Swarm and Evolutionary Computation 30 (2016) 78–92

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

PEAR: a massively parallel evolutionary computation approach for political redistricting optimization and analysis

Yan Y. Liu [a,b,e,*], Wendy K. Tam Cho [c,d,e], Shaowen Wang [a,b,e]

[a] CyberInfrastructure and Geospatial Information Laboratory and Department of Geography and Geographic Information Science, 605 East Springfield Avenue Champaign, IL 61820, United States
[b] CyberGIS Center for Advanced Digital and Spatial Studies at the University of Illinois at Urbana-Champaign, 1205 West Clark Street, Urbana, IL 61801, United States
[c] Department of Political Science at the University of Illinois at Urbana-Champaign, 1407 W. Gregory Street, Urbana, IL 61801, United States
[d] Department of Statistics at the University of Illinois at Urbana-Champaign, 725 S. Wright Street, Champaign, IL 61801, United States
[e] National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, 1205 West Clark Street, Urbana, IL 61801, United States

ARTICLE INFO

Article history:
Received 7 January 2016
Accepted 28 April 2016
Available online 26 May 2016

Keywords:
Combinatorial optimization
Evolutionary algorithm
Redistricting
High-performance parallel computing
Heuristics
Spatial optimization

ABSTRACT

Political redistricting, a well-known problem in political science and geographic information science, can be formulated as a combinatorial optimization problem, with objectives and constraints defined to meet legal requirements. The formulated optimization problem is NP-hard. We develop a scalable evolutionary computational approach utilizing massively parallel high performance computing for political redistricting optimization and analysis at fine levels of granularity. Our computational approach is based in strong substantive knowledge and deep adherence to Supreme Court mandates. Since the spatial configuration plays a critical role in the effectiveness and numerical efficiency of redistricting algorithms, we have designed spatial evolutionary algorithm (EA) operators that incorporate spatial characteristics and effectively search the solution space. Our parallelization of the algorithm further harnesses massive parallel computing power provided by supercomputers via the coupling of EA search processes and a highly scalable message passing model that maximizes the overlapping of computing and communication at runtime. Experimental results demonstrate desirable effectiveness and scalability of our approach (up to 131K processors) for solving large redistricting problems, which enables substantive research into the relationship between democratic ideals and phenomena such as partisan gerrymandering.

1. Redistricting

Political gerrymandering in U.S. dates back to at least 1812 when the term was coined by the *Boston Weekly Messenger* in an editorial cartoon depicting an eccentric election district that bore an uncanny resemblance to a salamander, complete with a head, arms, and a tail. That year, the Massachusetts state senate districts were redrawn under Governor Elbridge Gerry to favor the Democratic-Republican party by consolidating the Federalist vote. By all accounts, the redistricting was successful; while the Federalists won both the Massachusetts house and the governorship, the senate remained firmly in the control of the Democratic-Republican party. The practice of bolstering political power by carefully orchestrating voters into meticulously crafted districts is as old as the founding of the country. Even at its nascence, the irony was glaring; in what is touted as the greatest democracy in the world, politicians are able to essentially hand-pick their voters. It is not always, as we might presume in a democracy, the voters choosing their representative.

Gerrymanders are not looked upon favorably, and the practice of gerrymandering is obviously controversial, but regulating the practice has proven quite challenging. Despite general disdain for gerrymandering, its existence as part of American democracy is long-standing. While the Supreme Court has established guidelines and mandates to protect our democratic system of elections, there are so many ways to draw gerrymanders within the legal constraints that the phenomenon continues, scarcely impeded by regulation. The tools simply do not exist to conduct the types of analyses that are required by the Supreme Court to overturn gerrymanders. Plainly, adherence to democratic ideals is simpler to articulate than it is to ensure.

An under-explored vantage point stems from the realization that the Supreme Court's vision for regulation is closely aligned

* Corresponding author at: CyberGIS Center for Advanced Digital and Spatial Studies and National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, 1205 West Clark Street, Urbana, IL 61801, United States.
E-mail addresses: yanliu@illinois.edu (Y.Y. Liu), wendycho@illinois.edu (W.K.T. Cho), shaowen@illinois.edu (S. Wang).

http://dx.doi.org/10.1016/j.swevo.2016.04.004
2210-6502/

with a heavily computational analytic approach. The natural fit of a computational approach and the legal mandates portends that a significant gain to the science of redistricting will be realized via computational approaches. While others have sought to combine the insights of operations research approaches with redistricting, the proposed methods have been either computationally intractable or exhibited limited computational capabilities, unable to facilitate advanced quantitative study of the substantive problem at hand. We propose an effective evolutionary algorithm (EA) that embeds spatial characteristics such as contiguity, compactness, and a hole-free property into the algorithm design. Conventional EA operators, such as crossover and mutation, can disruptively break spatial constraints such as contiguity and hole-free requirements in redistricting plans, making multi-objective optimization even more difficult. We address this issue by efficiently incorporating the spatial configurations of the underlying problem within each EA operator, while still effectively utilizing EA's stochastic manner for exploring the solution space. As a result, our sequential EA outperforms other selected heuristic algorithms (e.g., simulated annealing, tabu, and GRASP) in the literature for identifying better solutions in less time in our case study of redistricting in North Carolina.

Our *P*arallel *E*volutionary *A*lgorithm for *R*edistricting, PEAR, is designed to tackle computational complexity and meet the challenge of generating a large number of districting plans for analysis. The scalability of PEAR to both the problem size and the number of processors is achieved by coupling the EA with a highly scalable parallel EA message passing framework. PEAR scales up to 131K processors and efficiently leverages the massive computing power made available from national cyberinfrastructures such as the Blue Waters supercomputer.[1] This parallel computing solution collectively evolves independent EA instances through periodic migrations. A non-blocking asynchronous migration mechanism eliminates the global communication barrier and the resulting significant communication cost in massively parallel computing environments. The scalability of our algorithm brings two desirable benefits for redistricting analysis. First, as the number of processors increases, our algorithm identifies solutions with higher quality. Second, larger numbers of processors provide a larger pool of good and feasible solutions, which in turn enables insightful statistical analysis of redistricting phenomena. To the best of our knowledge, PEAR is the first redistricting algorithm that is scalable to over a hundred thousand processors.

This paper is organized as follows. In Section 2, we situate our contribution within the previous literature, which has broached many different aspects of the problem including heuristic approaches, the consideration of spatial characteristics in redistricting, as well as parallel evolutionary computation. In Section 3, we formally define the redistricting optimization problem and describe how our problem formulation can be adapted to the mandates of the Supreme Court. Section 4 presents the evolutionary algorithm and its associated EA operators. Section 5 presents the parallelization of the proposed evolutionary algorithm and discusses the strategy for improving the efficiency of our algorithm computation in a massively parallel computing environment. Section 6 presents experiments and results to evaluate our sequential algorithm by comparing it with previously proposed heuristic approaches in the literature. The results of scalability analysis on the parallel EA are then presented. We also describe the application of our approach for studying the partisan gerrymandering phenomenon. Finally, in Section 7, we discuss the implications of our approach for the practice of redistricting.

[1] http://bluewaters.ncsa.illinois.edu.

2. Literature review

Full enumeration approaches. The use of computers for redistricting was advocated many years ago [53,43,54]. The 1960s saw the advent of early computer programs for this purpose [43,24,52,18]. The first explorations of this type focused on full enumeration approaches where the objective was to examine all possible redistricting maps in order to contextualize the current plan as simply one plan among a large number of other possibilities [17,23,46,50,48]. Because the redistricting problem is so computationally complex, the applications for this approach were for only *very* small redistricting problems. Indeed, the implemented methods are infeasible and not generalizable for any problem of practical size. In the 1960s, though enthusiasm was high, progress was essentially halted by computing technology that was insufficiently advanced to permit nuanced and helpful guidance for actual redistricting problems.

Simulation approaches. The goals and research design for the latest round of redistricting research have followed the 1960s agenda closely. We are still enthralled by the full enumeration approach but are mindful that the computing capacity to achieve this goal still eludes us. As a result, research has re-focused on an obtainable and more feasible objective in the current computing environment. Instead of a full enumeration, the literature has shifted toward methods that are able to provide a large number of possible maps. The goal is to create a decision support system that allows one to contextualize a particular redistricting map by creating an ability to understand a range of possible redistricting maps for a particular geographic area.

Cirincione et al. [8] attempt to derive a sampling distribution of maps. They utilize four different algorithms for generating plans: a contiguity algorithm, a compactness algorithm, a county integrity algorithm, and a county integrity and compactness algorithm. Their four algorithms all begin by randomly selecting a block group to serve as the “base” of the first district. In the next step, depending on the algorithm, some criteria are used to select an unassigned contiguous unit to expand the district. This process is repeated until the population of the district reaches a desired size. The next district begins with the random selection of a block group from those adjacent to one of the completed districts. In all, they examine 10,000 of these generated plans (2500 from each of the four algorithms). Others have implemented a similar approach where districts are created by choosing an adjacent unassigned unit, considering the criteria of equal apportionment, contiguity, and compactness, though they produce a considerably smaller number of simulated maps. Chen and Rodden [7] appraise only a small and limited number of different plans (25, 250, and 200 different plans) to investigate questions of partisan bias in different states.

Integer programming and heuristics. Another strand of the literature capitalizes on the operations research literature on search heuristics. The gain here is rather than simply simulating maps that satisfy a minimal legal standard, one can search the space of legally viable maps and identify those that exhibit desirable characteristics. After all, the public, politicians, and the courts are interested in maps that have a positive impact on democratic rule, not simply maps that unintelligently satisfy a minimum set of legal guidelines. In this vein, various integer programming approaches have been implemented. Mehrotra et al. [40] developed a branch-and-price methodology. Macmillan and Pierce [38] implemented a simulated annealing algorithm for redistricting. Implementing a good search heuristic is non-trivial given the astronomically large number of feasible maps. Indeed, because of the computational complexity, Mehrotra et al. [40] were only able to apply their algorithm successfully to county-level data from South Carolina. Likewise, while Macmillan and Pierce [38] restricted their

algorithm to county data from Louisiana, Maine (16 counties and 2 districts), and New Hampshire (10 counties and 2 districts), they found the run times to be "intolerably long." Altman and McDonald [5] implement four metaheuristics: simulated annealing, greedy search, tabu search, and greedy randomized adaptive search [12,21,19,11]. The authors state that their program "may yield a useful improvement over the starting map and may further enable the public to generate constitutionally viable plans." They also state that additional computational power may be needed and may be achieved in a parallel computing environment by using the *snow* package to run their R code. They do not present results from an actual redistricting application.

Spatial characteristics. Preserving contiguity, compactness, political subdivisions and majority-minority groups requires proper representation of geographic information systems (GIS) data and maintenance of the proper spatial and geometric properties. In designing a search heuristic, care must be taken to store the necessary spatial and geometric attributes in appropriate data structures (e.g., graphs) since maintaining these spatial characteristics has an impact on virtually all of the operations. For instance, the contiguity constraint influences the choice of new solution generation and evaluation. Izakian and Pedrycz [29] use a spatial scanner to search a map for clusters in their particle swarm optimization algorithm and had to check the contiguity of each cluster when using centroid to include regions. In EA, traditional binary string genetic algorithm (GA) operators (e.g., crossover and mutation) tend to produce non-contiguous solutions [37]. Unless using a penalty function, non-contiguous solutions have to be avoided by using either contiguity-preserving operators or repair functions [41]. Xiao [55] summarized both approaches and applied them in a small redistricting case study. In [55,56], the mutation operator is designed to maintain contiguity by changing either the shape or the location of a set of contiguous and moveable units. For crossover, since redistricting is similar to graph partitioning, crossover operators designed for graph partitioning, such as those in Galinier and Hao [15], can be used in EAs for redistricting [55]. The crossover and mutation repair strategies in Xiao [55], however, are inefficient for large redistricting problems because they involve costly and non-deterministic randomized trials and extra spatial and geometrical operations. Moreover, repairing contiguity is also likely to adversely affect gains in other objectives and constraints such as population deviation and competitiveness.

Parallel evolutionary computation. The structure of EA/GA is fortuitously highly adaptable for exploiting high performance and parallel computing resources for the stochastic and iterative evolutionary computation. Basic EAs/GAs evolve an initial population through a "survival of the fittest" rule via a set of standard stochastic operators, i.e., selection, crossover, mutation, and replacement [27,13,21]. In a parallel computing environment, a population may be naturally divided into a set of sub-populations (also called demes or islands) that evolve and converge with a significant level of independence. In this vein, various parallel EAs (PEAs) have been developed and applied to a broad and rich set of application domains [4,33,28,26,47,42]. Alba and Troya [2] showed that PEA not only improves computational efficiency over sequential EAs, but also facilitates more extensive exploration of the solution space, resulting in a larger and better set of solutions. Ocenasek and Pelikan [44] analyzed the complexity and scalability of parallel estimation of distribution algorithms, a new paradigm of evolutionary algorithms. Though much efficiency can be gained with additional processors, one must also be wary of the substantially increasing communication costs that arise with additional processor cores [30,10]. Synchronized migration has been linked to serious performance degradation on PEAs. As a result, asynchronous inter-deme interactions have been recognized as desirable for designing scalable PEAs [1,3,25,35].

In summary, computational complexity is formidable and a major bottleneck for redistricting analysis. A full enumeration of plans is still not possible, and investigations at fine levels of geographic granularity have proven exceedingly difficult. How to traverse the space of possible redistricting maps is not straightforward. The availability of massive computing power provided by supercomputers provides an avenue for the development of scalable heuristics that are able to efficiently exploit massively parallel high-end computing resources to find better solutions and create a large number of unique feasible solutions for further statistical study. This is the task that we tackle here. We develop a highly scalable parallel evolutionary computation approach that intelligently explores the space of possible maps and efficiently extracts high quality feasible maps that satisfy flexible redistricting criteria.

3. Redistricting analysis: a new computational approach

Drawing electoral maps amounts to arranging a finite number of indivisible geographic units of a study area (e.g., a U.S. state) into a small number of larger areas. For simplicity, call the former *units*, the latter *districts* or *zones*, and the study area *region*. Since every unit must belong to exactly one district, a districting map is a partition of the set of all units into a pre-established number of non-empty districts. The redistricting problem is an application of the set-partitioning problem that is known to be *NP*-complete and, thus, computationally intractable [16]. The total number of possible maps when drawing K districts using N units is a *Stirling number of the second kind*, $S(N, K)$, defined, combinatorially, as the number of partitions of an N-element set into K blocks [31]. Even with a modest number of units, the scale of the unconstrained map-making problem is awesome. If one wanted to divide $N=55$ units into $K=6$ districts, the number of possibilities is 8.7×10^{39}, a formidable number.[2] At the census block level, California has 710,145 census blocks for 53 districts. Pointedly, the number of possibilities in actual redistricting problems is of staggering proportions, creating a prohibitively large computational problem. The large problem size and the computational complexity involved when working with fine-grained redistricting data has plagued the progress of tool development for redistricting analysis.

3.1. Problem formulation

Solution space. The solution space in a redistricting problem is not characteristic of a rugged solution space. While the space landscape is hilly in the sense that it has the usual peaks and valleys, these peaks and valleys are not a rapid succession of precipices, but instead, a series of vast plateaus, and hence, not rugged in the traditional sense. These expansive plateaus manifest themselves throughout the landscape because many possible redistricting plans are extremely similar—it is readily evident that moving a single census block from one district to another does not induce much change, and there are a slew of such minor modifications to any redistricting plan. This idiosyncratic surface type coupled with the prohibitive size of the solution space behooves a tailored search algorithm that is able to make large moves within the solution space and has some mechanism to guide the search toward distinct areas in the solution space.

Problem variables. The goal of the redistricting problem is to identify a plan that optimizes one or more selected objectives (e.g., competitiveness, safe districts, and incumbent protection) while

[2] The numbers are taken from the 55 counties and 6 congressional districts in West Virginia, USA.

simultaneously satisfying legal constraints (e.g., contiguity and equipopulous districts). There are many ways to specify an objective function. A linear programming formulation of the problem might proceed as follows. We have a set of N geographic units, $u_1, u_2, \ldots, u_N$, that we wish to partition into a set of K districts/zones, $d_1, d_2, \ldots, d_K$. We can create an $N \times N$ adjacency matrix, $\mathbf{C}$, to indicate the adjacency of the various units, where the entries are defined as

$$c_{ij} = \begin{cases} 1 & \text{if unit } i \text{ and unit } j \text{ are adjacent or } i = j \\ 0 & \text{otherwise} \end{cases}$$

for $1 \le i \le N$ and $1 \le j \le N$. The convention that $c_{ij} = 1$ for $i=j$ is adopted to simplify the checking of connectedness of districts. The population of the N units is denoted by $p_1, p_2, \ldots, p_N$. So, if the districts are equipopulous, then the population in each district would be the average population, $\overline{P}$, given by

$$\overline{P} = \frac{1}{K}\sum_{i=1}^{N} p_i.$$

Let $\mathbf{X}$ be an $N \times K$ matrix with elements, x_{ik}, denoting our decision variables. To specify a map, these variables are chosen for $1 \le i \le N$ and $1 \le k \le K$ so that

$$x_{ik} = \begin{cases} 1 & \text{if unit } u_i \text{ is assigned to district } d_k \\ 0 & \text{otherwise.} \end{cases}$$

The population in district k is then

$$P_k = \sum_{i=1}^{N} x_{ik}\, p_i \quad \text{for } k = 1, 2, \ldots, K.$$

Constraints. We have constraints of at least three types.

1. Each unit must be assigned to exactly one district,
$$\sum_{k=1}^{K} x_{ik} = 1 \quad \text{for } i = 1, 2, \ldots, N.$$
2. The maximum population deviation across all K districts is no greater than a specified value M. That is, for any two districts, d_i and d_j,
$$|P_{d_i} - P_{d_j}| \le M \quad \text{for } i, j = 1, 2, \ldots, K.$$
To define the population deviation across all districts, we may formulate population criteria as
$$p = \frac{\max_k (P_k) - \min_k (P_k)}{\overline{P}}. \tag{1}$$
This measures the level of population deviation between the set of K districts. When the districts have identical population, $p=0$. As their population increasingly deviates from one another, the value of p increases. In this formulation, it is possible for p to exceed 1. This occurs when the difference in population between districts is sufficiently large. In these cases, we set p to its maximum value, 1, which already represents an extreme population difference.
3. The units in each district must form a connected set. That is, each unit is accessible from any other in the set via transitions encoded in the adjacency matrix $\mathbf{C}$.

Other constraints include the hole-free property and compactness. The hole-free property requires that no district is wholly contained in another district. Compactness is encouraged and valued. However, since it is neither uniformly enforced by the courts nor strictly defined, it has taken on various specifications. A popular conceptualization, via an area–perimeter criterion, which compares the perimeter of a shape to the area of the shape, was first proposed by Ritter in 1882 [14]. With the area–perimeter compactness, a circle is the most compact shape and would have an area-to-perimeter ratio or compactness value of 1. The value of a simple area-to-perimeter ratio would vary with the size of the shape, but we can create a scale invariance by dividing the area by the square of the perimeter. We may also normalize the measure to have values in the $(0, 1]$ range by including π in the numerator. These changes result in one of the most widely used compactness measures in the area–perimeter class of measures, C_{IPQ} [45], which is defined as

$$C_{IPQ} = \frac{4\pi A}{P^2}. \tag{2}$$

In our minimization formulation, we use $(1 - C_{IPQ})$ as the compactness measure. There are many ways to constrain shape or define compactness. An area–perimeter approach is only one such method. A shape may also be compared to a reference shape. We might use geometric pixel properties. Alternatively, other criteria may be based on the dispersion of elements in the area [34].

Objectives. Subject to the constraints above, we seek to optimize a single or multiple objectives. In a multi-objective scenario, we implement a weighted sum to incorporate multiple objectives in our solution fitness evaluation. The particular specification of the objective function is flexible, simply depending on the particular substantive interest. If one were interested in optimizing competitiveness, for instance, one might proceed as follows. Let D_k be the Democratic registration in district k and R_k be the Republican registration in district k for $k = 1, 2, \ldots, K$.[3] Assume that a district is most competitive when $D_k=R_k$. When all districts are considered, an overall measure of competitiveness in a map could be calculated as

$$f = T_p\big(1 + \alpha T_e\big)\beta, \tag{3}$$

where

$$T_p = \frac{1}{K}\left(\sum_{k=1}^{K}\left|\frac{R_k}{D_k + R_k} - \frac{1}{2}\right|\right),$$

$$T_e = \left|\frac{B_R}{K} - \frac{1}{2}\right|, \quad \text{for } 0 \le T_p \le 0.5, \;\; 0 \le T_e \le 0.5, \;\; \text{and } \alpha, \beta \ge 0.$$

Here, T_p measures competitiveness as a deviation of the Republican two-party registration from 0.50 in each district and T_e is a weighting factor, which captures the differential in the number of seats won by the two parties. In the formulation for T_e, B_R is the number of districts where Republican registration is larger than the Democratic registration; α defines the weight of T_e in the competitiveness measure; and β is a normalizing constant so that the value of f spans the range $[0, 1]$. For example, since $T_e \in \left[0, 0.5\right]$ when $\alpha = 1$, if we set $\beta = \frac{4}{3}$, then $f \in [0, 1]$.

This formulation with both the T_p and T_e components provides two layers of differentiation. By making T_e a weighting factor for T_p, we can simultaneously encourage the average registration differential, T_p, to be small while ensuring that this small differential is spread equally between the parties. Under our formulation for competitiveness, when $f=0$, Republican registration and Democratic registration is the same and the number of districts where Republicans dominate and the number of districts where the Democrats dominate are identical.

[3] The user may decide how best to measure partisan bias (with registration data, turnout data, presidential vote data, etc.), a substantive debate on which we take no position.

3.2. Formulation flexibility

The competitiveness, population, and compactness formulations are simply one of many ways to define constraints and objectives. One nice feature of our particular measures, shown in Eqs. (1)–(3), is that the values are normalized so that they span the same [0, 1] interval with 0 being the desired or optimal value. When the criteria are normalized in the same range, it simplifies the weight specification in a multi-objective function that encompasses measures reflecting competing interests that are ideally and simultaneously satisfied.

The criteria that are optimized are user-specified and may reflect any aspect of a redistricting plan that the user finds helpful to constrain or explore. In one specification, we could consider competitiveness as the sole objective, and population deviation and compactness as constraints. We may also consider optimizing (as a minimization problem) competitiveness, population deviation, and compactness while simultaneously treating population deviation as a constraint, by specifying a maximum threshold. While these choices are far from the only way in which one might guide such a computational model, they comprise reasonable examples of a real-world redistricting scenario that features the interplay of political interests and traditional districting principles. Once measures of the individual criteria are specified, the user may deploy any desired customized notion of how various criteria should be weighted. These specifications are modular, flexible, and customizable across a wide set of preferences, interests, and constraints.

4. Evolutionary algorithm

Once the objective function is specified, we proceed to search for solutions that exceed a user-defined threshold of goodness. There are two main criteria for the design of a search algorithm. First, a stochastic element helps the search avoid being trapped in local optima. Second, since the solution landscape for redistricting maps is characterized by a series of vast plateaus, the algorithm must be able to make large jumps from one plateau in the solution space to another. Evolutionary searches are able to satisfy both of these criteria: (1) population-based heuristic search algorithms such as EA are always characterized by a strong random/stochastic element, which helps the search process avoid being trapped in areas with local optima and (2) the crossover and mutation operator, in general, are intended to create large movements from one solution to the next. We develop an EA as a general redistricting solver. We employ efficient data structures and design effective EA operators to achieve these criteria for redistricting and handle spatial and non-spatial objectives and constraints. More importantly, the parallelization of our EA provides a scalable computational approach to employ a large number of processes that can simultaneously work on many plateaus and jump from one to another through inter-process communication.

4.1. Encoding and data structure

Similar to conventional binary string encoding, the basic data structure of the EA, the chromosome, is encoded as an integer array where each allele, indexed by the unit number, holds the zone (the terms *zone* and *district* are interchangeably used hereafter) number that is assigned to each of the geographic units. An example is displayed in Fig. 1, where we can see, for instance, that both geographic units 1 and 2 are assigned to zone 8 while geographic unit 3 is assigned to zone 5. Every geographic unit is assigned to exactly one zone.

In addition to the binary string encoding of a solution/chromosome, we maintain several spatial data structures to enable spatial constraint checking and operations. We store two graphs with auxiliary indexing data: the unit graph and the zone graph. Consider an example redistricting problem derived from partitioning the 87 county geographic units in Minnesota into its 8 congressional districts. The unit graph, which is also the county map, is shown on the left in Fig. 2.[4] We first convert the geographic data/map into a network graph with vertices and edges where the vertices are the counties and an edge exists whenever two geographic units are adjacent. We also define a virtual unit 0 which is a polygon derived by subtracting the shape of the region (state) from a rectangle that contains the region border. Unit 0 is particularly useful in identifying units on the region border. The adjacency structure is shown on the right in Fig. 2 where the blue lines indicate rook adjacency and the red lines indicate queen adjacency (unit 0 is not shown). Since both rook and queen adjacency fall within the legal definition of contiguity, we encompass both in our adjacency matrix. A zone graph, *Gz*, which is a $K \times K$ 0–1 matrix, stores the adjacency of the districts. The zone graph is used to check if a district is contained in another district, thus creating a "hole" in the redistricting map. Similar to the definition of unit 0, zone 0 is a virtual district on the zone graph that has only one unit, which is unit 0. To handle large problems, the unit graph is stored as a linked list, instead of a 2-D array (the number of edges of a planar graph is a linear factor of the number of vertices). Auxiliary data structures are also used to help accelerate sorting operations on unit attributes loaded from GIS data.

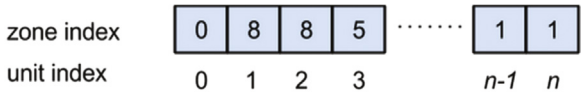

Fig. 1. Chromosome encoding.

Operations on the aforementioned data structures form the basis for handling the spatial configurations of the redistricting problem in the EA operators. These operations include, for example, finding neighborhood units, identifying the border units of a district, contiguity checking, hole checking at the district level, and compactness updating. We now describe how these operations are designed efficiently to prevent excessively hampering the EA's numerical performance.

Contiguity checking. As we form districts, we must ensure that all formed districts are contiguous and that all the units within each district remain contiguous. At the district level, the small size of the zone graph makes this check simple and straightforward. At the unit level, the check is non-trivial since the associated computational cost increases dramatically as the number of units grows. King et al. [32] described this cost and proposed a geograph approach to resolve the performance issue. Rather than performing the check after a new solution is generated, we adopt a strategy that integrates contiguity checking within the EA operators. The correctness of our strategy is guaranteed by the following two rules:

1. An initial solution satisfies the contiguity constraint; and
2. Any changes to the solution created by unit movement (from one district to another) does not lead to a non-contiguous solution.

While the second rule seemingly imposes a strong condition for the design of EA operators (e.g., crossover and mutation) to satisfy, we find that maintaining contiguity within our EA operators

[4] Our algorithm is intended to be utilized at fine levels of granularity (e.g., units being census tracts or census blocks), however, because it is difficult to visualize these small units, we illustrate our EA at the county level in this section.

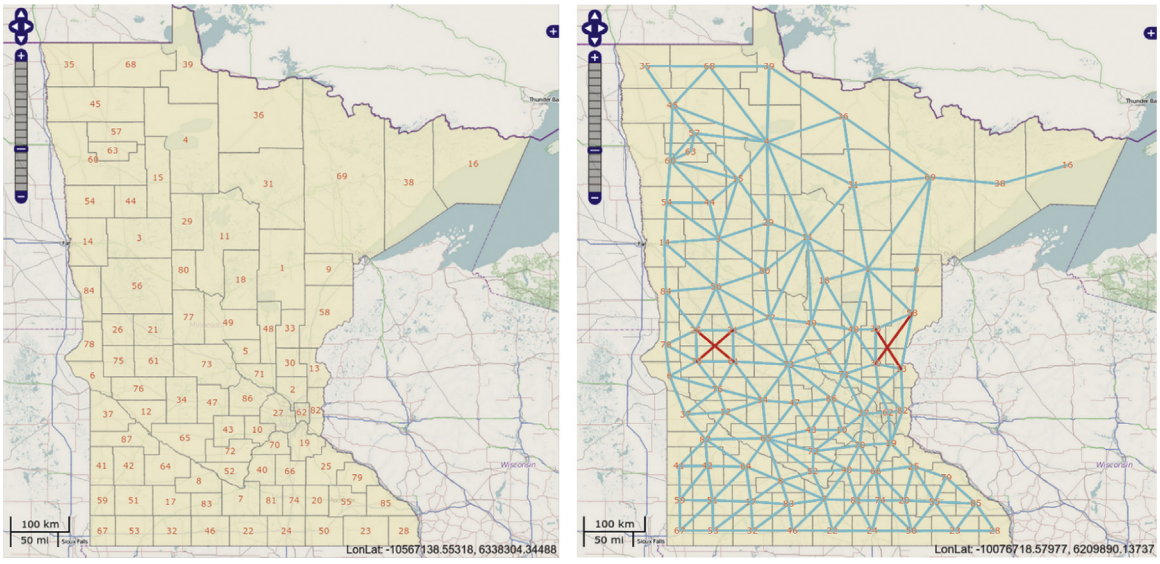

Fig. 2. Neighborhood representation. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

avoids the expensive solution-level contiguity checking and the often failed contiguity repair operation. The capability of traversing the solution space is still maintained by introducing randomization at the unit selection stage of each EA operator, as described in Section 4.2. To determine whether a move breaks contiguity, we check if all of the units in a district remain contiguous when a subset of units in the district is removed. The pseudocode for our contiguity checking is shown in Algorithm 1.

Algorithm 1. Contiguity checking.

```
1  U: set of units to be removed
2  S: a solution
3  Check that at least one unit in U is on zone boundary
4  V = { } // units that are neighbors of U and in the same
   zone
5  foreach u ∈ U {
6    foreach neighbor, v, of unit u {
7      if (v is in the same zone in S)
8        V = V ∪ {v}
9    }
10 }
11 G = { }; // connectivity graph as adjacency list
12 foreach u ∈ V {
13   G[u] = { };
14   foreach neighbor v of u {
15     if (v ∈ V)
16       G[u] = G[u] ∪ {v}
17   }
18 }
19 Starting from any unit, count the number of connected
   nodes on G, c
20 return (c > 0 and c == length(V));
```

Hole checking. Before a new solution is generated for evaluation, we must ensure that holes (i.e., when a zone is contained within another zone) are not created in the redistricting map. The hole checking function examines the zone graph matrix for zones with only one neighbor. If the neighbor of such a zone is not zone 0, the hole-free requirement is violated. If the only neighbor is zone 0, this zone is isolated, violating the district contiguity requirement. This case is unlikely unless the EA operations are improperly coded.

Compactness calculation. In our formulation, the worst compactness value among all districts in a solution determines the compactness for the redistricting map as a whole. To calculate the compactness of an individual district efficiently, we do not store the geometric information (i.e., the shape and coordinates) for each individual unit. Instead, we store the area and perimeter of each unit, and border length between any two adjacent units. This allows us to calculate the area of each district by summing the area of all the units in the district. The perimeter of a district is the summation of the border length between any two adjacent units in which only one unit belongs to the district. Calculating the perimeter for all districts can be efficiently accomplished with a dynamic programming approach where each unit–unit border is visited only once. This approach avoids expensive operations involving geometry such as a spatial join. Instead, the only cost comes from the preprocessing of the shape of each unit, which is done only once before the execution of the EA.

4.2. Evolutionary algorithm operators

An EA algorithm includes a set of operators that generates the initial population, produces new solutions from parent solutions, and evaluates them. For redistricting, the objectives and constraints are tightly coupled with a districting map's spatial configuration. As a general principle, each move in our EA operators maintains contiguity while still using randomization to traverse the solution space. This strategy ensures greater numerical efficiency, an important consideration in implementing these operators.

Population initialization. The EA algorithm begins by generating a set of initial solutions to form the initial population. To generate an initial solution, we first randomly select K seed units and then expand them to K districts by iteratively including a random number of direct neighbors. This iterative unit-expanding method guarantees contiguity. Seeding proceeds either via administration boundary or by region border. In the first strategy, K seeds are selected such that each belongs to a different higher-level administration boundary (e.g., county for voter tabulation district level redistricting). This strategy generates more compact solutions but may lead to maps with holes (in which case, we simply ignore and begin the process again to avoid expensive repair

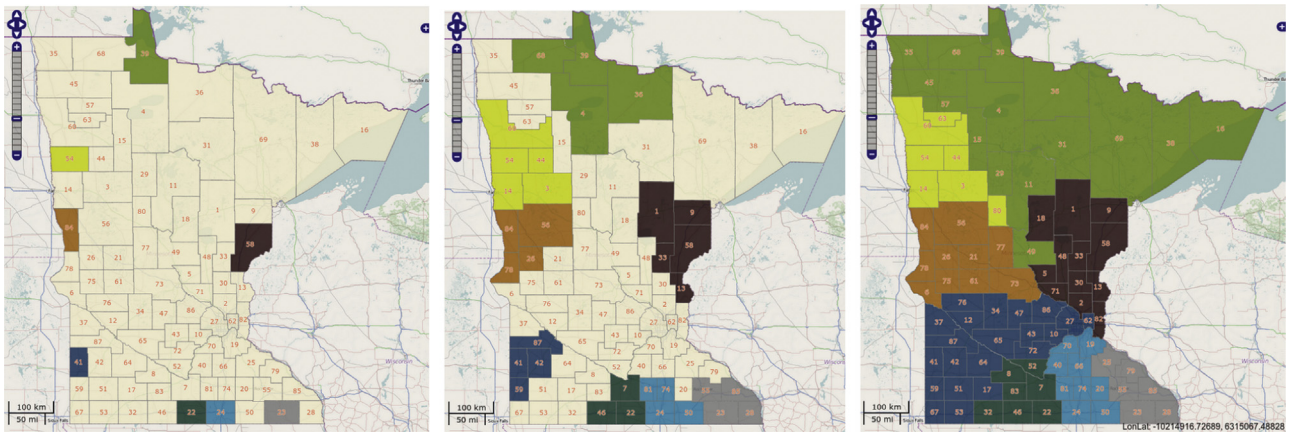

Fig. 3. Solution initialization.

procedures). The second strategy randomly selects K seeds on the region border, which guarantees the hole-free requirement since districts touching the region border cannot be completely surrounded by any other district. In the algorithm, a probability value is specified for each strategy to be applied in the population initialization.

The second strategy for initial solution generation is illustrated in the series of maps shown in Fig. 3. We begin by randomly choosing 8 geographic edge units to be our seeds. An example is shown in the leftmost map of Fig. 3. We then expand these seeds by selecting contiguous units randomly (center map in Fig. 3) until we have all 8 connected districts composed of contiguous geographic units in each (rightmost map in Fig. 3). The result is inserted into the initial population as a feasible solution. This process is repeated until the initial population is fully populated. The number of solutions in the initial population can be set to any value. The default is 200. The corresponding pseudo-code appears as Algorithm 2.

Algorithm 2. Generating a feasible contiguous solution.

```
1  // choose K initial seeds
2  Seeds = ∅
3  Get the list of units neighboring unit 0, N_v
4  while (|Seeds| < K) {
5    Randomly select a unit u from Nv
6    if (u is not selected before)
7      Seeds = Seeds ∪ {u}
8  }
9  // generate a solution from seeds
10 int soln [n] // initialized to zeros
11 Pool=Seeds
12 while (Pool ≠ ∅) {
13   Pop a unit u from Pool
14   Get the neighbors of u, N_u
15   foreach unit v in N_u {
16     if (soln [v] = 0 and v ∉ Pool)
17       Pool = Pool ∪ v
18       soln [v]=soln [u]
19   }
20 }
21 return soln
```

Fitness evaluation. The fitness value of a solution is the same as the objective function value, which is flexibility configured to be single- or multi-objective via a weighted sum formula. Contiguity and population equality must be satisfied by law.[5] We incorporate this legal requirement by considering a solution to be feasible if its population deviation is below a specified threshold. We operationalize this concept by calculating an unfitness value for each of our solutions. The unfitness value is zero when the population equality across the set of districts is within the threshold value and is otherwise the value of the population deviation measure. Since contiguity is required for all solutions, non-contiguous solutions are discarded. In this way, our formulation incorporates constraints that, unlike the objective, must be satisfied by any solution. Each solution has a fitness and an unfitness value, which facilitates flexible replacement strategies (either through a penalty function or other algorithmic logic).

Mutation. The spatial mutation operator is designed to maintain contiguity while leveraging randomization to select units for district reassignment. It has two procedures:

1. *Shift* moves a number of units from one district to a neighboring district. To ensure that a shift does not violate contiguity, the selected units include at least one unit on the boundary of the sending and receiving district.
2. *Mutate* makes a sequence of shifts to balance metrics such as population deviation. This sequence may have one or more cyclic shifts.

In *shift*, we randomly select a boundary unit and then expand it randomly to form a set of units to be moved. In *mutate*, we randomly choose a sequence of districts and randomly choose the number of cyclic shifts. The spatial mutation algorithm is outlined in Algorithm 3. The parameter *maxMutUnits* is set at runtime. Larger redistricting problems should use a larger value in order to avoid small and ineffective moves.

Algorithm 3. Shifting mutation.

```
1  // Multi-unit mutation process that runs K times to make a
   chain of shifts
2  mutate(solution) {
3    Generate a random sequence s of size K using the Fisher-
   Yates algorithm;
4    foreach zone z ∈ s in solution {
5      if (z is a source zone of previous shifts) continue
6      if (population in z is below a threshold) continue
```

[5] *Reynolds v. Sims*, 377 US 533 (1964), created the mandate of one man, one vote. Though strict equality is not demanded, there is no *de minimus* deviation that is allowable (*Karcher v. Daggett*, 462 US 725 (1983)), and minimizing the population inequality is a legal requirement.

```
      Randomly select a dstZone from neighboring zones of z
      shift (z, dstZone)
    }
  }
  // Select a subset of units from the source zone to the destination zone
  // srcZone: source zone to send the selected units
  // dstZone: destination zone to receive the selected units
  shift (srcZone, dstZone) {
    Randomly select up to two adjacent units in srcZone bordering dstZone;
    subZone = selected units;

    while |subZone| < maxMutUnits {
      Find neighbor units U of subZone;
      Generate a random number q ∈ 1, 2, ..., |U|;
      Randomly choose a subset U' ⊂ U, where |U'| = q;
      subZone = subZone ∪ U'.
    }
    dstZone = dstZone ∪ subZone;
    srcZone = srcZone − subZone;
  }
```

Crossover. Just as the conventional mutation operator is not suitable for spatial configuration, the binary string crossover operators must also be modified to take spatial considerations into account. Our crossover operator, shown in Algorithm 4, overlaps two redistricting maps (e.g., solution A, the first map in Fig. 4 and solution B, the second map in Fig. 4), resulting in a set of intersected subzones/splits, shown in the third map, solution C, in Fig. 4. Solution A has districts $A_1, A_2, \ldots, A_8$. Solution B has districts $B_1, B_2, \ldots, B_8$. The third map will have between K and K^2 split labels, C_{ij}, where $i \in \{A_1, A_2, \ldots, A_8\}$ and $j \in \{B_1, B_2, \ldots, B_8\}$. That is, each split label is formed from exactly one district from solution A and one district from solution B. However, depending on the district shapes, different splits may share a common label. We subsequently relabel them as new splits.

The splits shown in the third map form a new split-level redistricting problem. The number of districts in the new problem is still K. We form a new solution by treating splits as units via the same population initialization strategies which will combine these splits into a feasible solution of K districts. Since each unit belongs to only one split, it is straightforward to convert this split-level solution to the unit level. The new solution is then returned as the output of the crossover operator. The last map in Fig. 4 shows Solution D, the map after the crossover. It is worth noting that, while the crossover is able to generate large movements to new solutions, likely in different parts of the solution space, our empirical tests indicate that this process may be sufficiently disruptive to the desirable attributes of the underlying maps that the resulting child map may not be of higher quality than the parent maps. Solution D seems to represent such a case. Accordingly, we apply the crossover operator with relatively low probability. This permits larger moves in the solution space while maintaining reasonable numerical performance and EA convergence.

Algorithm 4. Crossover.

```
1  crossover (solution1, solution2) {
2    // Identify splits by assigning unique index to each split
3    foreach unit u {
4      Calculate split label as z1z2, where z1 is the zone index of u
5      in solution1, and z2 is the zone index of u in solution2
6    }
7    Group units with the same split label into the same split, form set Splits
8
9    // handle the case: two geographically separated splits are assigned with same index
10   newSplits = ∅
11   foreach split s ∈ Splits {
12     while s ≠ ∅ {
13       Find a unit u in s, construct a spanning tree in s, rooted at u
14       Form a new split s' to include all of the units on the spanning tree
15       newSplits = newSplits ∪ s'
16       Remove all the units in s' from s
17     }
18   }
19
20   // solve the split-level redistricting problem
21   Construct split graph (rook and queen neighborhood) from newSplits
22   Check holes and repair
23   Formulate a new redistricting problem
24   Generate a solution using the 2nd seeding strategy in the population initialization operator
25   Convert split-level solution to unit-level solution soln
26
27   return soln
28 }
```

After the mutation and crossover, a new solution, if its population deviation is above the defined threshold, goes through a fine-tuning procedure that examines the population difference between any pair of connected districts by checking the zone

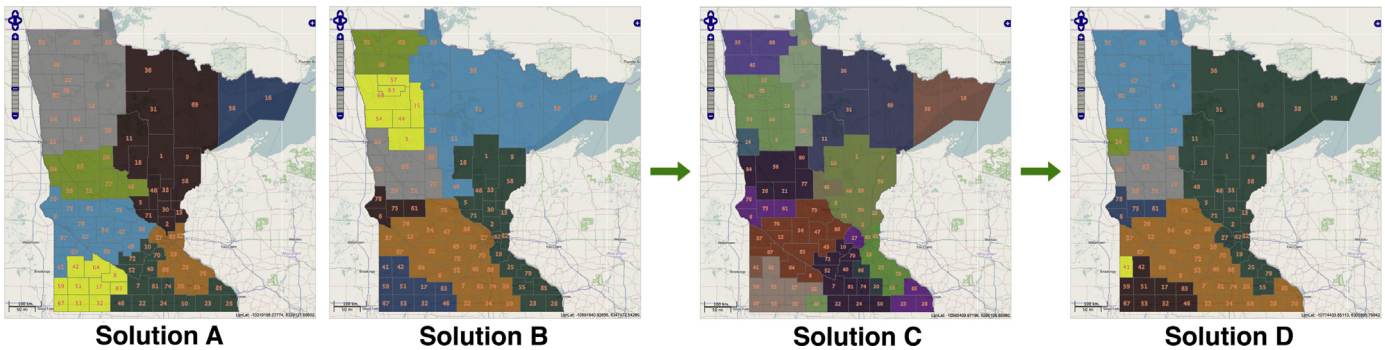

Fig. 4. Crossover operator.

graph. If necessary, some units are exchanged between a pair of districts to balance the population in a randomized way. The last step in an EA iteration is to evaluate the resulting solution and apply a replacement strategy to update the population. Both the fitness and the unfitness are considered in the replacement strategy. An existing solution with the worst fitness or unfitness is replaced if a new solution is better. If a new solution is better than the current best, it becomes the elite solution. Therefore, our EA is a *steady-state* EA [21] in which each iteration selects two parents to generate one child for replacement.

5. Parallelization

While the sequential EA algorithm efficiently incorporates the spatial characteristics of the redistricting problem in its formulation, the solution space for the redistricting problem is idiosyncratic, astronomically large, and characterized by sprawling plateaus. Traversal of this type of solution space requires significant computation that can be aided by employing parallel computing on a large number of processors. PEAR incorporates a migration strategy that exchanges solutions between any two directly connected islands (processes) at regular intervals. To ensure each evolutionary process is independent, each island uses a unique random number sequence generated by SPRNG [39]. The use of SPRNG in the message passing (MPI) model ensures that no two processes would repeat the same random number sequence (starting two processes with different seeds is not enough since they may use the same random number sequence) and, thus, exhibit similar search paths when running independently without external random noise. This is particularly important when a large number of processes run in parallel. Liu and Wang [36] develop a scalable PGA library with a suite of non-blocking migration operators (i.e., *export*, *import*, and *inject*) to eliminate the need for a global barrier in migration communication. PEAR extends this library to enhance the parallelism control by handling application-level sending and receiving buffers for non-blocking message passing. The regular migration of elite or random solutions on all of the islands enables simultaneous exploration of a large number of solution space plateaus as well as movement from one plateau to another through elite solution propagation and the collective but independent evolutionary searches surrounding these elite solutions. The asynchronous migration strategy maximizes the overlapping of computation and migration communication and removes prohibitively costly global synchronization in a massively parallel computing environment. Table 1 lists the configuration of our EA and PEA.

In Liu and Wang [36], the relationship between the configuration of the PGA parameters (i.e., migration intervals, migration rate, and topology attributes) and buffer sizes is established based on the underlying message passing communication library and supercomputer interconnect characteristics to avoid buffer overflow issues at the system and application levels. It also provides explicit programming control on the parallelism of the import operator by configuring the import buffer size and the interval for invoking the import operator in GA iterations. Further experiments, however, observed MPI communication layer failure when scaling from 16K processors on the Stampede supercomputer to larger numbers of faster processor cores on supercomputers such as Blue Waters (with more than 700K integer cores). With more processors, the outgoing message buffer, controlled by the MPI, experienced buffer overflow as message sending from the export operator become seriously skewed among PGA processes due to the outpaced runtime delays from numerical operations and non-blocking sending and receiving. Consequently, we extended the PGA library to manage the sending buffer at the application level and explicitly specified the degree of overlapping between EA iterations and message sending. Our enhanced PEA framework is shown in Fig. 5. The improved library has been tested and scales well on the Blue Waters supercomputer without failure and with marginal communication cost. This extension has minimum impact on PEAR's numerical performance. The communication cost remained consistently at around 0.015% with 16,384 processor cores on Blue Waters. Such scalability provides a dramatic increase in numerical capability, especially notable and consequential since experiments using synchronous migration exhibited an increased communication cost of 41.38%.

6. Evaluation

We evaluate our computational approach from three perspectives. First, we examine our base sequential algorithm by comparing its solution quality to that obtained by other heuristics. This comparison allows us to evaluate the effectiveness and numerical efficiency of our spatial EA operators. Second, we evaluate the

Table 1
PEAR parameter settings.

Parameter	Value
Population size per deme	200
Initial population	80% by region border, 20% by administration boundary
Selection	Binary tournament
Spatial mutation	Probability: 0.95; *maxMutUnits*: 15
Spatial crossover	Probability: 0.05
Repair	District population balancing on infeasible solutions
Replacement	Replacing the worst (fitness) or the unfittest
Elitism	Yes
Stopping rules	No solution improvement, bounded solution quality reached, fixed number of iterations, execution walltime
Process topology	2-D Torus
Number of islands per processor	1
Process connectivity d	4 (number of direct neighbors of each process)
Migration rate r	2 (number of solutions sent to neighbors in each export)
Export interval M_{expt}	50 (number of iterations between two consecutive exports)
Import interval M_{impt}	25 (number of iterations between two consecutive imports)
Sending parallelism	2 (number of exports to invoke before waiting for their finish)
Sending buffer size $K_{sendbuf}$	2 solutions. Actual memory requirement is $(2 \times n \times 4 + \text{buffer\_overhead})$ bytes
Receiving buffer size K_{impt}	8 solutions. Actual memory requirement is $(8 \times n \times 4)$
Multi-objective fitness function	Scalability test: $0.8 \times \text{competitiveness} + 0.2 \times \text{population\_deviation} + 0.3 \times \text{compactness}$

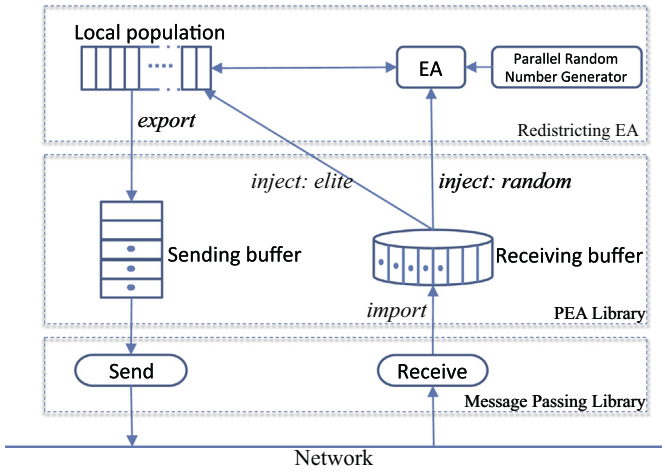

Fig. 5. Enhanced PEA framework.

parallel component by examining the scalability of the PEAR algorithm by quantifying the numerical work that is completed by different numbers of processors. Third, we take note of the large number of feasible solutions that are generated in the scalability test, consider a statistical analysis in the redistricting context, and discuss how the maps inform an investigation of partisan gerrymandering.

6.1. Implementation and case study

Our EA is implemented in ANSI C. It can be compiled on Linux, OS X, and Windows as a standard *makefile* project. PEAR uses MPI non-blocking functions (i.e., *MPI_Ibsend()*, *MPI_Iproble()*), and regular *MPI_Recv()* for asynchronous migration. It uses the C SPRNG 2.0 library to provide a unique random number sequence for each MPI process, which is necessary for running a large number of EA iterations. The evaluation of our sequential EA is tested on the ROGER supercomputer. The scalability of our code is tested against MVAPICH2 with up to 16,384 cores on the Stampede supercomputer and the Cray MPI with up to 131,072 integer cores on the Blue Waters supercomputer. Each node on ROGER is configured with the Intel Xeon E5–2660 processor (2.6 GHz, 20 cores/node). Stampede employs the Intel Sandy bridge processor (2.7 GHz, 16 cores/node). Blue Waters utilizes the AMD Bulldozer processor (2.3 GHz, 32 integer cores and 16 floating point cores per node).

Without loss of generality, the data for the empirical evaluation of our algorithm come from North Carolina. Our evaluation is at the voter tabulation district level (VTD). GIS census data are available online for all U.S. states while election results are available at the VTD level for some states. These data include the shape, population, election, and party registration information for each district. North Carolina provides an especially interesting redistricting application because its long history of voting right issues have helped shape the state of voting rights legislation in the United States. Our PEAR solver, however, is general and takes three types of inputs: the GIS data, the rook and queen neighborhood matrix, and information needed for compactness evaluation (i.e., area and perimeter information as well as the unit–unit border length information). The latter two types of input can be obtained using open source GIS libraries such as PySAL (http://pysal.org) and GDAL (http://gdal.org).

6.2. Comparison with other redistricting tools

To evaluate the performance of our EA algorithm and the effectiveness of the EA operators, we compare our sequential PEAR algorithm to the BARD redistricting software [5]. There are not many software choices that perform functional automated redistricting for realistic redistricting applications [5]. We choose to present a comparison with BARD v1.2.4 because it is a known R package that provides a set of heuristic algorithms for redistricting. BARD supports methods for generating initial redistricting plans, including random generation [22], random but contiguous equipopulous districts [8], and simple and weighted *k*-means based generation. In addition, it provides several heuristics to refine initial plans, including simulated annealing, greedy search, tabu search, and Greedy Randomized Adaptive Search Procedure (GRASP). BARD formulates objectives and constraints as score functions that can be defined by users. For computational efficiency, it employs efficient data structures and native C libraries for routines which exhibit poor performance in R.

In the BARD implementation, the greedy algorithm is a hill-climbing local search method. The tabu search uses the greedy algorithm to explore the solution space but maintains a tabu list to avoid duplicating search efforts in similar solution space regions. The GRASP algorithm is a multi-start greedy algorithm with multiple randomly generated starting candidate solutions. Lastly, the simulated annealing algorithm conducts a probabilistic search of the solution space and then conducts another hill-climbing search after the annealing process has completed. All of the heuristics create new solutions by examining exchangeable units near district borders through 1- or 2-exchange operations.

When we ran these heuristics, we used a seed solution to begin the search process. Contiguity is, of course, legally required in the output plan. It is worth noting that if the seed solution was not contiguous, BARD exhibited great difficulty in producing contiguous plans during the search. Likely, given the problem size (2690 units), the connectivity graph is too large to allow a fast merge of any two disconnected subsets of a same district. To bypass this issue, we generated only randomized contiguous plans to seed the BARD heuristics. In addition, because BARD's implementation of the area–perimeter compactness is sufficiently inefficient to make the computation prohibitively slow, we did not include compactness in the specification of the objective function. Instead, the fitness function in this experiment is a weighted sum of the competitiveness and equal population measures. We also ran two versions of GRASP. The default GRASP setting uses random sampling, which generated numerous violations of the contiguity requirement and hence was unable to produce comparable results. To fix this issue, we modified the R code for GRASP to constrain it to use random contiguous samples. This run is reported as "GRASP (contiguous)."

Table 2 displays the results from the five BARD runs and one sequential PEAR run. The best fitness value, the computation time, and the number of iterations required to achieve the best fitness are measured. Each heuristic ran until either the search converged or the stopping criteria were met. Both GRASP versions experienced long starving times (over 19 h) without improvement after the reported best fitness values were found. We had to terminate both runs manually. Among the five BARD runs, GRASP (contiguous) produced the best fitness (0.0411), but it took more than 5 h to reach this fitness and was not able to improve upon this fitness even after an additional 19 h of subsequent searching effort. The simulated annealing algorithm produced the solution with the worst fitness (0.1237), though it took the fewest number of iterations. Its iterations were the slowest, taking about a second for each iteration to complete. The greedy and GRASP (default) heuristics had almost identical performance: while their solution quality was the second best, they ran more quickly (48.88 and 47.72 iterations/second compared to 0.99 iterations/second for simulated annealing). The solution found by the tabu search is

Table 2
Performance of sequential heuristic algorithms.

	Solution quality			Cost			Cost per improvement		
Algorithms	Best fitness	Competitiveness	Equal population	Time (in seconds)	Iterations	Iterations per second	Improvements	Iterations	Time
Simulated annealing	0.1237	0.0696	0.3402	1489.12	1472	0.99	130	11.32	11.45
Greedy	0.0980	0.0659	0.2264	3817.91	186,619	48.88	858	217.50	4.45
Tabu	0.0984	0.0658	0.2288	1968.94	92,659	47.06	794	116.70	2.48
GRASP (default)	0.0980	0.0659	0.2264	3910.87	186,619	47.72	858	217.50	4.56
GRASP (contiguous)	0.0411	0.0491	0.0093	19,140.91	320,386	16.74	2075	154.40	9.22
PEAR snapshot 1	0.0403	0.0425	0.0316	1555.64	45,358	29.16	44	1030.86	35.36
PEAR snapshot 2	0.0326	0.0303	0.0419	3511.02	99,300	28.28	65	1527.69	54.02
PEAR snapshot 3	0.0219	0.0199	0.0300	5207.64	150,577	28.91	112	1344.44	46.50
PEAR snapshot 4	0.0180	0.0176	0.0198	6746.70	200,653	29.74	121	1658.29	55.76
PEAR snapshot 5	0.0167	0.0139	0.0279	8963.47	270,732	30.20	135	2005.42	66.40
PEAR final	0.0145	0.0117	0.0256	10,784.55	329,572	30.56	142	2320.93	75.95

slightly worse than the greedy and the GRASP (default) solutions, and took much less time to compute, likely because of the use of a tabu list. It, however, suffered from early convergence.

We ran the sequential PEAR algorithm for three hours. The sequential PEAR run handily outperformed all five BARD runs, obtaining better solution fitness and doing so in much less time. In Table 2, we display snapshot results (every half hour) of PEAR's metrics beginning after 1555.64 s into the run, at which time the sequential PEAR run identified a solution with fitness value 0.0403, which is better than the best result found by any of the BARD runs. As we can see from the snapshots, PEAR continued to steadily improve its solution quality without early termination or a long starving period during which no better solution is identified. The best fitness obtained by the sequential PEAR run is 0.0145, which is significantly better than any of the solutions produced by any of the BARD heuristics.

Table 2 also presents metrics for assessing the numerical efficiency as measured by the cost to gain fitness improvement. We present the overall computation cost as well as the computation cost per improvement. Since the iteration speed for the simulated annealing algorithm was so much slower than the other algorithms, its metrics are accordingly affected. For the BARD heuristics, the tabu search improved the most quickly, finding a better solution every 2.48 s. Neither the greedy nor the default GRASP algorithms was able to identify significant improvements. While they made a number of improvements (858), each improvement was sufficiently small that they did not make a significant contribution to the solution fitness. More pointedly, the total number of search iterations (186,619) eclipses this relatively small number of modest improvements. The GRASP (contiguous) run benefited from the multi-start strategy, but took almost 5 h more than PEAR to obtain its best solution. In comparison, the sequential PEAR algorithm initially appears to be slow in identifying improvements. On average, it took 1030.86 iterations and 35.36 s to make each improvement in the first snapshot. However, these improvements were substantial; the spatial EA operators were able to make significant gains in solution quality with a relatively small number of improvements. This performance can be attributed to two specific aspects of the PEAR algorithm. First, unlike the BARD heuristics that concentrate on improving a single solution, the evolutionary process in the sequential PEAR heuristic attempts to improve an entire population of solutions. The greater population size exacts a greater cost in iteration terms. Second, PEAR's more dramatic improvements clearly indicate that its numerical efficiency advantage has its roots in the utility of the evolutionary algorithm and the effectiveness of its spatial EA operators, not from the sheer quantity of iterations per second. Though it became increasingly difficult to find better solutions under tighter fitness thresholds, as evidenced by the increased cost per improvement shown in the subsequent snapshots, the sequential PEAR nonetheless steadily progressed toward better solutions.

In summary, the EA in PEAR was able to find significantly better solutions overall, was more effective and efficient in its search, and steadily improved even at increasingly tighter fitness levels in these experiments. We attribute the performance differential to our consideration of the redistricting problem as an inherently spatial optimization problem, and thus to our design of EA operators that explicitly, effectively, and efficiently incorporate spatial configurations.

6.3. Scalability analysis

With an effective sequential EA as the base, we now evaluate the capability of our parallel algorithm to scale its performance with the number of processors employed. Strong scaling tests, which increase the number of processors while keeping the global population size intact, are problematic for measuring the performance of a PGA/PEA because the time taken to reach a solution quality threshold depends on both the amount of numerical work done by all of the processors and the stochastic nature of the EA search as demonstrated by [36]. Instead, we evaluate the weak scaling of PEAR. In PEAR weak scaling tests, “data size” amounts to the numerical work done by each process instead of the number of problem variables (VTDs). This definition of weak scaling is termed “new-era” weak scaling [49]. Given that Liu and Wang [36] have demonstrated the advantages of asynchronous migration over synchronous migration, we use only asynchronous migration for our PEA. In our experiment, the population size of each process is set to 200. The size of the global population across all the processes is thus 200 times the number of processes. To avoid precision issues for floating point number comparison on fitness, the fitness value of each solution is multiplied by 10,000. The fitness function is a weighted sum of competitiveness, population deviation, and compactness, as specified in Table 1. Because our EA operators do not inherently consider compactness, including compactness as a weighted component in the fitness function makes fitness improvements highly dependent on compactness, allowing us to observe the gains of utilizing more parallel computing power in our parallel algorithm that is separate from the efficiency afforded by our spatially cognizant EA operators.

Fig. 6 shows the results of the weak scaling test from a two-hour PEAR run using 32,768, 65,536, and 131,072 integer cores on Blue Waters. The time taken to reach multiple solution quality thresholds (normalized fitness values; smaller value indicates better solution) is measured. In the plot, the solution threshold values are indicated besides each line/point. The downward sloping lines illustrate that for each solution quality threshold, utilizing more processors generally reduced the amount of time required to

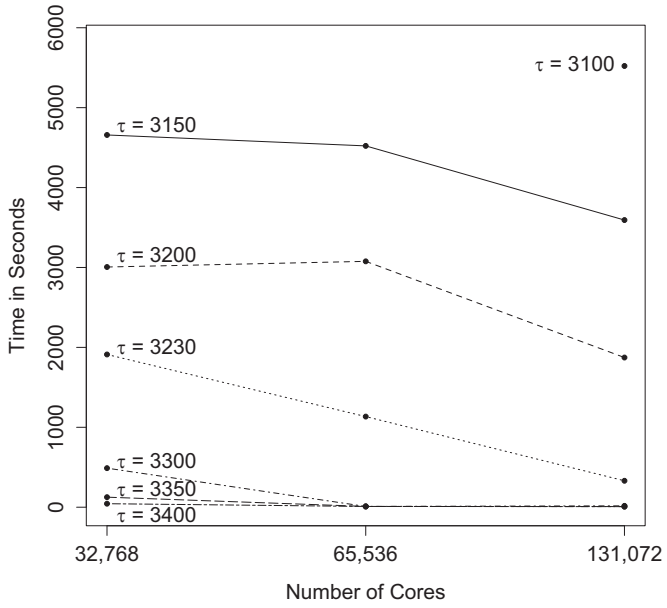

Fig. 6. PEAR weak scaling test results at varying solution quality (fitness) thresholds, τ.

reach each threshold. In addition, as evidenced by the single point in the upper right of the plot, we were only able to reach the tightest threshold of 3100 when we utilized 131,072 processors. Together, these results indicate that our algorithm scales well with desirable outcomes as the number of processors increases.

In total, 1,174,702 unique feasible solutions were generated (167,719 from the run using 32,768 cores; 337,165 from the run using 65,536 cores; and 669,818 from the run using 131,072 cores). The number of feasible solutions increases fairly linearly with the number of processor cores. The best solution found using 32,768 cores has a fitness value of 3133.98. The best solution found using 65,536 and 131,072 cores improved this value to 3,125.88 (8.10 less) and 3083.16 (50.82 less), respectively. It is well known that as the solution fitness approaches the optimal, it becomes much more difficult to find better solutions. Accordingly, the benefit of solution fitness improvement in this study demonstrates the direct benefit of harnessing more computing power to enhance the problem solving capabilities of an EA solver.

6.4. *Large set of feasible solutions for statistical analysis*

Solutions obtained from the scalability analysis are used for redistricting analysis. In cases of partisan gerrymandering, the Court has made it clear that it has the judicial obligation to determine "how much unfairness is too much" (126 S.Ct. at 2638 (n. 9)). To fulfill this responsibility, the Court needs data and measures to assess redistricting maps. Our algorithm generates a large number of good solutions, *two orders of magnitude more* and more efficiently than any other existing algorithms. Having a large number of solutions is essential for understanding redistricting maps because it allows us to place any particular map into context. Others have attempted to simulate maps [8,7], though none of those attempts even remotely approached the quantity or quality of the maps created by PEAR.

For this analysis, the existing 2011 Rucho–Lewis plan (shown on the left in Fig. 7) was used as a seed solution for PEAR. The figure on the right in Fig. 7 shows one of the initial compact plans that is generated with our population initialization strategy. Our algorithm was able to create more than 1 million reasonably good and feasible solutions satisfying the defined competitiveness, population deviation, and compactness principles.

Fig. 8 summarizes the 1,174,702 solutions identified by our algorithm on four particular redistricting criteria for quantitative redistricting research: competitiveness, responsiveness, biasedness, and vote efficiency.[6] Competitiveness measures similarity as a function of relative partisan proportions. Responsiveness and biasedness are elements from the seats-votes curve [9]. Responsiveness is a measure of how sensitive seat gains are to changes in vote proportion. Biasedness is the condition of favoring one party over the other and can be described as a deviation from bipartisan symmetry. The efficiency gap captures a difference in wasted votes between two parties in an election [51]. The histograms show the distribution of values of each of the criteria from the 1.1+ million maps. The green line shows how the 2001 map fared on each criteria while the red line indicates how the current 2011 map compares. The distributions allow us to assess whether the existing plans are outliers among other plans that could have been drawn. We can then understand whether among the possibilities, is this proposed plan particularly unresponsive to voter preferences? Is this proposed plan exceptionally biased toward one party? Could we have achieved the goals of this new plan while maintaining greater respect for other important criteria or traditional districting principles such as respect for political subdivisions or compactness? Is the shift toward a Republican or Democratic bias a function of shifting demographics and population migration or are the motivations of the partisan line drawers the driving force? On the other hand, if the proposed plan is not exceptional in any way but is still biased toward one party, then the Court may decide that grounds do not exist for revisiting the proposed plan. The pivot lies not within the plan itself or model simulations based on one particular plan, but in how that plan compares to other possibilities. Without the ability to generate a range of plans, we would not be able to make reasoned arguments about the impact of redistricting on the electoral process. We would, moreover, not be able to persuasively make the types of legal arguments that are required by Supreme Court mandates. Plainly, the ability to generate and analyze a large number of feasible redistricting plans is essential for ensuring our democratic values.

Statistical approaches with limited data have been tremendously insightful on any number of realms. More recently, with the proliferation of significant computing power, we have discovered the extensive and often surprising reach of technology, information, and computation into many realms of life. These very same capacities can shed insight into our governance structures, ideally enabling us to improve our democratic society. This is our approach here—to integrate technological advances with our articulated strategy for analyzing, contextualizing, and understanding redistricting plans.

7. Discussion

Our computational approach is designed to identify redistricting maps that satisfy a set of user-defined criteria with a particular focus on addressing fine levels of spatial granularity. We leveraged and enhanced a scalable PGA library to develop PEAR for the computationally intensive redistricting problem. By incorporating a set of spatial configuration operators and spatial EA operators to handle spatial characteristics and the associated computational challenges in the EA search process, and harnessing massive

[6] In creating these maps, we additionally considered population equality, compactness, and contiguity. These histograms are available from the authors though not presented here because additional graphs are not necessary to illustrated the value of our approach and take away from the conciseness of the illustration.

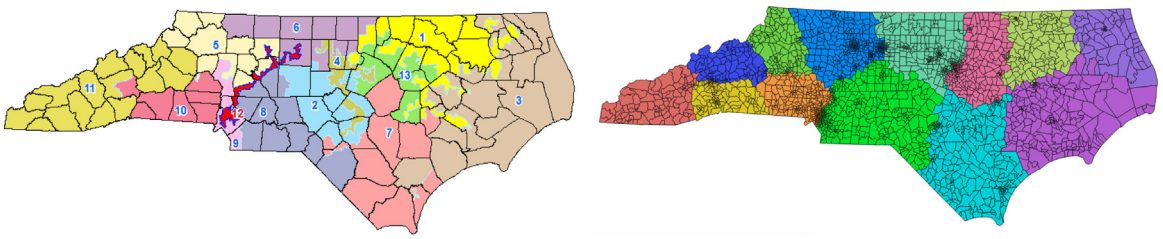

Fig. 7. The 2011 Rucho–Lewis plan and one initial compact plan.

computing power provided on supercomputers, PEAR provides a powerful and computationally scalable redistricting tool.

To be sure, many challenges and additional research directions remain. We plan to further improve the EA algorithm of PEAR by continuing to refine the spatial operators used in the sequential algorithm. We will consider how to integrate compactness within the EA operators. We will also examine other crossover operators that may perhaps be more efficient and effective for identifying new plateaus with good solution quality. In this vein, we have begun to employ a path relinking heuristic [20] where a sequence of gradual moves from one solution to another are vetted to identify better solutions.

While we have made considerable progress, numerical efficiency can be and needs to be further enhanced before our redistricting tool becomes fully functional and practical for general application. We will exert effort into continuing to advance the collective search strategy in a parallel computing environment. In this realm, we will devote attention to improving our specific migration strategies. Here, we have already made strides in designing distributed diversification and intensification search protocols. These protocols will greatly strengthen our solution space traversal, helping us to more efficiently avoid the exploration of

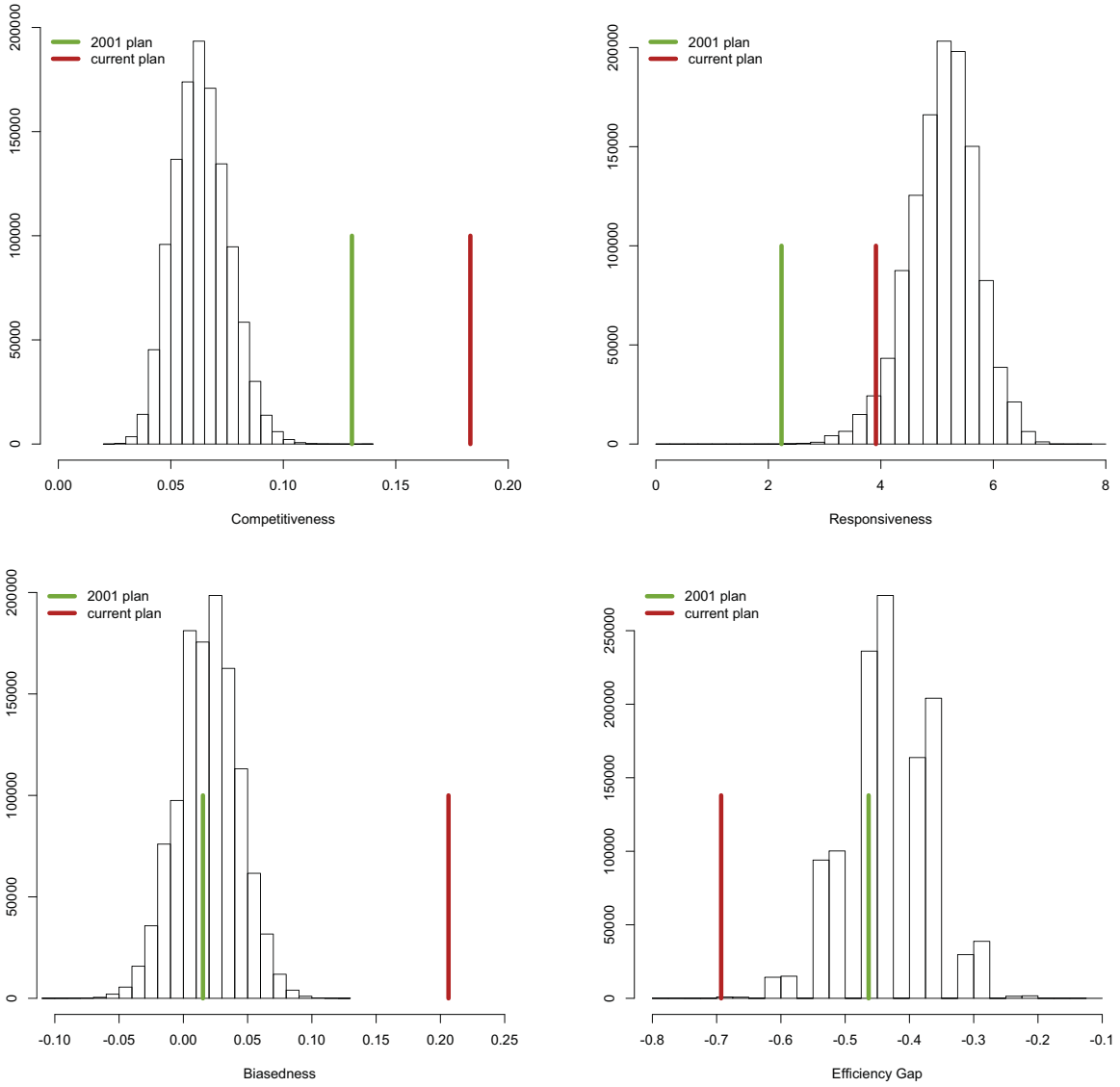

Fig. 8. Current plan versus 1,174,702 reasonably imperfect plans on multiple redistricting criteria. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

similar solution space regions by multiple processes (i.e., diversification) and allow more computing power to be gathered and focused on climbing particularly difficult peaks (i.e., intensification). Generating a large number of maps, which we have now achieved, is important and fundamental for the subsequent statistical analysis. Indeed, while our work has yielded significant gains, the computational complexity of the redistricting problem is formidable, leaving much progress to still obtain. Nonetheless, we have laid a foundation with key advancements upon which one can readily build. We will also extend the current weighted sum method for multi-objective evolutionary algorithm (MOEA) [57,58] to derive the set of pareto front solutions for preference-based decision making support. Our PEA implementation is sufficiently general for solving MOEA benchmark problems [59] and other domain-specific problems such as the unit commitment scheduling problem [60].

In the substantive realm, computational models such as the one developed in this paper, allow us to synthesize and organize massive amounts of computation and data to evaluate redistricting schemes and tailor them to notions of "fairness" and democratic rule. Importantly, our model is deeply rooted in the long-standing values and goals for democracy in the U.S. as they has been outlined by the Supreme Court. Our optimization algorithm is not borne out of convenience and ease, but instead, strongly tailored to the existing social systems and their structures and mechanisms for securing fairness and transparency in the electoral systems.

While our computational approach is formulated via an optimization strategy, our goal is not to seek a perfectly optimized plan per se because the perfectly optimized plan, while interesting, is not necessary, and further, has only limited value. Plainly, it is difficult, and likely impossible to identify the optimal plan given the many competing interests in any redistricting effort. No single plan will satisfy every interested stakeholder—there is no perfect plan. If there is no perfect plan for every constituent, then it makes sense that we will instead be choosing from a bounded set of "reasonably imperfect plans" [6]. The discrete optimization framework is ideal as a vehicle for identifying large sets of "reasonably imperfect" redistricting plans. The set of reasonably imperfect plans is useful at the districting drawing stage as well as for judges who are adjudicating the constitutionality of a redistricting plan. At the drawing stage, this information suitably drives an iterative bargaining process whether that process involves partisan legislators or members of an independent redistricting commission. When this bounded set of plans can be identified and the elements of the set have associated indicators of partisan bias, responsiveness, efficiency gaps, compactness, respect for political subdivisions, etc., the redistricting process is imbued with valuable structure that is otherwise non-existent. That structure alone makes the redistricting process more transparent and may itself serve to reduce legal challenges. When a legal challenge is mounted, this large set of reasonably imperfect plans produces the relevant background and allows one to place and understand the proposed plan in context. The purpose of our optimization strategy, then, is to search, synthesize, and organize massive amounts of information that will supply a common base of knowledge to guide an informed and intelligent debate.

In our analysis of North Carolina, we acknowledge that our statement that the current North Carolina plan is an outlier is a normative statement. We do not intend to and do not imply that we are creating any type of hard standard by which to judge whether a particular plan is an outlier. That type of judgment should be left to the Supreme Court. We simply argue that having and using a computational tool like PEAR is a useful and compelling way to examine whether an instance of partisan gerrymandering has occurred. It provides a much more nuanced and rich analysis than is available with extant tools and notions.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation (NSF) under Grant 1047916. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not reflect the views of NSF. Computational experiments in this research used the Blue Waters sustained-petascale computing resources, which is supported by the NSF (Grants OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. The performance study used the ROGER supercomputer at the University of Illinois, which is supported by NSF (Grant 1429699). The authors also acknowledge the Texas Advanced Computing Center (TACC) at the University of Texas at Austin for providing HPC resources via the Stampede supercomputer that have also contributed to our research results.

References

[1] Enrique Alba, Jose M. Troya, An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and Panmictic Islands, in: Jose Rolim, Frank Mueller, Albert Y. Zomaya, Fikret Ercal, Stephan Olariu, Binoy Ravindran, Jan Gustafsson, Hiroaki Takada, Ron Olsson, Laxmikant V. Kale, Pete Beckman, Matthew Haines, Hossam ElGindy, Denis Caromel, Serge Chaumette, Geoffrey Fox, Yi Pan, Keqin Li, Tao Yang, G. Chiola, G. Conte, L.V. Mancini, Domenique MaCry, Beverly Sanders, Devesh Bhatt, Viktor Prasanna (Eds.), Parallel and Distributed Processing, Lecture Notes in Computer Science, vol. 1586, Springer, Berlin, Heidelberg, London, UK, 1999, pp. 248–256.
[2] Enrique Alba, José M. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (1999) 31–52.
[3] Enrique Alba, Jose M. Troya, Improving flexibility and efficiency by adding parallelism to genetic algorithms, Stat. Comput. 12 (2) (2002) 91–114.
[4] Enrique Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Trans. Evol. Comput. 6 (5) (2002) 443–462.
[5] Micah Altman, Michael P. McDonald, BARD: better automated redistricting, J. Stat. Softw. 42 (4) (2011) 1–28.
[6] Bruce E. Cain, Redistricting commissions: a better political buffer?, Yale Law J. 121 (7) (2012) 1808–1844.
[7] Jowei Chen, Jonathan Rodden, Unintentional gerrymandering: political geography and electoral bias in legislatures, Q. J. Polit. Sci. 8 (2013) 239–269.
[8] Carmen Cirincione, Thomas A. Darling, Timothy G. O'Rourke, Assessing South Carolina's 1990s congressional districting, Polit. Geog. 19 (2) (2000) 189–211.
[9] F.Y. Edgeworth, Miscellaneous applications of the calculus of probabilities, J. R. Stat. Soc. 61 (3) (1898) 534–544.
[10] Ahmad Faraj, Pitch Patarasuk, Xin Yuan, A study of process arrival patterns for MPI collective operations, in: ICS'07: Proceedings of the 21st Annual International Conference on Supercomputing, ACM, New York, NY, 2007, pp. 168–179.
[11] Thomas A. Feo, Mauricio G.C. Resende, Greedy randomized adaptive search procedures, J. Glob. Optim. 6 (1995) 109–133.
[12] M.A. Fleischer, Simulated annealing: past, present, and future, in: Proceedings of the 27th Winter Simulation Conference, 1995, pp. 155–161.
[13] David B. Fogel, Evolutionary algorithms in theory and practice, Complexity 2 (4) (1997) 26–27.
[14] Y.S. Frolov, Measuring shape of geographical phenomena: a history of the issue, Sov. Geog. Rev. Transl. 16 (10) (1975) 676–687.
[15] Philippe Galinier, Jin-Kao Hao, Hybrid evolutionary algorithms for graph coloring, J. Comb. Optim. 3 (4) (1999) 379–397.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., New York, NY, 1979.
[17] R.S. Garfinkel, G.L. Nemhauser, Optimal political districting by implicit enumeration techniques, Manag. Sci. 16 (8) (1970), B-495–B-508.
[18] Burton C. Gearhart, John M. Liittschwager, Legislative districting by computer, Behav. Sci. 14 (5) (1969) 404–417.
[19] Fred Glover, Manuel Laguna, Tabu Search, Kluwer Academic Publishers, Boston, MA, 1997.
[20] Fred Glover, Manuel Laguna, Rafael Marti, Fundamentals of scatter search and path relinking, Control Cybern. 29 (3) (2000) 653–684.
[21] David Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Professional, Reading, MA, 1989.
[22] Bernard Grofman, For single member districts random is not equal, in: Bernard Grofman, Arend Lijphart, Robert B. McKay, Howard A. Scarrow (Eds.), Representation and Redistricting Issues, Lexington Books, Lanham, MD, United States, 1982.
[23] G. Gudgin, P.J. Taylor, Seats, Votes and the Spatial Organization of Elections, Pion, London, 1979.

[24] Curtis Harris, A scientific method of districting, Behav. Sci. 9 (1964) 219–225.
[25] William E. Hart, Scott B. Baden, Richard K. Belew, Scott R. Kohn, Analysis of the numerical effects of parallelism on a parallel genetic algorithm, in: IPPS'96: Proceedings of the 10th International Parallel Processing Symposium, IEEE Computer Society, Washington, DC, 1996, pp. 606–612.
[26] J. Ignacio Hidalgo, Francisco Fernandez, Juan Lanchares, Erick Cantú-Paz, Albert Zomaya, Parallel architectures and bioinspired algorithms, Parallel Comput. 36 (10–11) (2010) 553–554.
[27] John H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1992.
[28] Chun-Hsi Huang, Sanguthevar Rajasekaran, High-performance parallel bio-computing, Parallel Comput. 30 (9–10) (2004) 999–1000.
[29] Hesam Izakian, Witold Pedrycz, A new PSO-optimized geometry of spatial and spatio-temporal scan statistics for disease outbreak detection, Swarm Evol. Comput. 4 (2012) 1–11.
[30] Weihang Jiang, Jiuxing Liu, Hyun-Wook Jin, D.K. Panda, W. Gropp, R. Thakur, High performance MPI-2 one-sided communication over InfiniBand, in: IEEE International Symposium on Cluster Computing and the Grid, 2004, CCGrid 2004, IEEE Computer Society, Los Alamitos, CA, 2004, pp. 531–538.
[31] M. Keane, The size of the region-building problem, Environ. Plan. A 7 (1975) 575–577.
[32] Douglas M. King, Sheldon H. Jacobson, Edward C. Sewell, Wendy K. Tam Cho, Geo-graphs: an efficient model for enforcing contiguity and hole constraints in planar graph partitioning, Oper. Res. 60 (5) (2012) 1213–1228.
[33] Zdenek Konfrst, Parallel genetic algorithms: advances, computing trends, applications and perspectives, in: 18th International Parallel and Distributed Processing Symposium, 2004, pp. 162-. http://dx.doi.org/10.1109/IPDPS.2004.1303155.
[34] Wenwen Li, Michael F. Goodchild, Richard Church, An efficient measure of compactness for two-dimensional shapes and its application in regionalization problems, Int. J. Geogr. Inf. Sci. 27 (6) (2013) 1227–1250.
[35] Shyh-Chang Lin, W.F. Punch III, E.D. Goodman, Coarse-grain parallel genetic algorithms: categorization and new approach, in: Proceedings, Sixth IEEE Symposium on Parallel and Distributed Processing, 1994, pp. 28–37.
[36] Yan Y. Liu, Shaowen Wang, A scalable parallel genetic algorithm for the generalized assignment problem, Parallel Comput. 46 (2015) 98–119.
[37] Yan Y. Liu, Wendy K. Tam Cho, Shaowen Wang, A scalable computational approach to political redistricting optimization, in: Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure, XSEDE'15, vol. 6, ACM, New York, NY, 2015, pp. 1–2.
[38] William Macmillan, Todd Pierce, Optimization modelling in a GIS framework: the problem of political districting, in: Stewart Fotheringham, Peter Rogerson (Eds.), Spatial Analysis and GIS, Taylor & Francis, Abingdon, Oxfordshire, United Kingdom, 1994, pp. 221–246 (Chapter 11).
[39] Michael Mascagni, Ashok Srinivasan, Algorithm 806: SPRNG: a scalable library for pseudorandom number generation, ACM Trans. Math. Softw. 26 (3) (2000) 436–461.
[40] Anuj Mehrotra, Ellis L. Johnson, George L. Nemhauser, An optimization based heuristic for political districting, Manag. Sci. 44 (8) (1998) 1100–1114.
[41] Efrén Mezura-Montes, Carlos A. Coello Coello, Constraint-handling in nature-inspired numerical optimization: past, present and future, Swarm Evol. Comput. 1 (4) (2011) 173–194.
[42] Daniele Muraro, Rui Dilão, A parallel multi-objective optimization algorithm for the calibration of mathematical models, Swarm Evol. Comput. 8 (2013) 13–25.
[43] Stuart S. Nagel, Simplified bipartisan computer redistricting, Stanf. Law Rev. 17 (5) (1965) 863–899.
[44] Jiri Ocenasek, Martin Pelikan, Parallel mixed Bayesian optimization algorithm: a scaleup analysis, in: S. Cagnoni (Ed.), Workshop Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004, Electronic publication, 2004.
[45] Robert Osserman, Isoperimetric inequality, Bull. Am. Math. Soc. 84 (6) (1978) 1182–1238.
[46] L. Papayanopoulos, Quantitative principles underlying apportionment methods, Ann. N. Y. Acad. Sci. 219 (1973) 3–4.
[47] C. Patvardhan, Sulabh Bansal, A. Srivastav, Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems, Swarm Evol. Comput.
[48] D.J. Rossiter, R.J. Johnston, Program GROUP: the identification of all possible solutions to a constituency-delimitation problem, Environ. Plan. A 13 (2) (1981) 231–238.
[49] Vivek Sarkar, William Harrod, Allan E. Snavely, Software challenges in extreme scale systems, J. Phys.: Conf. Ser. 180 (1) (2009) 012045.
[50] J.W. Shepherd, M.A. Jenkins, Decentralizing high school administration in detroit: a computer evaluation of alternative strategies of political control, Econ. Geogr. 48 (1) (1970) 95–106.
[51] Nicholas O. Stephanopoulos, Eric M. McGhee, Partisan gerrymandering and the efficiency gap, Univ. Chic. Law Rev. 82 (2) (2015) 831–900.
[52] James Thoreson, John Liittschwager, Computers in behavioral science: legislative districting by computer simulation, Behav. Sci. 12 (1967) 237–247.
[53] William S. Vickrey, On the prevention of gerrymandering, Polit. Sci. Q. 76 (1961) 105–110.
[54] J.B. Weaver, S.W. Hess, A procedure for nonpartisan districting: development of computer techniques, Yale Law J. 72 (1963) 288–308.
[55] Ningchuan Xiao, A unified conceptual framework for geographical optimization using evolutionary algorithms, Ann. Assoc. Am. Geogr. 98 (4) (2008) 795–817.
[56] Ningchuan Xiao, David A. Bennett, Marc P. Armstrong, Using evolutionary algorithms to generate alternatives for multiobjective site-search problems, Environ. Plan. A 34 (4) (2002) 639–656 (⟨URL ⟩).
[57] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P.N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: a survey of the state of the art, Swarm Evol. Comput. 1 (1) (2011) 32–49.
[58] Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evol. Comput. J. 8 (2) (2000) 173–195 (URL).
[59] S. Biswas, S. Das, P.N. Suganthan, C.A. Coello Coello. Evolutionary multi-objective optimization in dynamic environments: A set of novel benchmark functions. In 2014 IEEE Congr. on Evol. Comput. (CEC). 3192–3199 http://dx.doi.org/10.1109/CEC.2014.6900487.
[60] A. Trivedi, D. Srinivasan, S. Biswas, T. Reindl. Hybridizing genetic algorithm with differential evolution for solving the unit commitment scheduling problem. Swarm and Evol. Comput. 23 (2015) 50–64 http://dx.doi.org/10.1016/j.swevo.2015.04.001.