Scientific Research Publishing

Journal of Information Security, 2022, 13, 140-164
https://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

Blockchain Based Redistricting with Public Participation

Mahalingam Ramkumar[1], Naresh Adhikari[2]

[1]Mississippi State University, Starkville, MS, USA
[2]Slippery Rock University, Slippery Rock, PA, USA
Email: ramkumar@cse.msstate.edu, naresh.adhikari@sru.edu

How to cite this paper: Ramkumar, M. and Adhikari, N. (2022) Blockchain Based Redistricting with Public Participation. *Journal of Information Security*, **13**, 140-164.
https://doi.org/10.4236/jis.2022.133009

Received: May 23, 2022
Accepted: July 8, 2022
Published: July 11, 2022

Abstract

Redistricting is the process of grouping all census blocks within a region to form larger subdivisions, or districts. The process is typically subject to some hard rules and some (soft) preferences to improve fairness of the solution. Achieving public consensus on the fairness of proposed redistricting plans is highly desirable. Unfortunately, fair redistricting is an NP hard optimization problem. The complexity of the process makes it even more challenging to convince the public of the fairness of the proposed solution. This paper proposes a completely transparent blockchain based strategy to promote public participation in the redistricting process, to increase public confidence in the outcome of the process. The proposed approach is based on the fact that one does not have to worry about how the NP hard problem was solved, as long as it is possible for anyone to compute a "goodness" metric for the proposed plan. In the proposed approach, anyone can submit a plan along with the expected metric. Only the plan with the best claimed metric needs to be evaluated in a blockchain network.

Keywords

Redistricting, Authenticated Data Structures, Blockchain Ledger

1. Introduction

Redistricting is a process of subdividing a geographical region into districts such that each district meets similarity criteria such as geographic contiguity, equality of population, etc. Some of the basic principles [1] underlying the congressional redistricting process include 1) contiguity of districts, 2) equality of population size, 3) geographic compactness, 4) protection against vote dilution of minorities, 5) preservation of the boundaries of other political subdivisions, 6) promo-

tion of electoral competition, 7) prohibition of partisan considerations, etc.

In the United States, congressional redistricting is performed every 10 years to account for dynamics in population. Specific redistricting principles, however, may differ among states. Congressional redistricting may be performed by a commission of political representatives, or by independent experts. Redistricting by a politically biased committee may lead to what is known as gerrymandering [2]. A gerrymandered redistricting process produces districts that give an advantage [3] [4] to one party over another in elections.

Ideally, redistricting seeks to create equal-population districts that are optimally compact, thereby optimizing some cost. A redistricting plan that maps b blocks to d districts has b^d possible solutions, making it an NP-hard problem [5]. While the notion of using sophisticated computer algorithms to automate redistricting to produce a fair district plan is appealing, the public often lacks trust in the computer platforms and algorithms. The lack of trust is not unjustifiable, especially for complex platforms needed to solve NP-hard problems. Sophisticated computer algorithms have been known to be misused to produce visually satisfying, but with subtle deliberate sub-optimal tweaks to create politically biased districts [5].

1.1. Problem Statement

Let the b blocks in a region R be $B_1 \cdots B_b$. Any block B_i is a polygon with n_i sides, typically represented by a sequence of $n_i + 1$ boundary points (or vertices of a polygon) in the counter clockwise (CCW) order, *viz.*,

$$B_i = \left\{(x_1, y_1), (x_2, y_2), \cdots, (x_{n_i}, y_{n_i}), (x_1, y_1)\right\}. \quad (1)$$

The last point is the same as the first point. Traversing the polygon in the CCW direction leaves the area enclosed by the polygon to the left. **Figure 1** depicts a region R with 10 blocks, grouped into 3 districts.

Assigning the b blocks to d districts is a process of creating d groups $D_1 \cdots D_d$ among the b blocks, subject to some hard and soft constraints. For our purposes we shall assume that the region is a state, and the districts are congressional districts. Hard constraints dictate that the blocks in a group should be contiguous (a block should share at least one side with another block in the same group), no

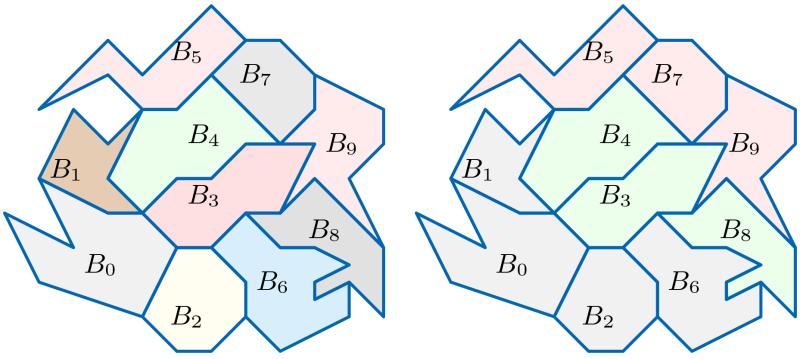

Figure 1. A region R with 10 blocks grouped into 3 districts.

overlap between blocks, and no overlap between districts. Some soft constraints include equal-population, compactness, convexity, and proximity of blocks in a district to the "center" of the district. The center, could be a geographic centroid, or the population centroid (which pulls the geographic centroid closer to more populated blocks). Other soft constraints may include consideration of racial mix, annual income, political leanings, etc.

In the United States, census blocks are public data [6], available in shapefile (.shp) format [7]. The shapefiles for states range from 4MB for the District of Colombia, to 760 MB for Texas. The total number of census blocks in the U.S.A is over 6 million. For example, the state of California has $b = 710145$ blocks grouped into $d = 53$ districts.

1.1.1. The Challenge

The computing process of interest to us in this paper is required to take possibly up to a million blocks as input, where each block is represented by possibly hundreds or even thousands of points. Each block may also be associated with non geographic data like population, racial mix, average annual income, political leanings, etc. The input will also include a set of weights $w_i \cdots w_n$ applied to various soft metrics to influence the output of the process, viz., the allocation of blocks to each group.

The utility of any computing process is ultimately limited by the extent of trust in the correctness of output of the process. In general, higher the complexity of the platform, the harder it is to ensure that every component of the platform is trustworthy. Given b blocks and d districts there are b^d possible ways of grouping the blocks ($b^d = 710145^{53}$ for California). Given the enormity of the computational requirement at hand—choosing the best out of b^d options—it is far from easy to justify the trust in the correctness of the output.

One approach to enhance trust in the integrity of platforms is to minimize the trusted computing base (TCB), the minimal amount of hardware/software that needs to be trusted [8] to guarantee platform integrity. In von Neumann (VN) computers—which is almost every computing platform today—each step in the execution of any process (a machine instruction) requires memory access. If data previously stored in some memory location at time t_s is retrieved later at time t_r, it is simply assumed that the memory contents were not modified between times t_s and t_r. This potential time-of-check-time-of-use (TOCTOU) [9] vulnerability is simply ignored in the VN model, by including external memory inside the TCB.

Unfortunately, trust in external memory is unjustifiable in modern computers with a complex hardware/software stack [10] [11] [12] [13]. A significant number of attacks that exploit this weakness, such as RAM scraping [14], buffer overflow[15], file-less malware [16], DMA malware attacks [17], invasive or semi-invasive attacks [18] etc., have repeatedly demonstrated exploits that illegitimately read/write memory.

1.2. Proposed Solution

The proposed solution is motivated by the fact that one does not have to be concerned about **how** the computation was actually performed, as long as one can **verify** the results of the computation. For NP optimization problems, while computing a solution may have exponential complexity, verifying the correctness of a proposed solution is typically a polynomial time algorithm. However, even while verification is substantially easier, the scale of data can make it hard to promote confidence in computing platforms used for verification.

A blockchain network is essentially a computing platform where no component needs to be included inside the TCB [19] [20] [21]. In the proposed approach, the verification algorithm is executed in a blockchain network. Specifically, the verification algorithm computes a goodness metric [22] for any proposed solution.

Anyone can submit a solution. In order to deter submission of incorrect solutions, it can be mandated that each submission should explicitly indicate the final metric, and be accompanied by a stake. The entity submitting the solution will lose the stake if the computed metric does not match the claimed metric. If the stake is high enough to serve as a deterrent, only one proposed solution (the one that claims the best metric) will need to be executed in the blockchain network.

1.3. Organization

The rest of the paper is organized as follows. Section 2 begins with a discussion of various measures of optimality of redistricting plans. Section 2.1 provides a overview of an important component of the proposed solution, *viz.*, hash tree based authenticated data structures (ADSes). Section 2.2 is an overview of blockchain networks as a platform for computing. More specifically, executing a process in a blockchain calls for a state-machine model for the process, where permitted state-changes are well-formed transactions.

Section 3 discusses 3 distinct steps involved in the verification of redistricting plans, and outlines the scope of different types of blockchain transactions in each step. Additional discussions and conclusions are offered in Section 4.

2. Overview

Let D be a simple polygon whose n sides are described by Cartesian coordinates $(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1}, (x_0, y_0))$ in the counter clockwise (CCW) direction, along the perimeter of the polygon. The area A [23], the perimeter P, and the coordinates of the centroid $C = (C.x, C.y)$ [24] of the polygon D, can be computed incrementally by considering 2 adjacent points at a time, as

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i + x_{i+1})(y_{i+1} - y_i) \tag{2}$$

$$P = \sum_{i=0}^{n-1}\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \tag{3}$$

$$C.x = \frac{1}{6A}\sum_{i=0}^{n-1}\left(x_i + x_{i+1}\right)\left(x_i y_{i+1} + x_{i+1} y_i\right)$$

$$C.y = \frac{1}{6A}\sum_{i=0}^{n-1}\left(y_i + y_{i+1}\right)\left(x_i y_{i+1} + x_{i+1} y_i\right) \quad (4)$$

Geometric compactness [25] [26] [27] is a commonly used metric for evaluating the suitability of a proposed solution. Several measures of compactness exist for a polygon D with area A and perimeter P.

1). Iso-perimetric ratio, computed as $I_{ip}(D) = A/P^2$;

2) Reock ratio $R = A/A_C$ [25] where A_C is the area of the smallest circle bounding the polygon D (with area A);

3) The convex hull ratio $H_D = A/A_{H_D}$ where A_{H_D} is the area of the convex hull of D.

For a district D with k blocks, the moment of inertia is computed as $I(D) = \sum_{i=1}^{k} w_i d_i^2$, where d_i is the distance between the centroid of the district and the centroid of block i. The weight w_i for block i can be based on the area of the block, or the population of the block, or more often, a combination of both.

2.1. Authenticated Data Structures

An authenticated data structure (ADS) [28]-[32] is a mechanism for computing a cryptographic commitment s to any number of discrete data items in a collection $\mathbf{S}$. Useful ADSes, typically based on hash trees, possess efficient strategies to incrementally update the commitment $s(t)$, with incremental changes to $\mathbf{S}(t)$.

Commitments are hashes, computed using a cryptographic hash function $h()$. Given $y = h(x)$ with pre-image x and digest y, it is impractical from a computational perspective to determine any $x' \neq x$ satisfying $y = h(x')$. One can thus reasonably conclude that digest y was computed **after** x was chosen, or that x "existed before" its *commitment* y.

2.1.1. Merkle Hash Tree

A Merkle hash tree [33] is an example of an ADS where the root of the tree (a single hash) is a commitment to a set of dynamic leaves. **Figure 2** depicts a tree with $s(t) = v_{0,0}$ (root of the inverted tree) and $\mathbf{S}(t) = \{L_0 \cdots L_7\}$ as the 8 leaves of the tree. For a balanced tree with $N = 2^d$ leaves, there are 2^i nodes in level $i = 0,1,\cdots,d$, where $d = \log_2 N$ is the maximum depth of the balanced tree.

A node $v_{r,k}$ at depth r, $0 \le k < 2^r$ is computed as

$$v_{r,k} = h\left(v_{r+1,2k}, v_{r+1,2k+1}\right). \quad (5)$$

A leaf-node is obtained by hashing the contents of the leaf. In **Figure 2**, leaf-node $v_{3,5} = h(L_5)$.

Any node at depth r has r *complementary* nodes, and r *ancestor* nodes. The 3 blue nodes in the figure are ancestors of node $v_{3,5}$. The red nodes—the node's own "sibling" of $v_{3,4}$, and the siblings $v_{2,3}$ and $v_{1,0}$ of its ancestors—are

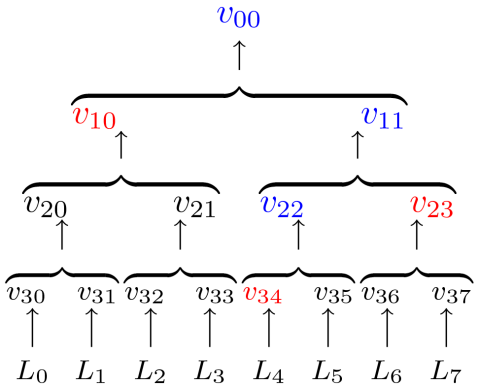

Figure 2. A Merkle hash tree of depth $d=3$ with $N=2^d=8$ leaves.

complementary to $v_{3,5}$. The complementary nodes of $v_{3,5}$ are the *verification objects* (VOs) for node $v_{3,5}$.

2.1.2. Prover-Verifier Protocols

For any one with access to only the root $v_{0,0}$ of the tree, knowledge of VOs of $v_{3,5}$, *viz.*, $\{v_{3,4}, v_{2,3}, v_{1,0}\}$, constitutes **proof of existence** of $v_{3,5}$ in the tree. Note that setting $x = v_{3,5}$ and following it with a sequence of 3 hash operations using the VOs, viz., $x = h(v_{3,4}, x)$, $x = h(x, v_{2,3})$, $x = h(v_{1,0}, x)$ will yield $x = v_{0,0}$ (the known root of the tree). Due to the one-way property of cryptographic hash functions, that $v_{3,5} = h(L_5)$ exists in the tree, is also proof of existence of leaf L_5.

If there is a legitimate reason to update $v_{3,5} \to v'_{3,5}$, using the same sequence of hash operations starting with $x = v'_{3,5}$ instead, one can compute the new value of the root, using the same VOs $\{v_{3,4}, v_{2,3}, v_{1,0}\}$. It is important to note that VOs of $v_{3,5}$ are unaffected by changes to $v_{3,5}$.

The need for an update to $v_{3,5}$ may be for purposes of changing the contents of leaf L_5, or adding a new leaf by increasing depth of L_5. For example, a new leaf (say) L' can be inserted alongside L_5, making $v_{4,10} = h(L_5)$ the new leaf node of L_5, and $v_{4,11} = h(L')$ the leaf-node for the new leaf, thus changing $v_{3,5}$ to $v_{35} = h(v_{4,10}, v_{4,11})$. Just as inserting a leaf promotes a leaf node to "a parent of 2 leaf nodes," deletion of a leaf will result in "a parent of 2 leaf nodes" being demoted to a leaf node.

For a tree with N leaves, $\log_2 N$ VOs can be used to prove

1) the existence of a specific leaf $L \in \mathbf{S}(t)$, or

2) that the new root is $s(t_+)$, corresponding to an update $L \to L'$ to a leaf, or

3) that the new root is $s(t_+)$, corresponding to insertion/deletion of a leaf.

In prover-verifier protocols based on ADSes, provers store all leaves and nodes. Verifiers are assumed to know only the root $s(t)$ of the tree. Provers can prove specific properties regarding a data collection $\mathbf{S}(t)$ (leaves of the tree) against the commitment $s(t)$ (root of the tree). If the total number of leaves is $|\mathbf{S}(t)| = N$, the proof takes the form of $\mathbb{O}(\log_2 N)$ VOs. Verifiers can

verify the proof by computing $\mathbb{O}(\log_2 N)$ cryptographic hash operations using the $\mathbb{O}(\log_2 N)$ VOs, and comparing the end result to the known commitment $s(t)$. The proofs also permit verifiers to update the commitment $s(t)$ corresponding to legitimate updates to $\mathbf{S}(t)$ (changing the contents of leaves, inserting/deleting leaves, etc.).

2.1.3. Geographic Collections

In the proposed approach, the leaves (of a hash tree/ADS) will correspond to one or more collections of geographic elements like

1) a side of a polygon, specified by 2 points (2 sets of (x, y) coordinates), or

2) a rectangular bounding box, also specified by 2 points (lower-left and upper-right corners), or

3) a more general key-value (k-v) collection.

For a geographic point $p = (p.x, p.y)$, $p.x$ and $p.y$ are typically latitude and longitude coordinates. Collectively, geographic elements in a collection may represent more complex objects like a polygonal region [34].

Leaves in a *boundary line* collection are of the form $[p_1, p_2, v]$, representing a boundary line connecting the 2 points. The value v may provide some information about the line. In such collections, an example of a meaningful update is to split a leaf $[p_1, p_2, v]$ (representing a boundary line) into 2 leaves as

$$[p_1, p_2, v] \to \{[p_1, p, v], [p, p_2, v]\}, \tag{6}$$

at a redundant point p on the line. Such an incremental operation replaces a leaf with 2 leaves (or a new leaf is added to the tree).

In a rectangular *bounding box* collection, leaves are of the form $[c_1, c_2, v]$, specifying the lower-left corner point c_1 and the upper-right corner point c_2. A rectangular bounding box can also be split into 2, for example, by a vertical line at X coordinate value $c_1.x < x_s < c_2.x$ as

$$[c_1, c_2, v] \to \begin{cases} [c_1, c, v] \\ [c', c_2, v] \end{cases} \quad c.x = c'.x = x_s,\ c.y = c_1.y,\ c'.y = c_2.y. \tag{7}$$

or across a horizontal line with Y-coordinate value $c_1.y < y_s < c_2.y$, as

$$[c_1, c_2, v] \to \begin{cases} [c_1, c, v] \\ [c', c_2, v] \end{cases} \quad c.y = c'.y = y_s,\ c.x = c_1.x,\ c'.x = c_2.x. \tag{8}$$

2.1.4. Key-Value Collections

In a key-value (k-v) collection [30] the leaves are of the form $[k, k_n, v]$ corresponding to a k-v pair $\{k, v\}$ with *next-key* k_n.

The first leaf added to an empty tree should be of the form $[k, k, v]$. A k-v pair $\{k, v\}$ can be added to the collection only if a leaf $[a, b, u]$—a k-v pair $\{a, u\}$ with next-key b—*already* exists in the collection such that

$$(a < k < b) \text{ OR } (b \le a < k) \text{ OR } (k < b \le a). \tag{9}$$

If the condition above is satisfied, the existing leaf $[a, b, u]$ is replaced with two leaves

1) $[a,k,u]$: old k-v pair $\{a,u\}$, but with K as next-key, and
2) $[k,b,v]$: new k-v pair $\{k,v\}$ with b as next-key.

2.1.5. Hierarchical Hash Trees

ADSes constructed using hash trees can have hierarchical structures. For example, one or more values in a leaf can themselves be a hash tree root.

As an example, assume that the verifier knows only the root ξ of a hierarchical ADS. Two sets of VOs can be used to prove that

1) "boundary line $[p_1, p_2]$ exists in a tree with root ξ_b" and
2) "$[b, \xi_b]$ exists in a tree with root ξ."

Such hierarchical collections will be very useful for our purposes. For example, the tree with root ξ may have a leaf for every block in a region. The value of the leaf for a block B_i may include a root ξ_v for a collection of boundary lines for the block.

2.1.6. State-Change Functions

The practical utility of such ADSes is that any entity with access to merely the root s of the hash tree (with such geometric objects as leaves), can verify proof (using VOs) that "a boundary line (p_1, p_2) exists in a polygon represented by hash tree with root s, and that splitting the line into two lines (p_1, p) and (p, p_2) will result in a new commitment s'."

The ability to enable verification of useful properties regarding process specific data $\mathbf{S}(t)$, and more importantly, update commitments in step with incremental changes to $\mathbf{S}(t)$, can be leveraged using "some external mechanism" to maintain *universal consensus* on the dynamic commitment $s(t)$ *at all times*.

Such a mechanism is a blockchain network, where updates to $\mathbf{S}(t)$ (and hence, the commitment $s(t)$) are due to *well-formed transactions*. Specifically, different types of transactions may be permitted for a process (for example, splitting a leaf, splitting a bounding box, etc.), each of which results in an easily calculable change to the commitment $s(t)$.

More generally, given a transaction T of type k, the current commitment $s(t)$, and a small set of VOs v, any one can reliably calculate the updated commitment as

$$s(t_+) = F_k\left(s(t), T, \mathbf{v}\right). \tag{10}$$

In other words, validating the correctness of a **state-change** function $F_k(\,)$ (for a transaction type k) will call for $\log_2 N$ hash operations, using $\log_2 N$ VOs $\mathbf{v}$. In blockchain networks, the transaction T triggering the state-change, and the updated state $s(t_+)$ after the transaction, are logged in the blockchain ledger.

2.2. Blockchain Networks

A blockchain network is a mechanism that employs a broadcast channel for maintaining consensus on the contents of a blockchain ledger. Only well-formed transactions (also broadcast over the channel) can be added to the ledger. What

makes a transaction well-formed will depend on process specific rules.

2.2.1. Ideal vs Practical Blockchain Networks

In an ideal blockchain network, participation is *universal*. The broadcast channel is *ideal*—every participant will hear every broadcast at the same time; every participant can then decide on their own, if the transaction is well-formed, and if so, add the transaction to the ledger. Every ledger entry changes the *ledger-hash*—a cryptographic commitment to the entire ledger. Specifically, if the current ledger with i transactions has a ledger hash d_i, then the ledger hash after adding the next transaction T_{i+1} is $d_{i+1} = h(d_i, T_{i+1})$.

If everyone is honest, everyone will maintain identical ledger copies, with the same ledger-hash. Broadcasting the succinct ledger hash makes it easy for every one to confirm that they do indeed maintain identical ledger copies.

In practical blockchain networks a ledger entry is not a transaction, but a commitment to a block of transactions. More specifically, each transaction is seen as a leaf of a hash tree, and the root of the tree is the commitment to the entire block; the ledger hash is thus a commitment to a "chain of block (hashes)."

In practical networks, only a small fraction of users may participate actively at any time. The broadcast channel, typically realized using a multi-hop peer to peer network, is far from ideal. Broadcasts may be heard at different times by different participants depending on the number of hops. Some participants may never hear some broadcasts. Furthermore, we cannot simply assume all participants to be honest. Even honest participants may sometimes have genuine disagreements regarding the correctness of a transaction. Even differences in time-of-arrival of transactions can cause differences between blocks added to different copies of ledgers, thereby creating **forks** in the ledger.

In spite of the practical limitations, it is possible to use well designed *incentive mechanisms* [35] to ensure that a) only well-formed transactions are added to the ledger, and b) close to universal consensus is maintained on the correctness of ledger entries. An incentive mechanism has three main goals a) motivate participation to ensure a reasonably large number of participants at all times; b) serve as a mechanism for penalizing ill-formed ledger entries; and c) minimize the chance of long-term survival of a forked ledgers.

2.2.2. Explicit vs Implicit States

A blockchain ledger is a log of state-changes due to transactions. For example, in crypto-currency applications, a transaction $[T(t): A \to B, c]$ at time t transfers c coins from wallet A to wallet B, reducing wallet A balance by c and increasing wallet B balance by c. The state of the application at any time t can be seen as the remaining balance is every wallet.

For this application, the transaction is considered as well-formed only if a) the transaction is signed by A, and if b) A had c or more coins prior to the transaction. Ill-formed transactions are simply ignored, and thus do not change the state of the application. As an example that is more relevant to this paper, a

transaction to split a boundary line $[p_1, p_2, v]$ into $[p_1, p, v]$ and $[p, p_2, v]$ is considered as well-formed **only if** p lies on the line connecting p_1 and p_2 (as splitting the line at a redundant point p does not change the shape or area inside the polygon).

In some networks like Bitcoin [36] the process state after each transaction is **not** made explicit in the transactions. In Ethereum [37], on the other hand, every transaction is accompanied by a cryptographic commitment to the updated state of the application/process affected by the transaction. In this paper, we restrict ourselves to transactions with explicit states.

2.2.3. Progression of Process States

In general, any process $\mathcal{P}$, with process states $\mathcal{S}$, can be executed in a blockchain network, if the process is modeled as state-machine. In such a model, a process $\mathcal{P}$ is defined by m types of transactions (or state-change functions). A transaction T^j of type j is executed only if T^j is well-formed. Execution if T^j results in a change in the process state $\mathcal{S}$. The progression of states of process $\mathcal{P}$ can be represented as

$$\mathcal{S}_0 \xrightarrow{T_1^{j1}} \mathcal{S}_1 \xrightarrow{T_2^{j2}} \mathcal{S}_2 \cdots \mathcal{S}_{n-1} \xrightarrow{T_n^{jn}} \mathcal{S}_n, \cdots \tag{11}$$

The utility of ADSes stems from their ability to capture any number of dynamic process states as leaves of a (possibly hierarchical) hash tree, whose root s is a commitment to all process states. The progression of states changes in Equation (11) can equivalently be represented as

$$s_0 \xrightarrow[VO_1]{(T_1^{j1})} s_1 \xrightarrow[VO_2]{(T_2^{j2})} s_2 \cdots s_{n-1} \xrightarrow[VO_n]{(T_n^{jn})} s_n, \cdots, \tag{12}$$

where the VOS accompanying each transaction serves as "proof of correctness" of the state-change.

3. Redistricting Verification Protocol

Given a set of blocks $B_1 \cdots B_b$ of a region R and districts $D_1 \cdots D_d$, any solution to the redistricting problem can be seen as a collection of b key-value pairs $\{B_i, D_j\}, 1 \le i \le b, 1 \le j \le d$. The conditions that need to be satisfied for the solution to be considered as correct are as follows:

1) No block is left unassigned;

2) No block is assigned to more than 1 district;

3) Blocks assigned to a district are contiguous.

While numerous soft restrictions can be imposed, in this paper we shall limit ourselves to equal-population and compactness metric. We shall use the normalized standard deviation of the population as a measure for deviation of population between districts. For the compactness metric we can use the moment of inertia I, or more specifically

$$I_j = \sum_{j=1}^{d} \sum_{i=1}^{n_j} d_i^2 A_{ij} \tag{13}$$

where n_j is the number of blocks in district D_j, A_{ij} is the area of the i^{th} block in district D_j, and d_i is the distance between the centroids of district D_j and the i^{th} block.

3.1. ADSes for Blocks, Districts and State

As the input to the process may involve tens of millions of points, and as data stored on general purpose computers are far from secure, the process involves constructing different types of ADSes such that a single commitment can be used to verify the integrity of any geographic data item, and correctness of any transaction performed to change data. For geographic data involving a billion points (or $N \approx 2^{30}$), only of the order of $\log_2(N) = 30$ VOS are required for verifying proof of integrity of any data item or proof of correctness of any transaction that makes an incremental change to data.

The verification protocol can be seen as consisting of 3 steps,

1) Constructing a block-ADS with b blocks, for validating the inputs, viz., points corresponding to every block.

2) Constructing a district-ADS with d districts, for evaluating a proposal specified by a key-value collection (with b key-value pairs of the form $\{B_i, D_j\}$, conveying that B_i is assigned to district D_j).

3) Constructing a "state-level" ADS.

Processes in the first step ensure that the possibly tens of millions of points representing possibly hundreds of thousands of blocks do indeed constitute a set of valid polygonal regions. For a block with n points, $\mathbb{O}(n)$ transactions will be called for; the total number of transactions will be proportional to the total number of points in the entire state. Consequently, the first step is possibly the most expensive one. Fortunately, this step will need to be performed only once. Once a block ADS is available for all blocks in the state, any one can access the data and confirm the validity of the block boundaries. The same block ADS can be used for any number of future redistricting attempts, as block boundaries typically remain stable.

The second step is a process for every district in the state. The complexity of such processes will depend on the number of district boundary points, which can be substantially less than the total number of points in the entire state. The purpose of the second step is to ensure that there are no overlaps between blocks included in any district, and to compute various metrics for each district.

The third step is a process for the entire state. The number of transactions for this process will be proportional to the number of state boundary points. The purpose of this is to ensure that districts do not overlap, and to compute overall metric for the state, and verify that the proposed metric is indeed correct.

The second and third steps will need to be repeated for every proposal. However, as mentioned earlier, if there is sufficient deterrent to incorrect proposals, only the solution with the best claimed metric will need to undergo the last 2 steps.

3.1.1. ADS Structures and Notations

The ADSes used at various steps include

1) the block ADS with root ξ_B, which is a collection with key-value pairs of the form $\{B_i, v_i\}$;

2) the District-ADS with root ξ_D, also a key-value collection with key-value pairs of the form $\{D_j, u_j\}$;

3) the "redistricting proposal," a key-value collection with root ξ_{BD}, with pairs of the form $\{B_i, D_j\}$.

4) a state ADS ξ_S with a single leaf $[s]$.

Values like v_i for block B_i ($\{B_i, v_i\}$ in tree with root ξ_B) and u_j for district D_j ($\{D_j, v_j\}$ in tree with root ξ_D) and value s for the state, include several values like area, centroid, population, multiple hash tree roots (for boundary-line collections, bounding box collections, key-value collections), and multiple point values needed to keep track of the progress of the process. Values associated with each polygonal structure like a block or a district or a state are summarized in Table 1.

For a block B_i, v_i conveys 10 values (in the top 5 rows of Table 1). The value u_j of a district D_j conveys 14 values (4 additional values A_u, p_{cu}, I and b_{ct}). The single state polygon is associated with 24 values.

In the rest of the paper we shall use the following notations for such values associated with blocks, districts and the state. For example, $v_i.p_b$ represents a point $p_b = (p_b.x, p_b.y)$ in value v_i of block B_i (in the block-ADS ξ_B). Tree

Table 1. Values associated with polygonal blocks, districts and state.

Values for All (Blocks, Districts and State) Polygons			
A	computed area	p_b	beginning of previous line
n	population	p_b	end of previous line
p_c	centroid	ξ_v	root of validated boundaries
p_s	start point	ξ_u	root of un-validated boundaries
ξ_r	root of bounding boxes		
ξ_r	root of redundant points		
4 More Values for District Polygons and State Polygon			
A_u	sum of block areas	p_{cu}	unverified centroid;
I	moment of inertia	b_{ct}	block count
10 More Values for State Polygon			
ξ_B	root of block ADS	ξ_D	root of district ADS
ξ_{BD}	redistricting plan (root of k-v collection)	ξ_d	root of district k-v collection
v_n	normalized variance	d_{ct}	number of districts
m_v	validated mean population	m_u	un-validated mean population
G	unverified metric	G_v	verified metric

roots in value v_i are represented as $v_i.\xi_u$, $v_i.\xi_v$, etc. Similarly, $u_j.p_b$, and $u_j.\xi_u$ represent a point and a tree root in the value u_j of district D_j, in the district-ADS ξ_D. For values associated with the state we shall use notations like $s.p_e, s.n, s.\xi_v$, etc.

All ADSes begin as empty trees (or $\xi_B = \xi_D = \xi_{BD} = \xi_S = 0$), and are incrementally constructed by blockchain transactions. In the rest of the paper we shall refer to "a tree with root ξ" as simply "tree ξ." We shall also use the notation $L \in \xi$ to imply that "L has been verified to exist in the tree with root ξ (using appropriate VOs)." For hierarchical collections such as the block ADS with root ξ_B, $L \in \xi_B.v_i.\xi_r$ implies $L \in v_i.\xi_r$, where $\{B_i, v_i\} \in \xi_B$.

3.2. Validating Polygons

A high level overview of the process for validating a polygonal region can be understood from Figure 3 with an eight sided polygon. By performing 2 types of simple operations, viz., splitting a rectangular bounding box (splits across red lines in the figure), and splitting boundary lines (at 6 bold points in the figure) it is possible to ensure that every boundary line fits wholly inside a rectangular box.

Some restrictions are imposed on mapping boundary lines to bounding boxes [34] to ensure that lines cannot cross each other inside a box, and make it easy to set the value v of the box to convey the lines mapped to the box. The restrictions are 1) no more than 2 lines may be mapped to a bounding box; 2) every line should start and/or stop at a corner of the BB; 3) in boxes with 2 boundary lines, both lines should be adjacent, and meet at a corner of the bounding box.

For the example in Figure 3, 6 transactions for splitting boundary lines at bold dots, and 20 transactions for splitting a bounding box vertically or horizontally (along red lines) will be needed to create conditions necessary for other transactions that map 1 boundary line, or map 2 adjacent boundary lines, to a

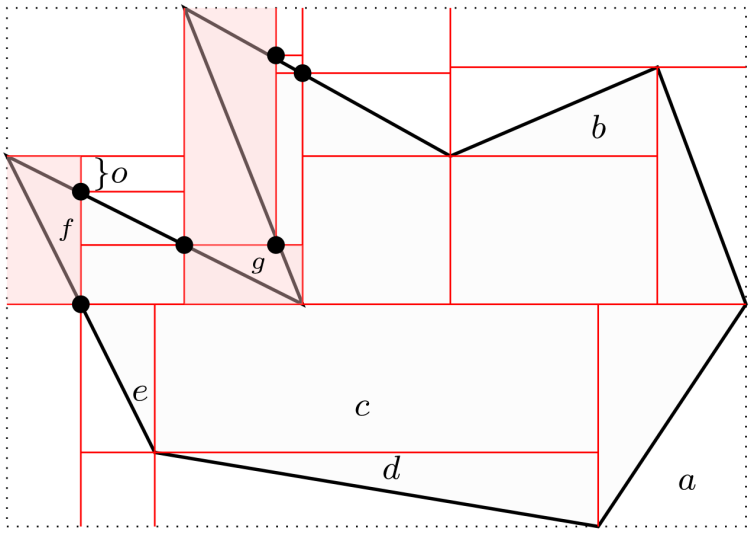

Figure 3. Process for validating a polygonal region.

bounding box. For the example in Figure 3, 8 transactions that map a single line, and 3 transactions that map 2 lines (3 red shaded boxes) are called for.

The boundary lines are initially available in the un-validated collection ξ_u. When a line from this collection ξ_u is shown to fit in a bounding box in ξ_r, the line is moved to a collection ξ_v of verified boundary lines (*i.e.*, a leaf is deleted from the tree with root ξ_u and added to the tree with root ξ_v). If all lines are mapped successfully, ξ_u will become empty, and ξ_v will hold all verified lines.

Lines have to be mapped sequentially, in the CCW order of points. A point value p_s is used to remember the starting point; point values p_b and p_e keep track of the beginning and end of the last-mapped line (by a previous transaction). To map a line $[p_1, p_2, v]$ it is necessary that $p_1 = p_e$ (the beginning of the current line is the end of the last-mapped line). An exception is the first line, for which $p_e = p_b = 0$, and causes $p_s = p_1$ to be set. The last line, with $p_2 = p_s$, will signify completion of mapping.

A transaction that maps a line $[p_1, p_2, v]$ incrementally computes the area as in Equation (2), and the centroid as in Equation (4). Specifically, the value A and coordinates $(p_c.x.p_c.y)$ of the centroid are incremented as

$$\begin{aligned} A+ &= p_1.x\,p_2.y - p_2.x\,p_1.y \\ p_c.x+ &= (p_1.x\,p_2.x)(p_1.x\,p_2.y + p_2.x\,p_1.y) \\ p_c.y+ &= (p_1.y\,p_2.y)(p_1.x\,p_2.y + p_2.x\,p_1.y) \end{aligned} \tag{14}$$

Transactions that map 2 adjacent lines $[p_1, p_2, v]$ and $[p_2, p_3, v]$ lines will repeat the same computation for the second line. Unless the unverified boundary lines points were specified in the CCW order, the computed area will be negative.

The constraints on mapping lines to bounding boxes prevent two lines from crossing each other **inside** a box. However, they do **not** prevent them from crossing **at** boundary points. Figure 4 illustrates such a situation with a bad polygon ABCD, in which AD and CB cross each other at point *E*. As shown in the figure, even with the restrictions, the polygon can still be mapped by splitting bounding boxes along dotted lines, to create a corner at *E*.

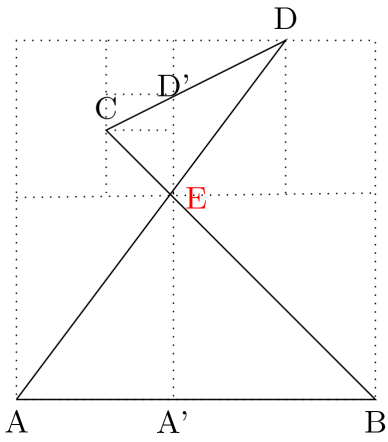

Figure 4. Redundant points in "bad" polygons.

Fortunately, unlike redundant points like D' (added by splitting side DA) and A' (by splitting side AB) in the figure, the redundant point E will occur **twice** in the polygon, as both DA and BC will need to be split at point E to "illegally" map the lines. Imposing a restriction that redundant points should be **unique** (as a well-formedness condition for transactions that map lines) will make it impossible to map polygons with intersecting sides.

When a line $[p_1, p_2, v]$ is mapped, the value p_b (of the last-mapped-line) can be used to determine if p_1 is a redundant point (*i.e.*, if $p_1 = p_e$ lies on the line connecting p_b and p_2). If so, the point is added to a key-value collection ξ_t as a key. If there are duplicate redundant points, the inability to add a redundant point to the key-value collection will cause the transaction to be deemed as ill-formed.

3.2.1. The "Value" of a Bounding Box

The value v in a bounding box $[c_1, c_2, v]$ can be set by transactions that map 1 or 2 lines to determine which areas inside the box fall inside the polygon [34]. The value v in every box with 1 or 2 mapped lines indicates i) a type τ; ii) an offset o; and iii) traversal direction (a binary flag) r. The 3 values unambiguously convey which parts of the bounding box (above/below a diagonal, above/below or to the left/right of a side of the box, or inside/outside the wedge shape formed by 2 lines) fall inside/outside the district.

15 types of bounding boxes can be created by mapping lines; type $\tau = 0$ with no lines; 2 types (types 1 and 2), where the line is a diagonal (one with positive slope, one with negative slope); 4 types (types 3 to 6) where a vertical or horizontal boundary line is one of the 4 sides of the box; and 8 types (types 7 to 14) with 2 lines in the box; the 8 different types are due to the 4 possible corners where the 2 lines meet, and if the shorter line is above/below the diagonal.

To uniquely identify the coordinates of the mapped line(s) only the type is needed for types 0-6. For types 7-14 (boxes with two lines) we also need an offset corresponding to the shorter line; the offset is $o = 0$ if the shorter line is a side of the box (the offset o for a block labeled f is shown in **Figure 3**).

Apart from the type τ and offset o, the direction of traversal inside the block is required to identify which region(s) inside the box are inside the polygon. For example, in **Figure 3** while both blocks marked a and b have the same type (positive-slope diagonal) the traversal direction in block a makes the region above the diagonal inside the polygon, while in box b the region below the diagonal is inside the polygon. Similarly, in box f the wedge formed by 2 lines is inside the polygon while in block g the wedge is outside the polygon.

3.3. Transactions for Constructing Block ADS

A transaction InitBlock(B_i, k, k_n, ξ_u, n) adds a k-v pair $\{B_i, v_i\}$ for key B_i (block identifier), by splitting an existing k-v pair for key k with next key k_n. For the first InitBlock() transaction (to add the first block to an empty tree with $\xi_B = 0$), $k = k_n = B_i$.

The value v_i in k-v pair $\{B_i, v_i\}$ is initialized as follows:

$$\begin{aligned} &v_i.\xi_u = \xi_u,\ v_i.n = n \\ &v_i.\xi_v = v_i.\xi_r = v_i.\xi_t = 0 \\ &v_i.A = v_i.p_c = v_i.p_s = v_i.p_b = v_i.p_e = 0 \end{aligned} \tag{15}$$

with all values set to 0—except the population n (specified by the transaction) and the root ξ_u of unverified boundary lines.

For a region R with b blocks, b InitBlock() transactions will need to be executed to create a leaf for every block in the block ADS.

3.3.1. Validating Polygon B_i

The types of transactions executed in the blockchain for validating any block is the same for every block. The order in which transactions are executed will be block specific (depending on which specific lines/boxes should be split).

Transactions for validating block B_i affect only values in v_i (in the k-v pair $\{B_i, v_i\} \in \xi_B$). For simplicity of notation, in this section, we shall simply refer to a root $v_i.\xi_r$ as simply ξ_r.

Leaves in tree ξ_r are rectangular bounding boxes of the form $[c_1, c_2, 0]$, with lower left corner at $c_1 = (x_1, y_1)$ and upper right corner at $c_2 = (x_2, y_2)$.

Leaves in tree $\xi_u\theta$ and ξ_v are boundary lines of the form $[p_1, p_2, v]$.

The tree ξ_t is a key-value collection. A leaf $[p_1, p_2, v] \in \xi_t$, implies existence of keys p_1 and p_2 and absence of keys between p_1 and p_2 (the value v has no meaning).

The 5 transaction types used for validating blocks are as follows:

1) SplitLine(B_i, p_1, p_2, p): Split boundary line $[p_1, p_2, B_i] \in v_i.\xi_u$ at a point p; this transaction is well-formed only if p lies on the line;

2) SplitBB (B_i, c_1, c_2, v, dir): Split BB $[c_1, c_2, 0]$ in $v_i.\xi_r$ vertically at X-coordinate v or horizontally at v (depending on the value dir; 0 for vertical and 1 for horizontal). If the tree is empty, the value v is ignored to create the first leaf $[c_1, c_2, 0]$;

3) MapLine (B_i, p_1, p_2, c_1, c_2): Delete leaf $[p_1, p_2, B_i] \in v_i.\xi_u$ and add it to $v_i.\xi_v$; this transaction is valid only if the line falls entirely inside box $[c_1, c_2, 0] \in v_i.\xi_r$, and p_1 and/or p_2 is a corner of the bounding box; furthermore, if p_1 is found to be redundant, add k-v pair $\{p_1, x\}$ to ξ_t (the value x is irrelevant); update area and centroid;

4) Map2Line ($B_i, p_1, p_2, p_3, c_1, c_2$): Move 2 leaves $[p_1, p_2, B_i]$ and $[p_2, p_3, B_i]$ from tree $v_i.\xi_u\theta$ to tree $v_i.\xi_v$, if both lines fall entirely inside box $[c_1, c_2, 0] \in v_i.\xi_r$, and the common point p_2 is a corner of the box; if p_1 is found to be redundant, add k-v pair $\{p_1, x\}$ to ξ_t; if p_2 is found to be redundant, add k-v pair $\{p_2, x\}$ to ξ_t; update area and centroid;

5) Finalize (B_i) This transaction is invoked to mark completion of verification of the polygon/block B_i.

Only if the points in ξ_u were specified in the CCW order, will the computed area be positive. A transaction Finalize (B_i) is well-formed only if $v_i.p_s = v_i.p_e$

(mapping completed), $A > 0$ (mapping was performed in the CCW order) and $v_i.\xi_u = 0$ (all input lines were correct). The value A (area) is divided by 2 (see Equation (2)); values $p_c.x$ and $p_c.y$ are then divided $6A$ (see Equation (4)).

This process is repeated for every block to complete the construction of the ADS ξ_B.

3.4. Creating and Verifying Proposals

As a redistricting proposal, the proposer submits a claimed metric G, and the commitment ξ_B to the input to the process (the block ADS).

A transaction Proposal (ξ_B, g, m) is invoked, and results in creation of a commitment ξ to 24 values in state-polygon information in s. This initializes 21 values to 0; only 3 values are set to non zero values, using the input to the transaction:

$$\begin{aligned} &s.\xi_B = \xi_B && \text{Completed Block ADS} \\ &s.G = g && \text{Unverified Metric} \\ &s.m_u = m && \text{Unverified Population Mean} \end{aligned} \tag{16}$$

Just as the block ADS with root ξ_B was created to summarize information regarding every block, the process of validating the redistricting proposal results in the creation of

1) a district ADS with root ξ_D,

2) a key value collection ξ_{BD} with block identity as key and district identity as value, and

3) the state ADS (with a single leaf), with root ξ_S.

On completion of the process, the district ADS will contain district wide information like area, centroid, population, number of blocks, moment of inertia, etc., for every district, and the statewide metrics will be available in ξ_S.

Successful execution of all block validation protocols simply demonstrates that every block is a valid polygon. Successful execution of processes for updating ξ_D and ξ_{BD} for every district will confirm that no overlaps exist in any of the blocks in any specific district. This is due to the fact that the district area is computed in 2 ways, by summing up areas of all blocks included in the district, and by "walking" along the boundary of the district (computed using Equation (2)).

However, it still does **not** rule out possible overlaps between blocks in different districts (and consequently, overlaps between districts). The final process which directly manipulates values in the lone leaf [s] in ξ_S involves creation of boundary lines for the entire state to confirm that no overlap exists between districts. Once again, this is achieved by computing the area in 2 ways.

In other words, to prove that $G = G'$ (the correctness of his/her proposal)

1) The proposer will need to execute different types of transactions that create/update the district ADS ξ_D and key value collection ξ_{BD}; this will involve processes for each district, for computing district-wide totals like population, area, moment of inertia, etc.

2) The proposer will then need to execute transactions to verify the boundary lines of the entire state, and compute state-wide totals (total moment of inertia *I*, and the normalized variance v_n, and the final metric G').

Every district that has been considered to create state-wise totals (population, area, etc.) will be added to a key-value collection $s.\xi_d$ to ensure that no district is double counted. It is in this final process that the a unverified mean population $s.m_u$ is used to compute normalized variance $s.v_n$, total moment of inertia, total number of blocks/districts ($s.b_{ct}$ and $s.d_{ct}$) inside the state, and the computed metric $s.G'$ (which should be equal to $s.G$). However, by the end of the process a mean $s.m_v$ is computed and verified to be the same as unverified $s.m_u$. Once the proposal has been verified, all boundary lines of the state will be in a tree with root $s.\xi_v$.

3.5. District ADS Construction

The district ADS with root ξ_D has key-value pairs of the form $\{D_j, u_j\}$). A transaction InitDistrict (D_j, k, k_n, p_{cu}) is invoked to add a k-v pair for district D_j (for the first leaf $k = k_n = D_j$). Only a single value in u_j, viz., $u_j.p_{cu}$, is initialized by the transaction as the (unverified) centroid of the district. All other (13) values are set to 0.

During the validation process, the area of the district will be computed incrementally in 2 different ways in $u_j.A$ (by MapLine and Map2Lines transactions) and $u_j.A_u$, by adding the areas of all blocks added to ξ_{BD}. The unverified centroid in $u_j.p_{cu}$ will be ultimately validated by a computed centroid in $u_j.p_c$. The moment of inertia will be computed for each district using the unverified centroid $u_j.p_{cu}$, and the computed values of block centroids and block areas in the block ADS ξ_B. The value $u_j.b_{ct}$ is the count of unique blocks added to the district.

In the block validation process, a leaf for a block B_i was initialized with the root ξ_u (collection of un-validated block lines). However, a transaction to initialize a leaf for a district D_j sets $\xi_u = 0$.

To populate $u_j.\xi_u$ with outer district lines for D_j, the prover will need to **choose** appropriate lines from block-ADS ξ_B. The prover is allowed to choose **any** validated boundary line $[p_1, p_2, B_i] \in \xi_B.v_i.\xi_v$, for **any** block B_i, and

1) add the line to $u_j.\xi_u$, and

2) add $\{B_i, D_j\}$ to k-v collection ξ_{BD} (only if k-v pair does not already exist).

Any number of lines may be added from the same block B_i. However, only for the first line from a specific block B_i, the k-v pair $\{B_i, D_j\}$ is **added** to ξ_{BD}. For subsequent lines, the existence of $\{B_i, D_j\} \in \xi_{BD}$ is merely confirmed. The fresh addition of a key-pair $\{B_i, D_j\} \in \xi_{BD}$ is accompanied by the following steps

1) $u_j.A_u$ is incremented by the computed area of block B_i, $\xi_B.v_i.A$;

2) $u_j.I$ (moment of inertia) is incremented by using i) the unverified cen-

troid $c_1 = u_j.p_c$ of the district, 2) computed centroid $c_2 = \xi_B.v_i.p_c$ of block B_i and 3) computed area $A = \xi_B.v_i.A$ of block B_i. Specifically, $u_j.I$ is incremented by Ar^2 where r is the distance between the district centroid c_1 and block centroid c_2;

3) $u_j.b_{ct}$ (number of blocks in the district) is incremented by 1.

The prover can also merge 2 lines $[p_1, p_2, v]$ and $[p_2, p_3, v]$ in the tree $u_j.\xi_u$ of yet-to-be-verified boundary lines into $[p_1, p_3, v]$ *if* p_2 is found to be a redundant point.

The purpose of such transactions are 2 fold: i) to form the boundary line of the district D_j in tree $u_j.\xi_u$, and 2) include at least one line from *every block in the district*. The process for validating the district polygon in $u_j.\xi_u$ will be very similar to the process for validating a block. Unless a line is chosen from **every** block inside the district, the values $u_j.A$ (computed during district validation) and $u_j.A_u$ (computed by adding areas of unique blocks to ξ_{BD}) will not match.

Once all necessary border lines have been copied to the tree $u_j.\xi_u$, transactions SplitLine(), SplitBB(), MapLine(), Map2Lines() can be used to ensure that the district lines of D_j in $u_j.\xi_u$ correspond to a simple polygon.

However, after all border lines have been mapped to bounding boxes, some internal lines will still remain in $u_j.\xi_u$, as at least one line had to be added from every internal block in the district. To be able to remove such lines from $u_j.\xi_u$ it is essential to demonstrate that such lines fall *inside* district D_i. To make this possible, the transactions MapLine(), Map2Lines() set the value v of bounding box leaves $[c_1, c_2, v] \in u_j.\xi_r$.

For internal bounding boxes with no lines, the value v will not be set by MapLine()/Map2Lines() transactions. A special transaction has to used to set the box value depending on the value of any adjacent box. For example the value of box c can be determined by examining the value of an adjacent box like d or e. With all boxes marked, in this manner, it is now possible to check if a point p lies inside the polygon (district D_j), by examining a single box in $u_j.\xi_r$ in which point p falls, and examining the value v of the bounding box. This test can be used to throw away remaining lines in $u_j.\xi_u$ that were mapped from internal blocks, but were not used to create the district boundary.

The transactions for validating a district are as follows: A transaction ChooseLine (D_j, p_1, p_2, B_i)

1) chooses a line $[p_1, p_2, B_i] \in \xi_B.v_i.\xi_v$ from the input ADS (a k-v pair $\{B_i, v_i\} \in \xi_B$ indicates the root $v_i.\xi_v$ of a validated boundary tree, in which this leaf $[p_1, p_2, B_i]$ should exist);

2) adds the line to tree $u_j.\xi_u$;

3) checks if $\{B_i, D_j\} \in \xi_{BD}$, and adds the pair if it does not exist;

4) if $\{B_i, D_j\}$ was added by the transaction the following additional steps are performed;

a) increment $u_j.A_u$ using the area of block B_i in the block ADS;

b) increment population $u_j.n$ by population of block B_i in block ADS;

c) increment moment of inertia using unverified district centroid, and computed centroid and area of block B_i in block ADS.

After district lines have been input to $u_j.\xi_u$, they are validated using the following transactions

1) SplitLine (D_j, p_1, p_2, p), for splitting boundary lines in $u_j.\xi_u$.

2) SplitBB (D_j, c_1, c_2, v, dir), for splitting bounding boxes in $u_j.\xi_r$.

3) MapLine (D_j, p_1, p_2, c_1, c_2), for moving a line from $u_j.\xi_u$ to $u_j.\xi_v$ (subject to mapping constraints), **and** set the value v of bounding box in $u_j.\xi_r$.

4) Map2Lines ($D_j, p_1, p_2, p_3, c_1, c_2$), for moving 2 adjacent lines from $u_j.\xi_u$ to $u_j.\xi_v$ (subject to mapping constraints), **and** set the value v of bounding box in $u_j.\xi_r$.

5) InnerBox (c_1, c_2, c_1', c_2', v'), to set the value of an interior box $[c_1, c_2, v = 0]$ using the value v' of an adjacent box $[c_1', c_2', v']$.

6) RemLine (p_1, p_2, c_1, c_2) to remove a line with end points p_1, p_2 from the tree $u_j.\xi_u$ by demonstrating that p_1 falls inside bounding box $[c_1, c_2, v]$, and examining the value *v*.

Finally, a transaction Finalize (D_j) is recognized as well-formed only if $u_j.p_s = u_j.p_e$ (mapping completed), $A > 0$ (mapping was performed in the CCW order) and $u_j.\xi_u = 0$ (all input lines were correct). It resets points $u_j.p_s, u_j.p_b, u_j.p_e$ to zero. The value $u_j.A$ (area) is divided by 2; values $u_j.p_c.x$ and $u_j.p_c.y$ are then divided by $6A$. Another requirement for this transaction to be well-formed is that the computed centroid $u_j.p_c$ should be the same as the unverified value $u_j.p_{cu}$ used for computing the moment of inertia $u_j.I$ of the district.

On completion of the Finalize (D_j) transaction for every district, the metrics like moment of inertia and total population are available for each district. At this stage it is guaranteed that no block can appear in multiple districts as the only blocks in the key-value collection ξ_{BD} can be added to a district, and duplicate keys (block identities) are ruled out in a key-value collection.

As the same block cannot be included in two different districts it follows that districts cannot have overlaps. More specifically, overlaps in districts can happen only if there are overlaps between blocks. However, by confirming that the sum of the areas of all blocks in any district is the same as the area of the district, we can conclude that blocks in the **same** district do not overlap.

To rule out overlaps between blocks in **different** districts we have one more step—to confirm that the area of the state is the sum of areas of all districts. Just as boundaries of each district were constructed using block boundary lines in the block ADS, the boundary lines of the state can be constructed using boundary lines in the district ADS ξ_D.

3.6. Constructing State Boundary Lines

Unlike processes for initialing leaves for each block in ξ_B or each district in

ξ_D, the "tree" ξ_S with a single leaf [s] was already initialized by the transaction Proposal() to create a proposal.

A transaction ChooseLine (D_j, p_1, p_2) can be invoked to add any line from any district D_j (from the verified boundary line tree $\xi_D.u_j.\xi_v$). If no entry exists in the k-v collection $s.\xi_d$ for key D_j, an entry is added. Every time an entry is added in the collection

1) $s.A_u$ is incremented by the area of district D_j ($\xi_D.u_j.A$);

2) $s.I$ is incremented by the moment of inertia of D_j ($\xi_D.u_j.I$);

3) $s.n$ is incremented by $n = \xi_D.u_j.n$, the population of district D_j, and $\Xi.v_n$ is incremented adding $\left(n - m_u\right)^2 / m_u$;

4) $s.d_{ct}$, the number of unique districts added to the k-v collection is incremented by 1;

5) $s.b_{ct}$, the number of unique blocks is incremented by the block-count of the district $\xi_D.u_j.b_{ct}$.

After the required lines are added to $s.\xi_u$ for creating the state boundary (along with other lines needed to ensure that at least one line from each district is added to $s.\xi_u$), the following transactions can be used to map lines in $s.\xi_u$ to create a verified collection of boundary lines in $s.\xi_v$.

1) SplitLine ($p_1, p_2, p \to$), for splitting boundary lines in $s.\xi_u$;

2) SplitBB (c_1, c_2, v, dir), for splitting bounding boxes in $s.\xi_r$;

3) MapLine ($p_1, p_2, c_1, c_2 \supset$), for moving a line from $s.\xi_u$ to $s.\xi_v$ (subject to mapping constraints), and set the value v of bounding box in $s.\xi_r$;

4) Map2Lines (p_1, p_2, p_3, c_1, c_2), for moving 2 adjacent lines from $s.\xi_u$ to $s.\xi_v$ (subject to mapping constraints), and set the value v of bounding box in $s.\xi_r$;

5) InnerBox (c_1, c_2, c_1', c_2', v'), to set the value of an interior box $[c_1, c_2, v = 0]$ in $s.\xi_r$ using the value v' of an adjacent box $[c_1', c_2', v']$;

6) RemLine (p_1, p_2, c_1, c_2) to remove a line with end points p_1, p_2 from the tree $s.\xi_u$ by demonstrating that p_1 falls inside a box $[c_1, c_2, v]$ and examining the value v.

Finally, a transaction ReportMetric() is considered well-formed only if $s.G = s.G'$, and $m_v = m_u$ where

$$m_u = m_v = \Xi.n / \Xi.d_{ct}$$

$$G = G' = w_1 \Xi.I + w_2 \sqrt{v_n} \tag{17}$$

If ReportMetric() is well-formed the commitment to ξ_S (to all values in s) is indicated as the state following the transaction. As s includes commitment to all other ADSes, it is also a commitment to all information regarding and all boundary lines, and all bounding boxes, and all other block/district/state specific information in Table 1.

4. Discussions

The strategy proposed in this paper is a significant improvement over a previous

strategy in [38]. The salient differences between the two stem from a new strategy for point-location queries proposed in [34].

In [38] the strategy for verifying that there are no intersections between lines of a polygon, used the traditional algorithmic approach (sweep-line) [39] with $\mathbb{O}(n \log n)$ complexity for n points. A state-machine model of this process will also call for $\mathbb{O}(n \log n)$ state-changes (or transactions). In this paper, the approach of mapping lines to bounding boxes in [34] was used instead. It calls for linear dependency between number of transactions and number of points. The number of operations will depend on how many redundant points will need to be created, and/or the number of bounding box cuts. Our observations with real-world data from [6] suggest a constant factor between $1.5n$ to $3.2n$.

Secondly, apart from being more efficient compared to [38], the end result of the transactions, *viz.*, creation of "map" like structures ξ_r for every polygon, can be used for purposes other than redistricting, for example, determine in which block/district a particular point (x, y) falls [34].

Finally, this paper addresses a minor flaw in [38]. Validating the district/state area merely by "walking" around the district/state boundary (incremental area computation using Equation (2)), and by adding up all block/district areas, still opens up the possibility of being able to swap a block inside the district with an equal area block outside the district. By constructing bounding box maps and checking at that least one point from every included block actually lies inside the district, this flaw in [38] is addressed in this paper.

To keep the discussion simple we have focused on a single compactness metric and a single population metric. Additions necessary to substitute metrics are trivial.

5. Conclusions

This paper presented a strategy for blockchain based redistricting to enhance public trust. Every step in the construction of useful geographic ADSes—a transaction—can be verified by anyone to be correct, using VOs that accompany every transaction.

Apart from improving public trust in the redistricting process, the authenticated data structures produced by the process can have utility in several other applications. For example, ADSes of different states could be combined after every redistricting to create nationwide maps.

In the description for validating blocks in Section 3.3 there was no need to set the values of bounding boxes to facilitate reliable point-location queries (point-location capabilities were only needed at district and state levels to throw out internal boundary lines from the collection of un-validated lines). However, it may be advantageous to do so, even at the block level to support further subdivisions of blocks. For example, blocks could be further subdivided into zones, parcels, etc., for use by local governments.

One practical limitation of the proposed approach is that it does not take into

account the fact that some districts/states may need to be discontinuous (for example, islands separated by water). This would call for additional transactions to handle such scenarios. Another practical issue not addressed in this paper is the validation of an important nongeographic input—the population of each block. As the process for constructing block ADS should be performed by a relevant government authority, the specific nature of authentication mechanisms is beyond the scope of this paper. Our current efforts include strategies for addressing some practical limitations of the proposed approach. More generally, the approach of merely verifying solutions to NP hard optimization problems in a blockchain network, can be extended to several other application scenarios. Our ongoing research includes other useful application scenarios.

Acknowledgements

Mahalingam Ramkumar was partially funded by the United States Department of Agriculture, Agricultural Research Service (USDA-ARS): 58-0200-0-002, "Advancing Agricultural Research through High Performance Computing".

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Crocker, R. (2012) Congressional Redistricting: An Overview. CRS Report for Congress (R42831).

[2] Altman, M. (1998) Districting Principles and Democratic Representation. Ph.D. Thesis, California Institute of Technology, Pasadena.

[3] Saxon, J. (2020) Reviving Legislative Avenues for Gerrymandering Reform with a Flexible, Automated Tool. *Political Analysis*, **28**, 372-394. https://doi.org/10.1017/pan.2019.45

[4] Cohen-Addad, V., Klein, P.N. and Young, N.E. (2018) Balanced Centroidal Power Diagrams for Redistricting. *Proceedings of the* 26*th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Seattle, 6-9 November 2018, 389-396. https://doi.org/10.1145/3274895.3274979

[5] Altman, M. and McDonald, M.P. (2010) The Promise and Perils of Computers in Redistricting. *Duke Journal of Constitutional Law & Public Policy*, **5**, 69-159.

[6] United States Census Bureau. Topologically Integrated Geographic Encoding and Referencing (TIGER) Database. https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-geodatabase-file.html

[7] ESRI (1998) ESRI Shapefile Technical Description: An ESRI White Paper. https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[8] Rushby, J.M. (1981) Design and Verification of Secure Systems. 8*th ACM Symposium on Operating System Principles*, Pacific Grove, 14-16 December 1981, 12-21. https://doi.org/10.1145/800216.806586

[9] Wei, J. and Pu, C. (2005) TOCTOU Vulnerabilities in UNIX-Style File Systems: An

Anatomical Study. 4*th USENIX Conference on File and Storage Technologies*, San Francisco, 13-16 December 2005, 155-167.

[10] Bright, P. (2018) Meltdown and Spectre: Here's What Intel, Apple, Microsoft, Others Are Doing about It. Ars Technica.

[11] De Lucia, M.J. (2017) A Survey on Security Isolation of Virtualization, Containers, and Unikernels. US Army Research Laboratory, ARL-TR-8029.

[12] Percival, C. (2005) Cache Missing for Fun and Profit. BSDCan. https://www.bsdcan.org/2015/

[13] Lipp, M., Gruss, D., Spreitzer, R., *et al.* (2016) ARMageddon: Cache Attacks on Mobile Devices. 25*th USENIX Security Symposium*, Austin, 10-12 August 2016, 549-564.

[14] Saini, H., Rao, Y.S. and Panda, T.C. (2012) Cyber-Crimes and Their Impacts: A Review. *International Journal of Engineering Research and Applications*, **2**, 202-209.

[15] Larochelle, D. and Evans, D. (2001) Statically Detecting Likely Buffer Overflow Vulnerabilities. 10*th USENIX Security Symposium*, Washington DC, 13-17 August 2001, 177-190.

[16] Patten, D. (2017) The Evolution to Fileless Malware. http://www.infosecwriters.com/Papers/DPatten_Fileless.pdf

[17] Stewin, P. and Bystrov, I. (2012) Understanding DMA Malware. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, c, d, 21-41. https://doi.org/10.1007/978-3-642-37300-8_2

[18] Samyde, D., Skorobogatov, S., Anderson, R. and Quisquater, J.J. (2002) On a New Way to Read Data from Memory. *First International IEEE Proceedings of Security in Storage Workshop*, 11 December 2002, 65-69.

[19] Ramkumar, M. (2018) Scalable Computing in a Blockchain. 2018 *IEEE* 39*th Sarnoff Symposium*, Newark, NJ, 24-25 September 2018, 1-6. https://doi.org/10.1109/SARNOF.2018.8720499

[20] Dotan, M., Pignolet, Y.A., Schmid, S., *et al.* (2021) Survey on Blockchain Networking: Context, State-of-the-Art, Challenges. *ACM Computing Surveys*, **54**, Article No. 107. https://doi.org/10.1145/3453161

[21] Ramkumar, M. (2018) Executing Large Scale Processes in a Blockchain. *Journal of Capital Market Studies*, **2**, 106-120. https://doi.org/10.1108/JCMS-05-2018-0020

[22] Hendrix, E.M.T. and Toth, B.G. (2010) Goodness of Optimization Algorithms. In: *Introduction to Nonlinear and Global Optimization*, Springer, New York, 67-90. https://doi.org/10.1007/978-0-387-88670-1_4

[23] Braden, B. (1986) The Surveyor's Area Formula. *The College Mathematics Journal*, **17**, 326-337. https://doi.org/10.1080/07468342.1986.11972974

[24] Bourke, P. (1988) Calculating the Area and Centroid of a Polygon. http://paulbourke.net/geometry/polygonmesh/

[25] Reock Jr., E.C. (1961) A Note: Measuring Compactness as a Requirement of Legislative Apportionment. *Midwest Journal of Political Science*, **5**, 70-74. https://doi.org/10.2307/2109043

[26] Polsby, D. and Popper, R. (1991) The Third Criterion: Compactness as a Procedural Safeguard against Partisan Gerrymandering. *Yale Law & Policy Review*, **9**, 301-353. https://doi.org/10.2139/ssrn.2936284

[27] Young, H.P. (1988) Measuring the Compactness of Legislative Districts. *Legislative Studies Quarterly*, **13**, 105-115. https://doi.org/10.2307/439947

[28] Anagnostopoulos, A., Goodrich, M.T. and Tamassia, R. (2001) Persistent Authenti-

cated Dictionaries and Their Applications. *Proceedings of the 4th International Conference on Information Security*, Malaga, 1-3 October 2001, 379-393. https://doi.org/10.1007/3-540-45439-X_26

[29] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A. and Stubblebine, S. (2001) A General Model for Authentic Data Publication. VC Davis Department of Computer Science Technical Report.

[30] Ramkumar, M. (2014) Symmetric Cryptographic Protocols. Springer, Berlin. https://doi.org/10.1007/978-3-319-07584-6

[31] Adhikari, N., Bushra, N. and Ramkumar, M. (2019) Complete Merkle Hash Trees for Large Dynamic Spatial Data. 2019 *International Conference on Computational Science and Computational Intelligence* (*CSCI*'19), Las Vegas, 5-7 December 2019, 1318-1323. https://doi.org/10.1109/CSCI49370.2019.00246

[32] Chelladurai, U. and Pandian, S. (2021) HARE: A New Hash-Based Authenticated Reliable and Efficient Modified Merkle Tree Data Structure to Ensure Integrity of Data in the Healthcare Systems. *Journal of Ambient Intelligence and Humanized Computing*, 1-15. https://doi.org/10.1007/s12652-021-03085-0

[33] Merkle, R.C. (1987) A Digital Signature Based on a Conventional Encryption Function. In: Pomerance, C., Ed., *Advances in Cryptology—CRYPTO*'87. *CRYPTO* 1987. *Lecture Notes in Computer Science*, Vol. 293. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48184-2_32

[34] Adhikari, N. (2020) Authoritative and Unbiased Responses to Geographic Queries. Ph.D. Thesis, Mississippi State University, Starkville.

[35] Xiao, Y., Zhang, N., Lou, W. and Hou, Y.T. (2020) A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Communications Surveys & Tutorials*, **22**, 1432-1465. https://doi.org/10.1109/COMST.2020.2969706

[36] Nakamoto, S. (2008) A Peer-to-Peer Electronic Cash System. Bitcoin.org.

[37] Buterin, V. (2015) A Next-Generation Smart Contract and Decentralized Application Platform. Ethereum White-Paper, 36 p.

[38] Adhikari, N., Bushra, N. and Ramkumar, M. (2019) Redistricting Using Blockchain Network. 2019 *First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications* (*TPS-ISA*), Los Angeles, 12-14 December 2019, 150-159. https://doi.org/10.1109/TPS-ISA48467.2019.00026

[39] Shamos, M.I. and Hoey, D. (1976) Geometric Intersection Problems. 17*th Annual Symposium on Foundations of Computer Science* (*SFCS* 1976), Houston, 25-27 October 1976, 208-215. https://doi.org/10.1109/SFCS.1976.16

Abbreviations

TCB	Trusted Computing Base	ADS	Authenticated Data Structure
VN	Von Neumann	CCW	Counter-clockwise
VO	Verification Objects	BB	Bounding Box