Fast Sequential Computation of Convex Regions

Anthony E. Pizzimenti
George Mason University
apizzime@gmu.edu

draft: last edited March 24, 2023.

Abstract

Outlier analysis of districting plans — which consists, in part, of generating extremely large sets of valid plans called *ensembles* — is a computationally intense task. These ensembles are often constructed using *Monte Carlo Markov chain* (or *MCMC*) techniques, which allow users to tune the likelihood of a plan's selection to properties of that plan, or of the districts it defines, via a scoring mechanism. *Compactness measures* are a type of numerical score which attempt to detect gerrymandering based on the spatial properties of a given district; some of these scores rely on the computation of a district's convex hull. In practice, however, districts are composed of thousands of individual polygons, which are themselves defined by hundreds or thousands of individual points: even using optimal general-purpose algorithms, determining the convex bounding regions of millions of points takes time. This inefficiency is compounded by repeated computation in an MCMC process, which may constitute hundreds of thousands or millions of steps. Here, we propose methods for thinning point sets, and algorithms which take advantage of MCMC's structure, to efficiently compute convex bounding regions at real-world scale.

Contents

1 Introduction

Rather than evaluating individual districting plans using standalone (and often fundamentally flawed [2, 3]) numerical measures of nefarious intent, the last half-decade has seen a dramatic shift toward context-based mathematical redistricting investigation. Broadly termed *outlier analysis*, this family of analytical tools combines computers and algorithms, mathematical and statistical theory, and political and demographic data to draw conclusions about individual districting plans by comparing them to (subsets of) the collection of all districting plans. Generally, these methods of analysis rely on a computer to generate large samples of legally valid plans called *ensembles*, then on researchers and experts to perform careful, nuanced comparisons of the algorithmically-drawn plans to real-world ones.

Recently, *Monte Carlo Markov chain* or *MCMC* techniques have emerged as the industry standard for ensemble generation. These tools combine *Markov chains* — random processes that serially sample from random variables — with *Monte Carlo* algorithms that allow us to simulate these chains. In redistricting contexts, we simulate Markov chains on the collection of all legally valid districting plans (called the *state space*) by treating each plan as a possible state in the chain, drawing a new plan based on the current one using a randomized procedure, and accepting that plan as the next state in the chain based on certain customizable criteria.

Because these simulated chains — which typically comprise hundreds of thousands or millions of plans — take time to compute, it is essential that the procedures used to evaluate each plan are as efficient as possible: the time it takes any individual procedure to judge a single plan is scaled linearly by the number of plans the simulated chain finds. Evaluation procedures that try to describe *compactness* are often computational resource sinks because they operate primarily on detailed spatial data. A typical evaluation procedure computes the *Reock score*, a measure of spatial compactness that idealizes districts as circles, comparing the area of a given district to the area of the smallest circle containing it. The underlying task, finding the *minimum bounding circle* or *smallest enclosing circle* of spatial data, is a well-studied computational geometry problem: given a set of points, find the circle with smallest possible radius that contains all the points.

Linear-time prune-and-search [5] and expected linear-time randomized [11] algorithms exist for finding minimum bounding circles, but using them in redistricting MCMC contexts presents two further challenges. First, finely-detailed spatial data used for redistricting are composed of millions of individual points, which must be stored, accessed, and operated on efficiently: for example, Iowa's 175,199 Census blocks are defined by 3,804,140 unique points — 8,079,844 when counting duplicates. Such large pointsets, especially ones with multiplicity, can worsen already-poor linear and possibly quadratic algorithmic running times. Second, we must calculate these minimum bounding circles for each district in each plan: if we simulate a chain with 500,000 steps (that is, up to 500,000 unique districting plans) and each plan has 20 districts, we compute ten million minimum bounding circles using a naïve implementation.

In this paper, we describe adjustments to both the algorithms and their inputs to massively improve performance in MCMC-for-redistricting contexts.

2 Definitions and examples

2.1 Foundations

To begin, we provide general, topologically-motivated definitions for two concepts underlying all redistricting work: *states* and *units*.

Definition (State, unit)**.** Given a topological space X (typically, $X = \mathbb{R}^2$ or $X = S_2$ under their respective standard topologies), a *state* $\mathcal{S}$ is a path-connected subset of X.[a] A *unit* u_i is a path-connected subset of $\mathcal{S}$; the collection of units $\mathcal{U} = \{u_1, \ldots, u_n\}$ tiles $\mathcal{S}$. Generally, states and units are polygons, and units are *atomic* — that is, units are un-splittable. Any polygon P is induced by a sequence of points, or *vertices*, which we denote V_P.

[a] For states with islands, like Massachusetts, we generally consider disjoint connected components to be "connected" to their nearest Euclidean neighbor.

Definition (Boundary unit)**.** Given a state $\mathcal{S}$, a *boundary unit* is a unit u such that $|\partial u \cap \partial \mathcal{S}| \neq \varnothing$. In other words, a boundary unit is a unit whose boundary intersects the state boundary on at least one point.

We then combine units together to form *districts*, and collections of districts make up *districting plans*:

> **Definition** (Districting plan, district)**.** A *districting plan* is an equivalence relation φ on $\mathcal{U}$; a *district* is an equivalence class of φ. Equivalently, a districting plan is a partition φ of $\mathcal{U}$, and a district is a component of φ.

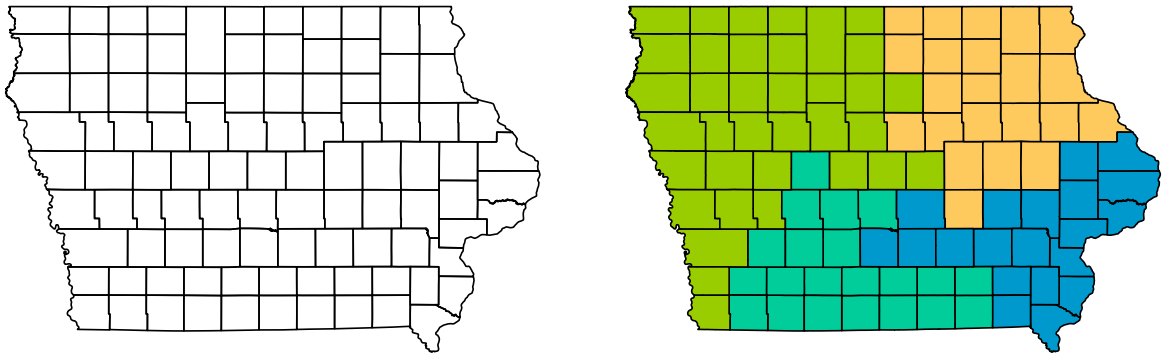

Figure 1: Iowa's 99 counties (left), and Iowa's 99 counties partitioned to form a districting plan (right).

In most jurisdictions, districts are required to be *contiguous*, which translates to path-connectedness under the subspace topology on $\mathcal{S}$ — in other words, a person must be able to choose any two places within a district and walk between them without leaving the district (outside naturally non-contiguous territory, like islands or river crossings). Iowa's[1] counties, shown in Figure 1, form the set of atomic units $\mathcal{U}$ for redistricting in the state: each county is assigned to one of four districts, and each district is contiguous. Given districts and the plans they compose, we can use numerical scores to translate spatial, demographic, and election data into summary statistics. With these statistics in hand, we are then able to compare individual district(ing plan)s head-to-head or against the backdrop of an ensemble.

2.2 Compactness measures

Compactness measures are a family of scores that attempt to quantify the spatial "weirdness" of a district. *Idealized compactness measures* are compactness measures that, given a district D, compare the properties of D to an idealized counterpart I. For example, given a district D, Polsby-Popper [8] defines I as the circle with perimeter equal to the perimeter of D, and defines compactness as the ratio of the perimeter of D to the perimeter of I. Schwartzberg [10] defines I as the circle with the same *area* as D, and defines compactness as the ratio of the area of D to the area of I. Because these scores rely only on the area or perimeter of each polygon, they are trivial to compute.

Two idealized compactness measures, the convex hull score and the Reock score [9], actually require the construction of an idealized district I: given a district D, each score constructs an I using a subset of D's (polygonal) vertices, and returns the ratio of the area of D to the area of I.

[1] The author is from Iowa, and greatly appreciates their home state's ubiquity as a redistricting example.

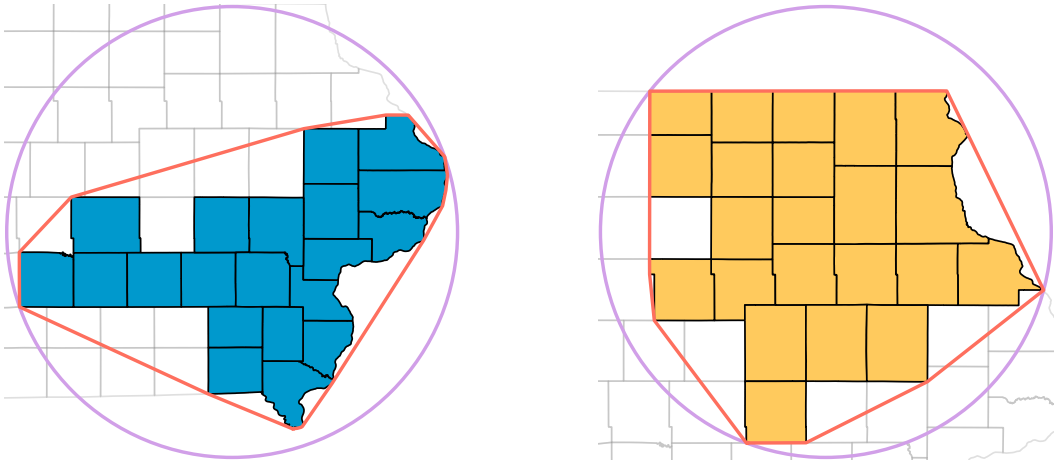

Figure 2: The convex hull (red) and minimum bounding circle (purple) of Iowa's first (left) and second (right) Congressional districts.

Definition (Convex hull, Convex hull score)**.** Given a simple polygon P, the *convex hull* $\mathrm{CH}(P)$ of P is, equivalently:

(a) the intersection of all convex polygons which contain P;
(b) smallest-area convex polygon which contains all points in P;
(c) the polygon induced by the longest subsequence of P's vertices V_P such that, for all pairs of adjacent vertices (v_i, v_j), all vertices of P lie to the right of the directed line $\overrightarrow{v_i v_j}$;
(d) the intersection of all half-planes formed by the $\overrightarrow{v_i v_j}$ as defined in (c).

The *convex hull score* of P is then

$$\sigma_{\mathrm{CH}}(P) = \frac{\mathrm{Area}(P)}{\mathrm{Area}(\mathrm{CH}(P))}$$

Definition (Minimum bounding circle, Reock score)**.** Given a simple polygon P, the *minimum bounding circle* $\mathrm{MinCircle}(P)$ is the minimum-radius circle which entirely encloses P. Equivalently, the minimum bounding circle of a pointset V is the minimum-radius circle which contains all points in V. The *Reock score* is then

$$\sigma_{\mathrm{Reock}}(P) = \frac{\mathrm{Area}(P)}{\mathrm{Area}(\mathrm{MinCircle}(P))}$$

The convex hull and Reock scores idealize each district as the smallest possible convex region and the smallest possible circle containing it, respectively. These measures of compactness are more detailed, as they require data underlying each polygon to give a compactness measurement; Polsby-Popper and Schwartzberg, on the other hand, require only information about perimeter and area. To demonstrate, consider the polygons P_L and P_R in Figure 3, each with area 3 and perimeter 8.

The Polsby-Popper (resp. Schwartzberg) scores for P_L and P_R are the same, as the polygons have equal area

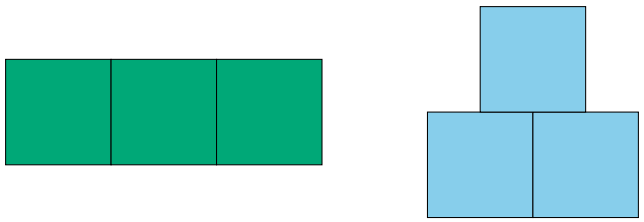

Figure 3: Two different polygons, P_L and P_R, with equal perimeter and equal area.

and equal perimeter:

$$\begin{aligned}\sigma_{\mathrm{PP}}(P_L) &= \frac{4\pi \cdot \mathrm{Area}(P_L)}{\mathrm{Perim}(P_L)^2} \\ &= \frac{4\pi \cdot \mathrm{Area}(P_R)}{\mathrm{Perim}(P_R)^2} \\ &= \sigma_{\mathrm{PP}}(P_R)\end{aligned} \qquad \begin{aligned}\sigma_{\mathrm{Sch}}(P_L) &= \frac{1}{\frac{\mathrm{Perim}(P_L)}{2\pi \cdot \sqrt{\mathrm{Area}(P_L)/\pi}}} \\ &= \frac{1}{\frac{\mathrm{Perim}(P_R)}{2\pi \cdot \sqrt{\mathrm{Area}(P_R)/\pi}}} \\ &= \sigma_{\mathrm{Sch}}(P_R)\end{aligned}$$

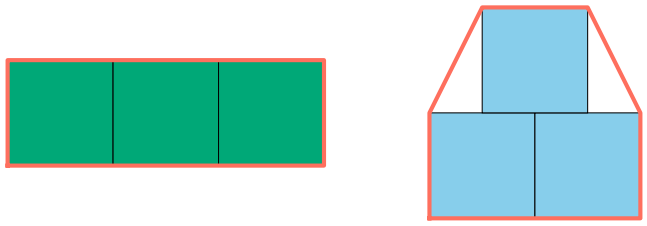

Figure 4: The polygons P_L and P_R have different convex hulls despite having the same area and perimeter. It is also clear that P_L is its own convex hull, but P_R is not — as a result, these polygons *must* have different convex hull scores.

However, by superimposing $\mathrm{CH}(P_L)$ and $\mathrm{CH}(P_R)$ on their respective polygons, as in Figure 4, we can clearly see that $\sigma_{\mathrm{CH}}(P_L) \neq \sigma_{\mathrm{CH}}(P_R)$: P_L is itself convex, while P_R is not. Inherent to their definitions, the convex hull and Reock scores require far more detailed input data than Polsby-Popper or Schwartzberg do, and are thus more costly to compute. Furthermore, in practical settings, the number of districts is often much greater than four and there are far more than 99 base units to play with; the computational cost again increases.

2.3 Computational modeling

By itself, computing the convex hull or Reock score for a given district or districing plan is relatively cheap: even when the number of points defining each district's polygon is quite large, efficient algorithms on modern hardware can easily and quickly perform the required operations. Contemporary redistricting analyses, however, are often based on *outlier analysis*: the construction of extremely large samples of legally valid districting plans called *ensembles*, and the subsequent comparison of individual plans to those ensembles. The industry-standard

method of ensemble generation is via *Monte Carlo Markov chain* or *MCMC* techniques: many variations on these MCMC processes exist, but they are all based on the same underlying principles.

Definition (Random process, Markov chain, Markov property)**.** Given a random variable R, a *random process* is a series of repeated draws from R indexed by some set $\mathcal{I}$, denoted as $\{R(i)\}_{i\in\mathcal{I}}$. Generally, we set $\mathcal{I} = \mathbb{N}$. A *Markov chain* is a random process $\mathcal{M}$ where the probability that $R(i+1) = r_{i+1}$ is dependent only on the probability that $R(i) = r_i$ — this is called the *memoryless* or *Markov* property. We write the Markov property of random processes as

$$\begin{aligned}&\mathbf{P}\left(R(i+1) = r_{i+1} \mid R(i) = r_i,\ R(i-1) = r_{i-1},\ \ldots,\ R(0) = r_0\right)\\ =\ &\mathbf{P}\left(R(i+1) = r_{i+1} \mid R(i) = r_i\right)\end{aligned}$$

Because the number of base units in $\mathcal{U}$ is finite, there must be a finite number of equivalence classes $\varphi \subseteq \mathcal{U} \times \mathcal{U}$. We choose our (finite) sample space to be $\mathcal{P}$, the set of all valid districting plans (equivalently, a subset of all equivalence classes φ on $\mathcal{U}$). The size of $\mathcal{P}$, however, grows astronomically with the number and arrangement of units: while there are only 117 ways to split a four-by-four grid into four equally-sized pieces, there are more than 706 quadrillion ways to split a nine-by-nine grid into nine equally-sized pieces [7]. This rate of growth, often called *combinatorial explosion*, makes enumerating the plans in $\mathcal{P}$ impossible: the universe would cease to exist before our program terminates. Instead, we generate a representative sample of districting plans called an *ensemble*.

Definition (Proposal function, Monte Carlo Markov chain, ensemble)**.** To simulate a Markov chain $\mathcal{M}$ which samples from $\mathcal{P}$, we define a *proposal function* $\Phi : \mathcal{P} \times [0,1] \to \mathcal{P}$ which consumes a valid districting plan $P_i \in \mathcal{P}$ and a real number α in $[0,1]$, then generates (proposes) a new districting plan $P_{i+1} \in \mathcal{P}$. Proposal functions Φ belong to a class of randomized algorithms called *Monte Carlo* algorithms.

We then simulate the Markov chain $\mathcal{M}$ by repeatedly applying Φ to our initial and subsequent states:

$$\mathcal{M} = \left(P_0, P_1 = \Phi(P_0, \alpha_0), P_2 = \Phi(P_1, \alpha_1), \ldots, P_N = \Phi(P_{N-1}, \alpha_{N-1})\right),$$

where N is a pre-determined number of iterations in the simulation. The strategy of simulating a Markov chain via Monte Carlo proposals is called *Monte Carlo Markov chain*. The resulting sequence of plans $\mathcal{M}$ is an *ensemble*.

In this work, our simulated Markov chains are ReCom chains [4], where a plan generation function uniformly randomly chooses a pair of adjacent districts, computes the union of the units those districts comprise, and re-splits the set of units into two new districts; the resulting plan is proposed as the next iteration of the chain. An example ReCom step on Iowa's counties is shown in Figure 5.

Because raw geometric data is unwieldy, we use a dual interpretation of this redistricting problem, as in [4]: we first create a *dual graph* of the units in $\mathcal{U}$, then conduct a random walk on *equipartitions* of the dual graph, shown in Figure 6.

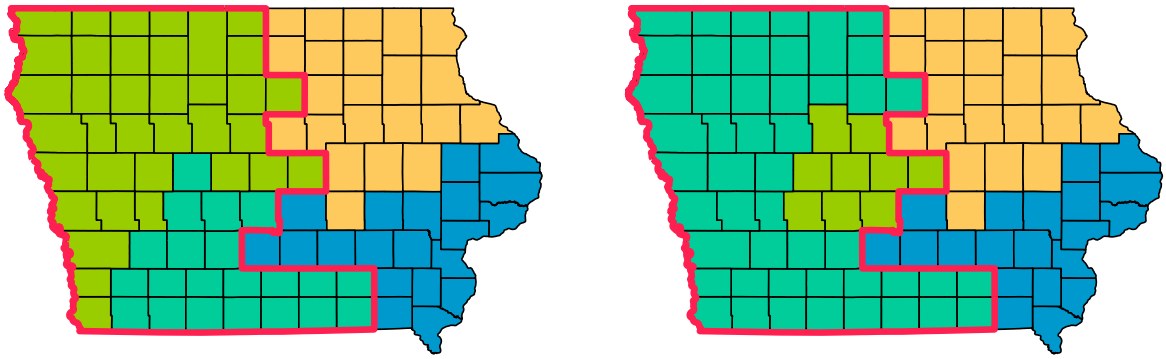

Figure 5: Iowa's Congressional redistricting plan (left) and a new, population-balanced Congressional districting plan proposed by ReCom (right). The two adjacent districts selected to be merged and re-split by ReCom have a colored border.

Definition (Dual graph, equipartition)**.** Given a set of units $\mathcal{U}$, the dual graph $G_{\mathcal{U}} = (V_G, E_G)$ is constructed in the following way: each vertex v_i in V_G corresponds to a unit u_i in $\mathcal{U}$, and an edge (v_j, v_k) is drawn if $j \neq k$ and $|\partial u_j \cap \partial u_k| > 1$ (i.e. u_j and u_k are adjacent: the intersection of u_j's and u_k's boundaries contains more than one point). Because the units in $\mathcal{U}$ are path-connected and tile $\mathcal{S}$, $G_{\mathcal{U}}$ is planar. Each vertex v_i in V_G is assigned a nonnegative weight, usually set to the total population of its corresponding unit u_i. An *equipartition* is an equivalence relation φ on V_G such that the subgraphs induced by equivalence classes of φ are connected and have equal weights (up to some weight tolerance $0 < \varepsilon < 1$).

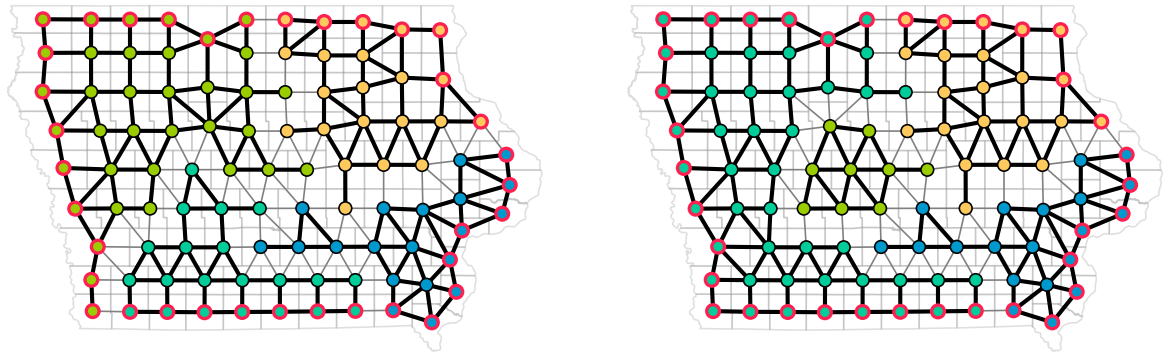

Figure 6: Equipartitions of the dual graph of Iowa's counties, which correspond to the districting plans shown in Figure 5. *Cut edges*, edges whose incident vertices are in different equivalence classes (i.e. different districts), are lighter-weight; boundary vertices have colored borders.

Treating redistricting as a graph partitioning problem lets us take advantage of efficient graph algorithms rather than deal with cumbersome spatial data. In this setting, we are able to associate election, spatial, demographic, and other data from each unit to its respective vertex in the dual graph; this association lets us operate on these data at each iteration of the simulated chain. For our experiments, we assume that the points defining each unit u_i are associated to their respect — that is, each vertex "knows" spatial information about its geometric counterpart. We can additionally classify the vertices of our dual graph $G_{\mathcal{U}}$ based on this information:

Definition (Boundary vertex, internal vertex, cut edge, cut vertex)**.** A *boundary vertex* is a vertex v_i whose corresponding unit u_i is a boundary unit. An *internal vertex* is a vertex which is not a boundary vertex. Equivalently, because $G_{\mathcal{U}}$ is planar, a boundary vertex is a vertex adjacent to the external face of $G_{\mathcal{U}}$. A *cut edge* is an edge (u, v) where u and v belong to different districts; both u and v are the *cut vertices*.

Next, we propose algorithms which take advantage of the self-similarity of sequential plans proposed by ReCom, as well as a minimal set of input data, to efficiently compute convex hulls and the scores dependent on them.

3 Algorithms

3.1 Theoretical justification

Core to efficient minimum bounding circle computation is the size reduction — in fact, minimization — of the input data set. Because commonly-used minimum circle-finding algorithms take linear (with large constants) or expected-linear (but worst-case quadratic) time, it is important to drastically reduce the number of points the algorithms are asked to handle. To do so, we state and prove three lemmas which justify important optimizations.

Lemma 3.1 (Equality of minimum bounding circles)**.** Given a polygon P,

$$\mathrm{MinCircle}(P) = \mathrm{MinCircle}(\mathrm{CH}(P))\,.$$

Proof. Given a polygon P, its minimum bounding disk B (which is bounded by the minimum bounding circle S) is a convex region containing P. Because $\mathrm{CH}(P)$ is the intersection of all convex regions containing P, one of those convex regions must be B, so B must contain $\mathrm{CH}(P)$. As such, $\mathrm{MinCircle}(P) = \mathrm{MinCircle}(\mathrm{CH}(P))$. ■

Lemma 3.1 asserts that the minimum bounding circles of a polygon and its convex hull are the same. In practice, given a district D made up by a set of units $\mathcal{U}_D \subseteq \mathcal{U}$, we can first compute the convex hull of the points inducing the units in $\mathcal{U}_D$, then compute the minimum bounding circle of the resulting hull. In addition to reducing the amount of geometric information required to compute minimum bounding circles by first finding the hull of each district, we can reduce the number of units required to find each hull via another lemma:

Lemma 3.2 (Donut)**.** Let D be the union of units $\mathcal{U}_D = \{u_1, \ldots, u_k\}$. Next, let E be the set of units $u_i \in \mathcal{U}_D$ which contain a point on the boundary of D; these are *exterior* units. Let I be the set of remaining units. Then,

$$\mathrm{CH}\Big(\bigcup E\Big) = \mathrm{CH}(D)\,.$$

Lemma 3.2 is called the Donut lemma because the sets E and I are disjoint, and make a "donut" and "donut hole" out of the units in $\mathcal{U}_D$, respectively.

Proof. First, note that $\mathcal{U}_D$ is the disjoint union of E and I, and D is the union of $\bigcup E$ and $\bigcup I$. Next, let V^* be the set of points which define $\mathrm{CH}(D)$. As each point in V^* must be on the boundary of D, we know that they must also be on the boundary of some unit in E; further, we know that no point on the boundary of D can be on the boundary of any unit $u_i \in I$, otherwise u_i would be in E. Thus, we have

$$\begin{aligned}\mathrm{CH}\Big(\bigcup E\Big) &= \mathrm{CH}\Big(\bigcup E \cup \bigcup I\Big) \\ &= \mathrm{CH}(D)\,.\end{aligned}$$

■

Lemma 3.2 provides us with an additional optimization by ignoring units whose points are irrelevant to the computation of minimum bounding circles. In practice, this is readily attainable in ReCom chains: the dual graph G marks vertices on the boundary of the region and similarly tracks vertices incident to cut edges (as in Figure 6) [6]. Determining the exact set of vertices required — and, by duality, the exact set of units E — is at worst linear in the total number of vertices of G. Additionally, by using the ReCom proposal scheme, we must only re-identify the cut edges for the two districts which were modified. Finally, we posit an important lemma, called the approximation lemma:

Lemma 3.3 (Approximation)**.** Let U be the union of units $\{u_1, \ldots, u_k\}$, and U^* the union of $\{u_1^*, \ldots, u_k^*\}$, where $u_i^* = \mathrm{CH}(u_i)$. Then,

$$\mathrm{CH}(U) = \mathrm{CH}(U^*)$$

Corollary 3.3.1. Let U be the union of units $\{u_1, \ldots, u_k\}$, and U^* the union of $\{u_1^*, \ldots, u_k^*\}$, where $u_i^* = \mathrm{CH}(u_i)$. Then, any point on $\mathrm{CH}(U)$ induces one of the u_i^*.

Proof. Let V be the sequence of vertices (points in the plane) which define $\mathrm{CH}(U)$, and V^* the sequence of vertices which define $\mathrm{CH}(U^*)$. We proceed iteratively by constructing V and V^* in the fashion of the monotone chain algorithm [1]. In brief, we show that the upper (and, by symmetry, lower) hulls of U and U^* are the same: we first establish a point that must be on the upper hull of both U and U^*, then show that each following point on the upper hull of U must also be in U^*, and so must also induce the upper hull of U^*. We conclude by "reversing" the iterative process to show equality of the lower hulls.

Let u_1 be the unit induced by the vertex with the smallest x-coordinate of all vertices in U; call this vertex v_1. Because v_1 has the smallest x-coordinate of all vertices, it has the smallest x-coordinate of all vertices of u_1, and so must be on u_1^*. Now, because v_1 is the leftmost vertex and induces both u_1 and u_1^* — and is thus contained in U and U^* — it is the first vertex in both V and V^*. In other words, v_1 is a vertex of the convex hulls of U and U^*.

Choose v_2 to be the next vertex on V, and let u_2 be the unit induced by v_2. (Note that it is possible for $u_1 = u_2$.) Now, v_2 is the vertex with the smallest x-coordinate such that all vertices in U lie in the half-plane bounded by the directed line $\overrightarrow{v_1 v_2}$. Because all vertices in U lie in this half-plane, all

vertices of u_2 lie in this half-plane. Thus, all vertices of u_2 lie to the right of the half-plane bounded by the directed line $\overrightarrow{v^* v_2}$, where v^* is the vertex on u_2^* which immediately precedes v_2, so v_2 is on u_2^*. Because v_2 is on both V and u_2^* and no preceding point on u_1 or u_2 (which necessarily includes points on u_1^* and u_2^*) forms a bounding half-plane with v_1, v_2 must be on U^*.

We continue in this manner until we reach the rightmost vertex (that is, the point with the largest x-coordinate), which is guaranteed to be on both U and U^*, thus verifying that the upper hulls of U and U^* are the same. To show that the lower hulls are the same, we execute a similar iterative process: we start at the rightmost vertex and choose the vertex with the *largest* x-coordinate such that all other vertices lie in each respective half-plane at each step. We continue until we reach the leftmost vertex, at which point we have constructed both the upper and lower hulls.

Because each sequence contains the same vertices in the same order, we have $V = V^*$, so $\mathrm{CH}(U) = \mathrm{CH}(U^*)$. ■

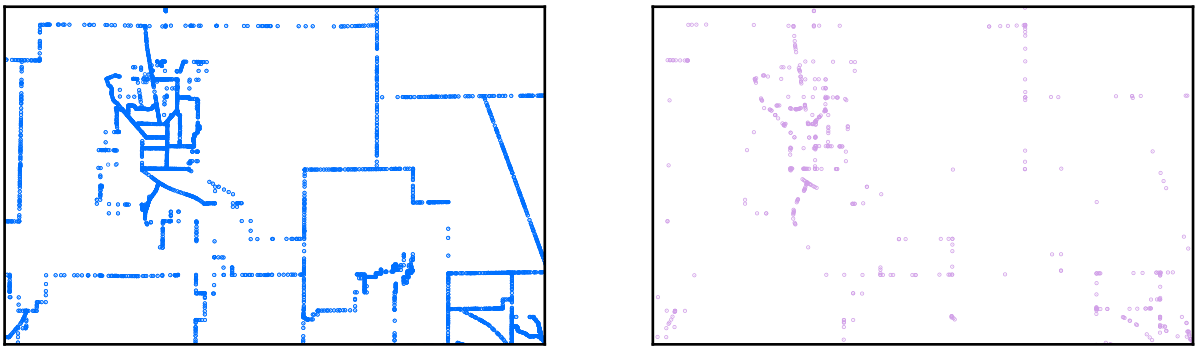

Figure 7: The points inducing voter tabulation districts (VTDs), a typical set of base units, in a suburb of Indianapolis. Left shows the points which define the units themselves; right shows the points which define the convex hulls of the same. Polygonal approximation, for the purpose of computing minimum bounding circles, is justified by Lemma 3.3.

Broadly, Lemma 3.3 says that, regardless of whether we use exact or approximate polygonal data, the resulting convex hulls are the same. With these optimizations in hand, we can describe and compare a range of approaches for repeatedly computing convex hull and Reock scores. In practice, the application of this lemma greatly reduces computational cost: Figure 7 demonstrates the stark difference between the number of points defining polygons and the number of points defining those polygons' convex hulls.

Finally, these results lead to an important corollary, which justifies thinning our geometric dataset before any computation takes place.

Corollary 3.3.2 (Minimality of $\mathscr{P}$)**.** The set

$$\mathscr{P} = \bigcup_{u \in \mathcal{U}} \mathrm{CH}(u_i)$$

is the minimal (with respect to inclusion) set of points sufficient to compute the convex hull of any union of units — in particular, any district — exactly.

Proof. Let $\mathscr{P}^-$ be a strict subset of $\mathscr{P}$ such that $\mathscr{P} - \mathscr{P}^- = \{p\}$. Let $\mathcal{U} = \{u_1, \ldots, u_n\}$, with $D = \bigcup \mathcal{U}$. Suppose that p is a point inducing D, so p belongs to one of the u_i.

Without loss of generality, suppose that u_i is an external unit on D, and suppose that p induces the convex hull of D. Then, by Lemma 3.3, p must induce $u_i^* = \mathrm{CH}(u_i)$. As $\mathscr{P}^-$ does not contain p, we have

$$\mathrm{CH}\big(\mathscr{P}^-\big) \neq \mathrm{CH}(\mathscr{P}) = \mathrm{CH}(D)\,,$$

so $\mathscr{P}^-$ does not admit the same convex hull as $\mathscr{P}$. As such, no proper subset of $\mathscr{P}$ is guaranteed to admit the convex hull of D, so $\mathscr{P}$ is the minimal subset which does so. ■

3.2 Programmatic descriptions and correctness

In programmatic settings, each plan P is treated as a function which maps district labels $i \in \{1, \ldots, k\}$ to sets of units $\mathcal{U}_i$, such that $P(i) = \mathcal{U}_i$: for example, the units in district 1 can be obtained by $P(1) = \mathcal{U}_1$. Next, we have a routine **Points** which takes collections of units $\mathcal{U}_i$ as input, and outputs the union of those units' point sets. Finally, the routine **MinCircle** implements a minimum enclosing circle-finding algorithm such that, given a set of points P, the return values of **MinCircle** are the radius and center of $\mathrm{MinCircle}(P)$.

Algorithm 1 (**Naïve**) a naïve algorithm for computing minimum enclosing circles for each district in a plan P_t. At iteration $t = 1$, we must compute the minimum bounding circles for all districts, as they have not been "seen" by the algorithm before. At any iteration $t > 1$, all but two districts i and j in the plan P_t remain the same, so we must only re-compute scores for districts i and j.

if $t = 1$ **then**
  $M \leftarrow \{1, \ldots, k\}$ ▷ Initially, we compute circles for all districts.
else
  $M \leftarrow \{i, j\}$ ▷ Only districts i and j were modified by ReCom.
end if

for each district index d in M **do**
  $S_d = \textbf{Points}(P_t(d))$ ▷ Points defining units in district d.
  $C_d = \textbf{MinCircle}(S_d)$ ▷ Compute the minimum bounding circle of d.
end for

return $(C_1, \ldots, C_k)$ ▷ Sequence of minimum enclosing circles.

Correctness. Algorithm 1 is trivially correct: the minimum bounding circle of each district is determined by first finding all units $\mathcal{U}_d$ composing district d, then computing the minimum bounding circle C_d of the points defining the $\mathcal{U}_d$, for all d in M. ■

Naïve is an easily implemented but demonstrably slow algorithm. It suffers under extremely large input sizes: in Iowa, for example, a single district built out of counties is defined by 72,043 unique points. If we instead use Census blocks, 836,419 unique points define the same district — this results in *at least* a ten-fold increase in algorithmic running time. To reduce the number of points, we take advantage of the approximation lemma. Here, we define a routine **HullPoints** which takes collections of units $\mathcal{U}_i$ as input, and outputs the union of the points defining those units' convex hulls.

Algorithm 2 (**ConvexHull**) computes minimum bounding circles using only the points defining the convex hull of each unit comprising each district, rather than using points defining the entire unit.

if $t = 1$ **then**
  $M \leftarrow \{1, \ldots, k\}$ ▷ Initially, we compute circles for all districts.
else
  $M \leftarrow \{i, j\}$ ▷ Only districts i and j were modified by ReCom.
end if

for each district index d in M **do**
  $S_d = \textbf{HullPoints}(P_t(d))$ ▷ Points defining convex hulls of units in district d.
  $C_d = \textbf{MinCircle}(S_d)$ ▷ Compute the minimum bounding circle of d.
end for

return $(C_1, \ldots, C_k)$ ▷ Sequence of minimum enclosing circles.

Correctness. Algorithm 2 only slightly modifies Algorithm 1: instead of computing the minimum bounding circle of all points on units composing district d, it computes the minimum bounding circle of points on the *convex hulls* of units in d. By Lemma 3.3, we know that the convex hull of the union of polygons is the same as the convex hull of the union of the polygons' convex hulls. This algorithm then correctly computes convex hulls, and consequently correctly computes minimum bounding circles. ■

Next, we can introduce a second optimization justified by Lemma 3.2. This algorithm uses a subroutine called **Boundary** which, given a district index d, finds the units on the boundary of d. Though this is somewhat difficult in a geometric setting — finding which units' boundaries touch the dividing line between two districts, or coincide with the boundary of the entire jurisdiction — our dual environment lets us abstract geometric coordinates away in favor of adjacency relationships. Our graph-based abstraction secures an important computational advantage, as it strips out unnecessary information and allows us to pre-compute boundary vertices and quickly identify cut vertices, which is important in the use of Lemma 3.2.

Algorithm 3 (**Donut**) computes minimum bounding circles using only the points defining the convex hull of each unit comprising each district, rather than using points defining the entire unit.

if $t = 1$ **then**
  $M \leftarrow \{1, \ldots, k\}$ ▷ Initially, we compute circles for all districts.
else
  $M \leftarrow \{i, j\}$ ▷ Only districts i and j were modified by ReCom.
end if

for each district index d in M **do**
  $B_d = \textbf{Boundary}(d)$ ▷ Units forming the "donut" around d.
  $S_d = \textbf{HullPoints}(B_d)$ ▷ Points defining convex hulls of the donut of d.
  $C_d = \textbf{MinCircle}(S_d)$ ▷ Compute the minimum bounding circle of d.
end for

return $(C_1, \ldots, C_k)$ ▷ Sequence of minimum enclosing circles.

Correctness. Algorithm 3 adds a single step to Algorithm 2 as we incorporate the use of Lemma 3.2. Because Lemma 3.2 says that we can use only the boundary units (and, by extension, the points defining those units) to correctly compute global convex hulls, and the other steps in this algorithm are justified by the proof of Algorithm 3, this algorithm correctly computes minimum bounding circles. ■

Finally, we are able to complete the optimization by using Lemma 3.1: rather than compute the minimum bounding circle of each set of hull points S_d, we first compute the convex hull $\mathrm{CH}(S_d)$ of the units, then compute $\mathrm{MinCircle}(\mathrm{CH}(S_d))$.[2]

Algorithm 4 (**COMBINED**) computes minimum bounding circles using only the points defining the convex hull of each unit comprising each district, rather than using points defining the entire unit.

if $t = 1$ **then**
  $M \leftarrow \{1, \ldots, k\}$ ▷ Initially, we compute circles for all districts.
else
  $M \leftarrow \{i, j\}$ ▷ Only districts i and j were modified by ReCom.
end if

for each district index d in M **do**
  $B_d = \textbf{BOUNDARY}(d)$ ▷ Units forming the "donut" around d.
  $S_d = \textbf{HULLPOINTS}(B_d)$ ▷ Points defining convex hulls of the donut of d.
  $C_d = \textbf{MINCIRCLE}(\mathrm{CH}(S_d))$ ▷ Compute $\mathrm{MinCircle}()$ of d based on its convex hull.
end for

return $(C_1, \ldots, C_k)$ ▷ Sequence of minimum enclosing circles.

Correctness. Algorithm 4 adds a single step to Algorithm 3 in its use of Lemma 3.1. Because this lemma shows that the minimum bounding circle of S_d is the same as the minimum bounding circle of the hull of S_d, this algorithm correctly computes minimum bounding circles. ■

4 Results, discussion, and future work

To test our theoretical findings, we simulate a Markov chain, generating 1,000 nine-district Congressional districting plans in the state of Indiana.[3] We then program subroutines for each of the minimum bounding circle-finding algorithms described in Section 3.1 and perform these subroutines on every plan produced by the chain: this way, we can directly compare the efficiency of each algorithm, as they are performing different operations on the same input data. Table 1 shows a sample of these results, and they clearly show the benefits of these optimizations — they reduce the time it takes to find minimum bounding circles by literally thousands of times.

As expected, Algorithm 1 (**NAÏVE**) performs abysmally, requiring more than eight hours of processing time to compute 9,000 minimum bounding circles. As expected from our earlier point-counting estimates, **CONVEXHULL** makes an immediate and considerable performance improvement: it needed no more than 20 minutes to compute

[2] Lemma 3.1 is used implicitly in each of the preceding algorithms, as the minimum bounding circle of $\mathcal{U}_i$ depends on the convex hull of $\mathcal{U}_i$. However, we use it here as an explicit optimization to reduce the number of points involved in the calculation, rather than treat it solely as justification for using convex hulls of individual units.

[3] We use a short-length chain for this experiment: shorter chains are easily repeatable, and neither the autocorrelation of districts or the convergence of the chain to its stationary distribution have any bearing on the efficiency of minimum bounding circle computation.

	NAÏVE	CONVEXHULL	DONUT	COMBINED
Time (seconds)	35.4941	1.0452	0.0374	0.0306
	37.3586	0.9975	0.0363	0.0261
	37.6074	1.0829	0.0344	0.022
	37.5681	1.0738	0.0364	0.0326
	39.7257	1.2078	0.0422	0.0491
	29.0643	1.2098	0.0559	0.0413
	24.8241	1.2456	0.0366	0.0398
	18.0935	1.2292	0.0471	0.0194
	34.0147	1.259	0.0353	0.0382
	19.6954	1.065	0.04	0.0277

Table 1: Minimum bounding circle computation times for a uniformly random sample of 10 plans from the 1,000-plan ensemble. Lemma 3.3 is a home run, improving computation times by (at most) a factor of 40; Lemma 3.2 also provides significant efficiency advantages, further reducing running times by similar amounts.

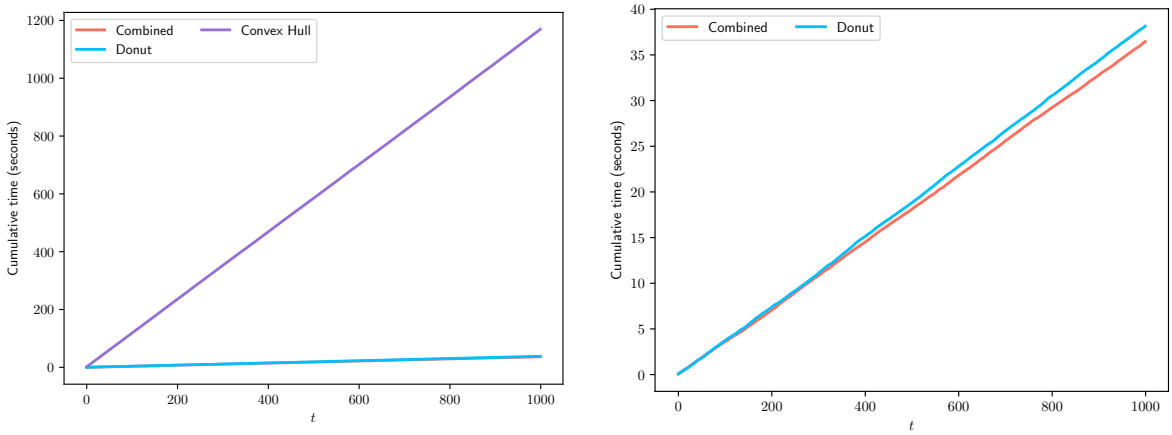

Figure 8: Direct comparison of **CONVEXHULL**, **COMBINED**, and **DONUT** (left); direct comparison of **COMBINED** and **DONUT**, the two best-performing algorithms (right).

the same number of bounding circles. To re-frame this result, **CONVEXHULL** required only *four percent* of the time that **NAÏVE** did to compute exactly the same information. Figure 8 zooms in on these performance gains, making plain the superiority of **DONUT**: it needed only *three percent* of the time that **CONVEXHULL** did to find the same results. Compounded, **DONUT** is nearly a thousand-fold efficiency increase over **NAÏVE**. Using all the results together, **COMBINED** is a slight performance boost over **DONUT**.

Though we provide substantive theoretical and practical improvements for handling geometric data in MCMC contexts, more work in this area remains. Storing and accessing geometric data is, at present, prohibitively clunky: points which induce k polygons are stored k times, leading to unnecessarily bloated datasets; retrieving geometric information associated to vertices in the dual graph requires a linear-time scan at each step of the chain per district modified. We also want to better characterize the "shape" of our input data: based on its underlying algebraic and topological structure, can we determine whether specific configurations of units might occur? Are there easy ways to identify such configurations on-the-fly, and can we tailor algorithms to specially handle them?

Because these MCMC-based techniques require both substantial time and computing resources, and because the redistricting investigations they support move rapidly, algorithmic efficiency is of key importance. Rectification of civil rights harms simultaneously demands care and speed: these problems are firmly entrenched in our electoral systems, and do material harm to those who suffer them. Well-designed, thoughtful algorithms can lead to better representational outcomes, and the improvements described here are a small but meaningful step in that direction.

References

[1] A.M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979.

[2] Mira Bernstein and Moon Duchin. A Formula Goes to Court: Partisan Gerrymandering and the Efficiency Gap. *Notices of the American Mathematical Society*, 64(9):1020–1024, 2018.

[3] Daryl DeFord, Natasha Dhamankar, Moon Duchin, Varun Gupta, Mackenzie McPike, Gabe Schoenbach, and Ki Wan Sim. Implementing partisan symmetry: Problems and paradoxes. *Political Analysis*, page 120, 2021.

[4] Daryl DeFord, Moon Duchin, and Justin Solomon. Recombination: A Family of Markov Chains for Redistricting. *Harvard Data Science Review*, 3(1), mar 31 2021.

[5] Nimrod Megiddo. Linear-Time Algorithms for Linear Programming in $\mathbb{R}^3$ and Related Problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

[6] MGGG Redistricting Lab. GerryChain. https://github.com/mggg/GerryChain.

[7] MGGG Redistricting Lab. The Known Sizes of Grid Metagraphs, 2018.

[8] Daniel D. Polsby and Robert D. Popper. The Third Criterion: Compactness as a Procedural Safeguard against Partisan Gerrymandering. *Yale Law and Policy Review*, 9(2):301–353, 1991.

[9] Ernest C. Reock. A Note: Measuring Compactness as a Requirement of Legislative Apportionment. *Midwest Journal of Political Science*, 5(1):70–74, 1961.

[10] Joseph E. Schwartzberg. Reapportionment, Gerrymanders, and the Notion of Compactness. *Minnesota Law Review*, 50, 1965.

[11] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In Hermann Maurer, editor, *New Results and New Trends in Computer Science*, Lecture Notes in Computer Science, page 359370, Berlin, Heidelberg, 1991. Springer.