitlau, "Nonpartisan Political Redistricting by Computer," *Operations Research* 13 (1965): 998–1006.
17. J. R. Birge, "Redistricting to Maximize the Preservation of Political Boundaries," *Social Science Research* 12 (1983): 205–214.
18. M. Shahmizad and A. Buchanan, "Political Districting to Minimize County Splits," *Operations Research* 73 (2024): 752–774.
19. R. Swamy, D. M. King, and S. H. Jacobson, "Multiobjective Optimization for Politically Fair Districting: A Scalable Multilevel Approach," *Operations Research* 71 (2022): 536–562.
20. P. Belotti, A. Buchanan, and S. Ezazipour, "Political Districting to Optimize the Polsby-Popper Compactness Score With Application to Voting Rights," *Operations Research* 73 (2025): 2330–2350.
21. A. Lara-Caballero, S. G. de-los Cobos-Silva, R. A. Mora-Gutiérrez, E. A. Rincón-García, M. Á. Gutiérrez-Andrade, and P. Lara-Velázquez, "Multiobjective Genetic Algorithms for Reinforcing Equal Population in Congressional Districts," *Mathematical Problems in Engineering* 2019 (2019): 1–14.

22. A. Ponsich, E. A. R. García, R. A. M. Gutiérrez, S. G. D. L. C. Silva, M. A. G. Andrade, and P. L. Velázquez, “Solving Electoral Zone Design Problems With NSGA-II: Application to Redistricting in Mexico,” *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, (2017), 159–160.
23. J. Simon and J. Lehman, “Antimander: Open Source Detection of Gerrymandering Though Multi-Objective Evolutionary Algorithms,” *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (2020), 51–52.
24. “Baker v. Carr, 369 U.S. 186,” (1962).
25. “Reynolds v. Sims, 377 U.S. 533,” (1964).
26. “Wesberry v. Sanders, 376 U.S. 1,” (1964).
27. B. Grofman and J. Cervas, “Recent Approaches to the Definition and Measurement of Compactness,” Available at SSRN 3919249 (2021).
28. K. Deb, K. Sindhya, and J. Hakanen, “Multi-Objective Optimization,” in *Decision Sciences*, ed. R. N. Sengupta, A. Gupta, and J. Dutta (CRC Press, 2016), 145–184.
29. H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, “How to Specify a Reference Point in Hypervolume Calculation for Fair Performance Comparison,” *Evolutionary Computation* 26 (2018): 411–440.
30. P. Efraimidis and P. Spirakis, “Weighted Random Sampling,” in *Encyclopedia of Algorithms* (Springer, 2008), 1024–1027.
31. D. DeFord, M. Duchin, and J. Solomon, “Recombination: A Family of Markov Chains for Redistricting,” *Harvard Data Science Review* 3 (2021): 3.

Appendix A: Objectives for Nondominated NSGA-II Plans

TABLE A1 | Nondominated plans in Cluster-NSGA-II (P_N).

Trial	Plan	Pop dev (PD)	Compactness (PP_s)	County splits (CS)
0	4	364 609	0.669	16
0	5	28 911	0.729	13
0	15	55 665	0.711	13
1	0	847 911	0.686	8
1	4	32 235	0.725	13
1	11	210 011	0.702	7
3	5	21 861	0.767	13
3	8	211 385	0.692	10
3	31	20 541	0.74	19
4	11	19 989	0.775	12
4	13	313 155	0.691	10
5	11	17 103	0.771	14
8	12	6577	0.809	19
8	16	16 587	0.771	18
9	29	24 739	0.71	18
10	3	48 163	0.737	9
10	7	22 561	0.725	15
11	13	36 489	0.748	10
11	14	39 359	0.716	11
11	17	12 335	0.775	15
12	11	171 749	0.719	8
12	19	7689	0.758	19

(Continues)

TABLE A1 | (Continued)

Trial	Plan	Pop dev (PD)	Compactness (PP_s)	County splits (CS)
12	21	7299	0.796	26
13	3	129 857	0.707	9
13	10	22 401	0.74	14
14	8	20 245	0.744	17
15	13	37 719	0.746	11
16	31	4263	0.824	27
17	11	27 365	0.757	13
17	15	31 615	0.767	11
18	1	166 537	0.74	7
18	4	211 321	0.694	9
19	0	89 829	0.72	8
20	1	56 751	0.725	8
20	3	60 427	0.714	12
20	10	43 995	0.747	9
21	16	64 403	0.702	15
22	3	153 563	0.696	10
22	6	30 887	0.789	10
24	16	220 209	0.683	11
25	17	416 399	0.659	9
25	22	211 137	0.689	14
27	6	15 189	0.783	13
27	8	483 799	0.692	7
27	12	47 537	0.739	10
27	22	15 407	0.74	22
28	5	77 869	0.691	11
29	33	15 005	0.75	22

TABLE A2 | Nondominated plans for Vanneschi et al. ($P_{V,VI}$).

Trial	Plan	Pop dev	Compactness	County splits	Source
0	0	7091	0.7	33	Van_VI
0	2	5833	0.785	31	Van_VI
13	0	100 579	0.675	32	Van_VI
13	5	63 081	0.682	28	Van_VI
13	6	54 695	0.684	23	Van_VI
13	7	75 847	0.681	30	Van_VI
13	8	40 351	0.685	32	Van_VI
13	9	91 313	0.679	30	Van_VI
13	11	39 319	0.689	28	Van_VI
19	0	1 441 335	0.671	24	Van_VI
22	1	1983	0.792	30	Van_VI
24	0	32 119	0.693	30	Van_VI
24	1	1611	0.805	27	Van_VI
24	4	25 421	0.698	26	Van_VI
24	5	25 471	0.694	30	Van_VI

TABLE A3 | Nondominated plans in Vanneshi et al. ($P_{V,SI}$).

Trial	Plan	Pop dev	Compactness	County splits	Source
3	3	11 341	0.695	26	Van_SI
3	6	23 397	0.69	25	Van_SI
3	7	20 427	0.691	25	Van_SI
4	0	3839	0.721	29	Van_SI
4	4	8143	0.708	28	Van_SI
4	5	8171	0.706	28	Van_SI
9	0	2107	0.774	31	Van_SI
9	2	2451	0.738	31	Van_SI
11	1	15 221	0.692	29	Van_SI
13	0	957	0.788	29	Van_SI
13	6	1485	0.785	29	Van_SI
13	7	1169	0.786	29	Van_SI
19	0	4993	0.71	24	Van_SI
21	2	23 405	0.689	24	Van_SI
21	5	28 727	0.676	23	Van_SI

Appendix B: Similarity Scores and Run Time**TABLE A4** | Similarity scores and run times for each trial in Cluster-NSGA-II.

Trial number	Pareto set size	Number of plan pairs	Similarity score	Run time (s)
0	24	276	0.611	50 172
1	24	276	0.626	44 204
2	36	630	0.612	49 104
3	35	595	0.630	51 589
4	32	496	0.618	49 189
5	25	300	0.636	52 589
6	24	276	0.613	50 584
7	34	561	0.617	48 149
8	27	351	0.623	50 524
9	33	528	0.620	60 031
10	22	231	0.630	55 419
11	22	231	0.613	57 614
12	23	253	0.626	47 909
13	16	120	0.628	49 664
14	32	496	0.632	55 000
15	37	666	0.617	54 569
16	36	630	0.622	60 038
17	39	741	0.624	48 503
18	36	630	0.622	50 361
19	28	378	0.621	62 017
20	23	253	0.630	55 107
21	39	741	0.625	55 853
22	33	528	0.608	54 894
23	42	861	0.620	58 134
24	30	435	0.632	53 990
25	36	630	0.627	52 118
26	48	1128	0.627	52 266
27	27	351	0.624	51 529
28	41	820	0.629	60 551
29	36	630	0.630	59 496

TABLE A5 | Similarity scores and run times for each trial in VNSGA-II VI.

Trial number	Pareto set size	Number of plan pairs	Similarity score	Run time (s)
0	4	6	0.946	109 535
1	5	10	0.644	113 090
2	16	120	0.910	126 832
3	14	91	0.740	110 050
4	12	66	0.833	120 216
5	7	21	0.986	109 283
6	18	153	0.769	119 839
7	10	45	0.716	96 093
8	11	55	0.763	129 612
9	15	105	0.732	134 932
10	20	190	0.897	129 211
11	8	28	0.904	103 786
12	10	45	0.755	100 174
13	13	78	0.982	104 987
14	10	45	0.776	102 495
15	6	15	0.806	102 892
16	19	171	0.821	100 670
17	21	210	0.792	105 288
18	24	276	0.827	100 925
19	5	10	0.769	95 958
20	10	45	0.803	103 929
21	13	78	0.853	104 515
22	9	36	0.814	103 881
23	11	55	0.950	106 086
24	6	15	0.955	98 613
25	11	55	0.981	100 091
26	16	120	0.877	106 564
27	14	91	0.738	98 949
28	11	55	0.650	102 999
29	9	36	0.715	111 061

TABLE A6 | Similarity scores and run times for each trial in VNSGA-II SI.

Trial number	Pareto set size	Number of plan pairs	Similarity score	Run time (s)
0	10	45	0.940	110 060
1	6	15	0.989	101 866
2	10	45	0.972	99 935
3	8	28	0.886	117 515
4	6	15	0.985	106 302
5	8	28	0.968	110 277
6	4	6	0.819	105 393
7	4	6	0.975	106 113
8	8	28	0.990	108 209
9	9	36	0.985	158 297
10	6	15	0.997	169 807
11	5	10	0.991	159 894
12	9	36	0.755	102 789
13	8	28	0.963	109 268
14	10	45	0.941	114 229
15	8	28	0.966	119 648
16	5	10	0.973	111 145
17	14	91	0.751	108 072
18	7	21	0.868	110 644
19	5	10	0.987	102 458
20	4	6	0.971	105 188
21	7	21	0.965	117 265
22	8	28	0.957	155 169
23	7	21	0.687	166 037
24	7	21	0.946	162 231
25	6	15	0.949	120 364
26	5	10	0.825	110 195
27	4	6	0.960	111 865
28	6	15	0.792	115 073
29	4	6	0.997	125 717