

# Continuous Cartogram Construction

Donald H. House<sup>‡</sup>  
Visualization Laboratory

Christopher J. Kocmoud<sup>†</sup>  
Texas Center for Applied Technology

Texas A&M University

## Abstract

Area cartograms are used for visualizing geographically distributed data by attaching measurements to regions of a map and scaling the regions such that their areas are proportional to the measured quantities. A continuous area cartogram is a cartogram that is constructed without changing the underlying map topology. We present a new algorithm for the construction of continuous area cartograms that was developed by viewing their construction as a constrained optimization problem. The algorithm uses a relaxation method that exploits hierarchical resolution, constrained dynamics, and a scheme that alternates goals of achieving correct region areas and adjusting region shapes. It is compared favorably to existing methods in its ability to preserve region shape recognition cues, while still achieving high accuracy.

**Keywords:** cartogram, value-by-area map, map transformation, anamorphosis, thematic cartography, constrained optimization.

## 1 INTRODUCTION

A common technique for the visualization of geographically distributed data is to attach representations of the data to a map of the area of study. Misperceptions can arise, however, when geographic area and quantitative data are not correlated, because visualizing data on a traditional map inherently emphasizes regions with the greatest geographic area, while making the presentation of very small regions quite difficult. A common technique for overcoming these problems is to use an area cartogram, in which the areas of map regions are rescaled to conform to the data.

Area cartograms are often used for visualization of the geographic distribution of "routine" data in a variety of disciplines, including political science, social demographics, epidemiology and business. For example, the results of the popular vote in the contiguous 48 states in the 1996 U.S. presidential race are visualized in Color Plate 1a using traditional thematic mapping. Each state is colored a shade of red or blue denoting the majority winner as Clinton or Dole, respectively, with the color saturation indicating the magnitude of the winning percentage. This visualization fails to answer the important question, "Who won the national election?" Without prior knowledge of the populations of each of the states, and thus their electoral votes, the viewer has no clear indication of the winner. This map, in fact, produces an intrinsic distortion of the data. It could easily mislead one to reach exactly the wrong conclusion, that Dole was the winner, since his states cover a higher percentage of the total geographical area. Since elections are not won on area, but on population, a much better representation would be the visualization of Color Plate 1b. This map is an area cartogram that shows the same data,

but with the states rescaled using the algorithm presented in this paper. Through the spatial transformation of states relative to the data, the cartogram emphasizes data distribution rather than geographic area, and makes it readily apparent that Clinton won.

The map in Color Plate 1b is an example of a popular form of the area cartogram, the continuous area cartogram. Here the map regions are not merely scaled proportional to the data, but the map topology is maintained – common region boundaries remain attached to each other, and the spatial ordering of regions with respect to each other is consistent with the original map. Although continuous cartograms have visualization potential, they are not used very often in practice, since they are difficult to construct effectively. Therefore, having a robust algorithm for automatically constructing continuous area cartograms would be a great aid to geographers.

A good continuous area cartogram solution maintains the topology of accurately deformed areas while also preserving recognizable regional boundaries so that the map can still be read correctly. This paper examines previous algorithmic attempts at the automatic construction of cartograms and presents a new approach based on reformulating the cartogram construction task as a constrained optimization problem.

## 2 EXISTING METHODS

Several computer algorithms have already been developed to construct continuous area cartograms [1, 2, 3, 5, 9, 11, 12, 13]. We present detailed background on many of the most effective of these methods and make comparisons of their results with ours elsewhere [7]. Here we confine ourselves to three distinctly different methods.

Tobler developed a *Pseudo-Cartogram Method* that creates an equal density approximation by compressing or expanding lines of latitude and longitude until a least root mean square error solution is obtained [12]. This method has proven to be an effective way to "preprocess" a map prior to cartogram construction, but is rarely used alone since the resulting cartograms can contain extensive area error. Dorling takes a *Cellular Automaton* approach where a grid is superimposed on the map, and individual grid cells are swapped until every geographic region has a number of cells corresponding to its desired area [1]. While this method is very effective at achieving area, regions tend to lose their unique contours and acquire a shape reflecting the grid.

A number of other methods are radial in nature [2, 5, 9], using a rubber-sheet paradigm. These methods apply radial transformations to the map, giving a result that is highly accurate, but also prone to "ballooning" distortions and the transformation of straight lines into curves. Gusein-Zade and Tikunov's *Line Integral Method* is the most sophisticated of these methods. It repeatedly applies radial transformations such that the density of a selected cell is made uniform while leaving all other cells unchanged, with the vector sum of transformations applied as a line integral around each of the region boundaries [5].

<sup>‡</sup> house@viz.tamu.edu, Texas A&M University, 216 Langford, College Station, TX 77843-3137. <http://www-viz.tamu.edu/>

<sup>†</sup> c-kocmoud@tamu.edu, Texas Engineering Experiment Station, 214 WERC, College Station, TX 77843-3407. <http://teesweb.tamu.edu/>

### 3 THE CONTINUOUS AREA CARTOGRAM CONSTRUCTION PROBLEM

In contrast to earlier methods, we propose that the problem of automatically constructing a continuous area cartogram can be thought of as a constrained optimization problem posed in the following way. We assume that a map is represented as a connected collection of polygonal regions, and that there exists an area assignment function that apportions total map area among these regions. Normally, this function will assign area proportional to some geographically distributed quantity that is to be visualized. We wish to rescale map regions in such a way that we maximize "recognizability" of the regions of the rescaled map, subject to the constraints that 1) the surface area of each rescaled region is held at its assigned value, and 2) that map topology is maintained (i.e. connectivity and spatial ordering of regions must be identical to those in the original map). The search space then becomes the space of all maps that conform to the area and topological constraints imposed by the area assignment function and the original map. Within this space, we wish to select the map in which regions are maximally recognizable.

The issue of "recognizability" is fraught with difficulties, as it is at best a vaguely defined cognitive term and not a measurable quantity. Lacking experimental data to back up our assumptions, and based on our personal experience with the problem, we decided to measure recognizability using two factors. The first of these is conformance of the orientation of region edges to their orientation in the original map. This tends to prevent rotation and shearing of regions. The second is that the lengths of the edges of each individual region should maintain the same proportions to each other that they had in the original map.

The optimization problem, thus posed, can be seen to be of very high dimension. Also, it is a problem to which we must generally expect that there is no obvious or trivial solution. The constraints and objective function are likely to be quite adversarial. For example, there is nothing preventing regions that share a common edge from being scaled in the opposite direction, thus making it impossible to maintain exact edge length proportionality in both regions. Likewise, the constraints themselves can easily conflict, since arbitrarily scaling regions is quite likely to break map topology as regions attempt to grow over other region boundaries, and any naive algorithm will be likely to attempt to turn portions of regions "inside out", with edges crossing over each other, in an attempt to create "negative areas" that cancel excessive size in other parts of the region.

Due to the high dimensionality of the problem and its difficulty, we elected to model our algorithm after the simulated annealing approach [6]. Although this approach is known to be quite slow, it is especially well suited to this type of problem. We did not follow simulated annealing in any precise sense, but instead exploited its key features. In simulated annealing there is the notion of temperature – large moves are made in the search space when the temperature is high, and the search is more local when temperature is low. Temperature is decreased in stages as the optimization proceeds. In our algorithm, the notion of map coarseness substitutes for temperature, and the map becomes more refined (i.e. less coarse) in stages as the algorithm proceeds. Likewise, in simulated annealing, a kind of jitter is added to the search process by having a non-zero probability of accepting new search space configurations that lower the objective function value. In our algorithm, we achieve a similar result by alternating objective functions between orientation and edge length and by

alternating between an area achieving process and a shape maintaining process.

In recognition of the difficulty of the problem, we also relax the area constraint somewhat in that we are willing to examine configurations that only approximate the area constraint. To be precise, one could rewrite the area constraint to be one in which the area must be maintained within some tolerance. Topological constraints, however, are strictly maintained in our algorithm.

### 4 THE ALGORITHM

In designing the algorithm, we divide the process of creating continuous cartograms into two distinct but conflicting tasks: adjusting region sizes and retaining region shapes. The algorithm, as shown in Figure 1, iterates over a map sampled at consecutively higher levels of detail [7]. Within a level of detail, it achieves desired areas without regard to shape and then restores shape while attempting to hold the areas fixed. These tasks are performed within the modeling paradigm of a constraint-based physical system.

```
Regions := LoadFullResolutionMap (MapDataFile);
coarseness := max_coarseness;
AreaTargets := CalculateDesiredArea (Regions);
repeat
  SimplifyMap (Regions, coarseness);
  repeat
    AchieveAreas (Regions, AreaTargets);
    RestoreShapeWhileMaintainingArea (Regions);
  until adverse effect of area upon shape increases;
  ReconstructMapToFullResolution (Regions, coarseness);
  coarseness := coarseness / 2;
until coarseness < min_coarseness;
```

Figure 1: Pseudocode of our continuous cartogram algorithm.

The map vertices and region data are loaded or obtained via GIS database linkages or other sources. The data structure holding the map organizes the vertices into consistent clockwise order for each polygonal region. The map is initially acted upon at a coarse resolution and refined later to progressively higher levels of detail. Thus, gross configuration changes in the system occur during initial phases and finer details are adjusted in later phases.

Our "relaxation process" alternates between the two goals of resizing regions to their correct areas and then restoring region shapes while attempting to hold their areas fixed, switching goals when the solution "stagnates." The resulting "jitter" within the system seems to provide ample randomization to bounce the map out of local optima.

The sample graph in Figure 2 shows the cycling effect between achieving area and restoring shape. The alternating relaxation process continues until the resulting map from the area-adjusting routine contains more shape error than the preceding area-adjusted map, signaling stagnation at this resolution. In our implementation we progress to the next level of refinement after the second such increase in shape error. In Figure 2, the first and second refinement lines mark shifts to higher resolution. When this occurs, the map is reconstructed to its full resolution, resampled at twice the previous level of detail, and run through the

alternating relaxation process again. The refinement process continues until the solution stagnates at the desired resolution.

The example depicted in Figure 3 shows stages of the run graphed in Figure 2, beginning with the initial map in Figure 3a and the first coarsely sampled map in Figure 3b. The deformation resulting from the first attempt to achieve region areas is shown in Figure 3c, and Figure 3d shows the subsequent result of holding the areas fixed while attempting to restore shape.

#### 4.1 Hierarchical Resolution

The first step in the simplification of map resolution is the identification of certain shared vertices that would cause a break in the map topology if they were simplified away. These “key vertices” are all interior vertices shared by three or more regions and all perimeter vertices shared by two or more regions. The key vertices of a western U.S. map are highlighted in Figure 4a. Simply connecting the key vertices, as demonstrated in Figure 4b, is impractical due to the gross shape error introduced. Instead, we simplify between key points at a resolution relative to the size of the region, as shown in Figure 4c.

A minimal number of simplified edges are constructed between two key points such that the distance between any vertex and its simplified edge is within an allowed offset distance, computed as a percentage of the length of the region’s bounding box diagonal. In this manner the simplification is based upon a collection of resolutions custom-scaled to each region, providing a balanced simplification of region details that is independent of region size.

Consider the California example depicted in Figure 4d, where the coarseness level is set to allow a maximum offset of 6% of the diagonal length. The first pair of key vertices bordered the coastline and Mexican border, and a simplified line connecting them was tested first. Since at least one vertex was beyond the allowed distance from the line, a simplified line connecting the next vertex down the coastline was tested. This process continued down the entire coast, as shown in Figure 4e, until the edge merely consisted of adjacent vertices spanning the Mexican border. Two more simplified edges were required in order to complete the coastline simplification, as shown in Figure 4f. Continuing in this way, the original forty-seven edges were reduced down to seven. With each simplified edge, we store offset information to allow reconstruction (i.e. “unsimplification”) of its vertices after it is scaled and rotated by the resizing process. The minimum distance between a line  $P$  and a vertex  $R$  will always be along a path perpendicular to the line. As demonstrated in Figure 4g, a parallel vector  $\mathbf{u}$  and a perpendicular vector  $\mathbf{v}$  are calculated for every vertex with respect to the origin of the simplified edge  $\mathbf{e}$ . The proportional offset values to be used later for reconstructing the edge relative to its new scale and orientation are

$$U = \|\mathbf{u}\|/\|\mathbf{e}\| \text{ and } V = \|\mathbf{v}\|/\|\mathbf{e}\|. \quad (1)$$

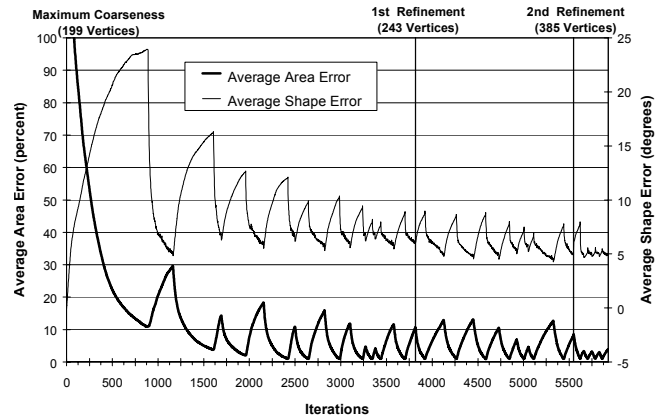


Figure 2: Example algorithm area and orientation error for the 1996 U.S. population cartogram.

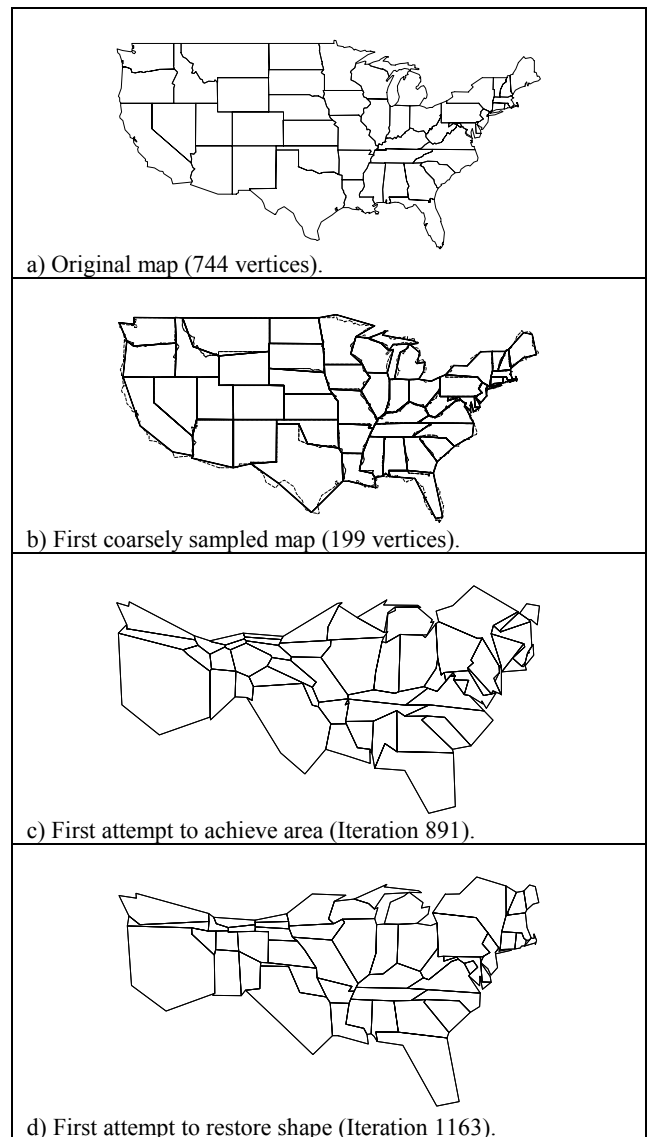


Figure 3: Example stages of the U.S. cartogram.

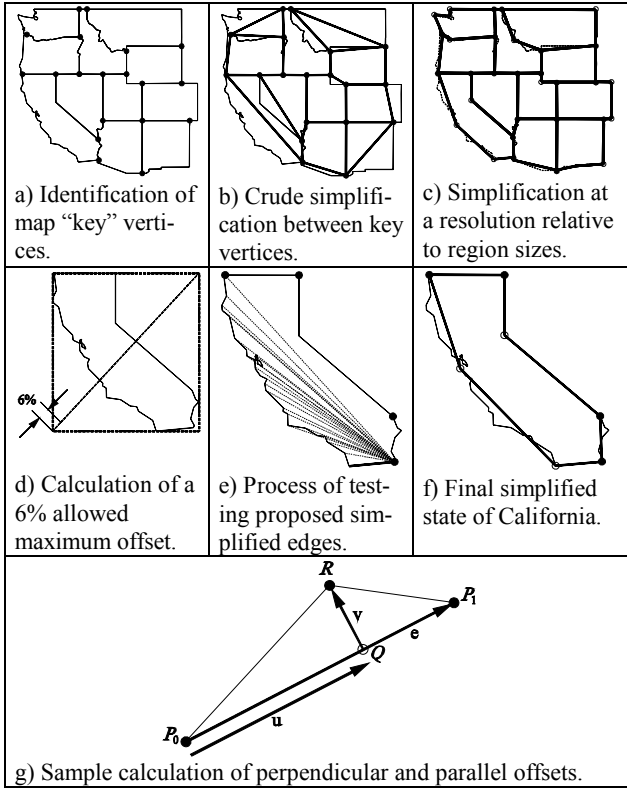


Figure 4: Identification of map key vertices and example simplifications of the western U.S. and California.

## 4.2 Dynamics

Our method uses a dynamic system paradigm, in which area and shape maintaining forces act upon the map vertices. Within this paradigm we frequently apply strong, one-time forces upon vertices to prevent a break in map topology. In a momentum-based Newtonian physical system this could lead to oscillations and possible instability. Instead, we base our method on Aristotelian dynamics, where the *velocity* of a point is directly proportional to the total force upon it. Therefore, a vertex only moves when a force is acting upon it, regardless of its velocity at a previous point in time. The resulting motion is similar to that of a very heavily damped Newtonian physical system.

## 4.3 Achieving Desired Areas

Region areas are achieved using *area springs*, linear physically-based elements that exert force outward along their length when compressed and inward when stretched. Within a single region, area springs exert equal forces upon each vertex in a direction that bisects each interior angle. For an isolated region, this would result in simply scaling the region. However, as shown in Figure 5, adjacent regions exert area forces upon shared vertices, resulting in a tug of war, with the region of greatest error having the stronger hand.

The area of a region with  $n$  vertices  $(x_i, y_i)$  is given in [4] as

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

where the operator  $\oplus$  is addition modulo  $n$  and  $i \in [0 \dots n-1]$ . The desired area  $A_{\text{desired}}$  of each region is made proportional to its share of the geographic quantity being visualized, scaled so that the final map will have the same total area as the original map.

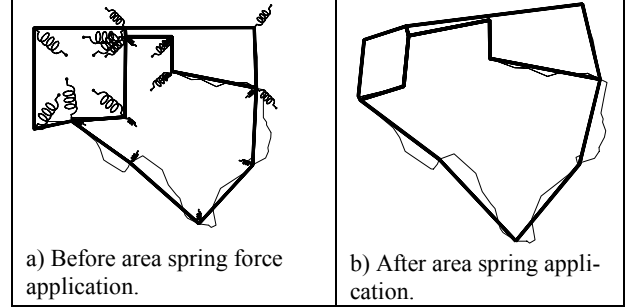


Figure 5: Examples of area springs for a simplified Texas, New Mexico, and Oklahoma map.

The force projected by the area springs is proportional to the percentage area error of a region,

$$\epsilon_{\text{area}} = 100(A - A_{\text{desired}}) / A_{\text{desired}}. \quad (3)$$

The area spring force to be applied to each region vertex is

$$\mathbf{F}_{\text{AS}} = \frac{K_{\text{AS}} \epsilon_{\text{area}}}{N_{\text{vertices}}} \mathbf{u}, \quad (4)$$

where  $K_{\text{AS}}$  is a user-specified scaling parameter,  $N_{\text{vertices}}$  is the number of region vertices, and  $\mathbf{u}$  is the direction bisecting the angle formed by the vertex and its adjacent edges.

The pseudocode for resizing regions is given in Figure 6. It begins with the distribution of area spring forces, as necessary, to the region vertices. This is followed by superimposing topological constraint forces that prevent regions from inverting and intersecting. These are described in detail in a later section. The net forces are applied to the individual vertices in the dynamics procedure, thereby affecting vertex velocities and consequently their positions.

```

procedure AchieveAreas (Regions, AreaTargets);
  repeat
    DistributeAreaSpringForces (Regions, AreaTargets);
    DistributeTopologicalConstraintForces (Regions);
    PointDynamics (Regions);
  until average area error increases;

```

Figure 6: Pseudocode of the process to resize regions to their desired areas.

## 4.4 Restoring Region Shapes

Shape restoration is achieved by two force-exerting elements: *orientation springs* and *edge length proportionality springs*. The goal of orientation springs is to influence region edges to return to their initial map orientation. When an edge has deviated, equal and opposite restoring forces are applied perpendicularly to its endpoints.

The initial orientation of a simplified edge (which substitutes for any number of original edges) can be calculated as a unit vector in the direction of the line connecting the initial locations of its

two endpoints. For edge  $k$ , the angular difference between the current orientation direction  $\mathbf{d}_k$  and the initial orientation  $\mathbf{d}_{0k}$  is

$$\varepsilon_k = \cos^{-1} \left( \frac{\mathbf{d}_k \cdot \mathbf{d}_{0k}}{\|\mathbf{d}_k\| \|\mathbf{d}_{0k}\|} \right). \quad (5)$$

The orientation spring force on edge  $k$  is given by

$$\mathbf{F}_{OS} = K_{OS} \varepsilon_k \mathbf{u}_\perp, \quad (6)$$

where  $K_{OS}$  is a user specified magnitude and  $\mathbf{u}_\perp$  is a direction vector perpendicular to edge  $k$ . The direction of positive force for each vertex is determined from the cross-product of the initial and current orientations. Thus, the orientation spring forces tend to induce a rotation of the edge back to its initial orientation, as shown in Figure 7a.

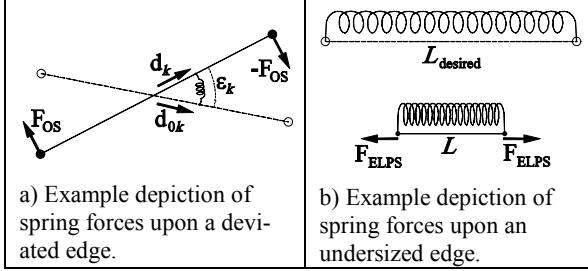


Figure 7: Orientation springs and edge length proportionality springs.

The purpose of edge length proportionality springs is to influence region boundary edges to remain in the same proportion to each other as they were in the original map. This spring exhibits force parallel to the edge to adjust it nearer to the desired proportional length. The fraction of the original perimeter taken by edge  $k$  is

$$f_k = \frac{\|\mathbf{e}_{0k}\|}{\sum_{i=0}^{n-1} \|\mathbf{e}_{0i}\|}, \quad (7)$$

where  $n$  is the number of region edges, so that the desired length of edge  $k$  is always

$$L_{desired} = f_k \sum_{i=0}^{n-1} \|\mathbf{e}_i\|, \quad (8)$$

Thus, the current percentage of error is

$$\delta_k = 100(1 - \|\mathbf{e}_k\|/L_{desired}). \quad (9)$$

The force is given by

$$\mathbf{F}_{ELPS} = K_{ELPS} \delta_k \mathbf{u}_\parallel, \quad (10)$$

where  $K_{ELPS}$  is a user-specified scaling parameter and  $\mathbf{u}_\parallel$  is a unit direction vector parallel to the edge. The force is projected equally in opposite directions upon the two endpoints, as shown in Figure 7b, to scale the edge towards its desired length.

The shape restoration pseudocode, given in Figure 8, begins with the distribution of shape forces, either from the orientation springs or the edge length proportionality springs. We have found that isolating the two shape mechanisms, whose forces are often contradicting, enables each to retain its ground more effectively against adversarial area constraint forces. The looping construct halts when the average shape error, computed as a weighted sum of degrees of orientation error and unit lengths of edge proportion error, ceases to decrease.

```

procedure RestoreShapeWhileMaintainingArea (Regions);
  useOS := true;
  repeat
    if useOS then
      DistributeOrientationSpringForces (Regions);
    else
      DistributeEdgeLengthProportionalitySpringForces
        (Regions);
    useOS := not useOS;
    DistributeAreaConstraintForces (Regions);
    DistributeTopologicalConstraintForces (Regions);
    PointDynamics (Regions);
  until average shape error increases;

```

Figure 8: Pseudocode of the shape restoration process.

## 4.5 Maintaining Area

The key component of the shape restoration process, and our entire method, is holding the region *areas* fixed while their *shapes* are readjusting. This is done through applying area constraint forces, which attempt to cancel those components of the shape forces that would cause a change in area. The resulting constrained dynamic environment enables shape adjustments to occur without significant loss of accuracy in area.

### 4.5.1 Constrained Dynamics

We utilize constrained particle dynamics, as detailed in tutorial form by Witkin in [14], to ensure that our regions obey a specific area constraint. We make the obvious modifications to Witkin's method to utilize Aristotelian dynamics.

We describe our physical system through a set of differential equations involving the current system state and applied forces upon it. If  $\mathbf{x}_i$  is a vector of the positions of the vertices in region  $k$ , we see from equation (2) that requiring the region to maintain its desired area  $A_k$  is equivalent to maintaining the constraint equation

$$C_k(\mathbf{x}_k) = \frac{1}{2} \sum_{j=0}^{n-1} (y_j + y_{j \oplus 1})(x_{j \oplus 1} - x_j) - A_k = 0, \quad (11)$$

where  $j$  is an index of the  $n$  vertices in region  $k$ . Rather than process each region separately, we define a state vector  $\mathbf{x}$  of all vertex positions, its time derivative  $\dot{\mathbf{x}}$  which is a vector of vertex velocities, and a vector  $C(\mathbf{x})$  whose elements are scalar constraints of the form given by equation (11). Aristotelian dynamics functions relate point velocity directly to the total force  $\mathbf{F}_T$  on the system by the global equation

$$\mathbf{F}_T = \dot{\mathbf{x}}, \quad (12)$$

with the assumption that each map vertex has unit mass. Total force consists of all applied forces  $\mathbf{F}_A$ , and a set of constraint forces  $\mathbf{F}_C$  that guarantee that the applied forces do not violate our constraints. Thus,

$$\mathbf{F}_T = \mathbf{F}_A + \mathbf{F}_C. \quad (13)$$

Witkin's analysis [14] shows that the constraint forces  $\mathbf{F}_C$  are scalar multiples of the columns of the jacobian matrix  $\mathbf{J}$ , where the components of  $\mathbf{J}$  are found by taking the partial derivatives of equation (11),

$$\frac{\partial C_i(\mathbf{x})}{\partial \mathbf{x}_j} = \frac{1}{2} [\dot{y}_{i,j \oplus 1} - \dot{y}_{i,j \ominus 1}, \dot{x}_{i,j \oplus 1} - \dot{x}_{i,j \ominus 1}]. \quad (14)$$

Here the region index  $i$  is in  $[0 \dots m-1]$ , the edge index  $j$  within the region is in  $[0 \dots n-1]$  and the operators  $\oplus$  and  $\ominus$  are modulo  $n$ . This occurs when the constraint forces are parallel to the gradient of their associated constraint function. This can be expressed as

$$\mathbf{F}_C = \mathbf{J}^T \boldsymbol{\lambda}, \quad (15)$$

where  $\boldsymbol{\lambda}$  is an unknown vector of Lagrange multipliers.  $\boldsymbol{\lambda}$  is determined by solving the constrained dynamics equation

$$\mathbf{J}\mathbf{J}^T \boldsymbol{\lambda} = -\mathbf{J}\mathbf{F}_A \quad (16)$$

and substituting the result into equation (15), yielding the necessary constraint forces.

#### 4.5.2 Implementing Dynamic Area Constraints

As given in the pseudocode in Figure 8, the area constraining forces  $\mathbf{F}_C$  are computed *following* the distribution of the applied shape restoration forces  $\mathbf{F}_A$ . Thus, the constraints serve as a mediator, canceling shape forces that would tend to change region areas.

$\mathbf{J}$  is an  $m \times n$  matrix where  $m$  is the number of area constraints (i.e. the number of regions) and  $n$  is the total number of vertices. This matrix can be quite large for maps with hundreds of vertices and many regions. However, each of our constraints typically influences only a handful of vertices, so that most of the partial derivatives in equation (14) are zero. Therefore, we implement  $\mathbf{J}$  as a *sparse matrix*, as detailed in [14], with  $m$  indexed lists of the non-zero entries in each row. Equations (15) and (16) require the implementation of two operations upon the sparse matrix: *matrix times vector* and *matrix-transpose times vector*. Both operations are straightforward, performing normal matrix multiplication using the indexed offsets within each list.

To solve equation (16) for  $\boldsymbol{\lambda}$  we use the bi-conjugate gradient method [8]. Note that when the system is overconstrained there may be no exact solution, in which case the bi-conjugate gradient method gives the solution with the least mean squared error.

Having found  $\mathbf{J}$  and  $\boldsymbol{\lambda}$ , we solve equation (15) for our constraint forces and apply them to the vertices. The only other forces applied during the shape restoration process are from topological constraints to ensure the integrity of the map topology. Since the topological constraints *follow* the dynamic area constraints, some of our area maintaining forces can be canceled by the more important goal of preserving map topology. The combination of topological constraints, overconstraining and roundoff error can result in region areas changing somewhat during the shape restoration process. However, in practice we have found this drift to be small and easily taken care of by our iterative relaxation process, which periodically adjusts area.

#### 4.6 Maintaining Topological Integrity

During the relaxation and map reconstruction processes, there is a high probability that a region will attempt to invert, intersect itself, or intersect another region. Breaks in map topology are prevented by utilizing topological constraint forces. Region inversions are prevented by applying hinge and edge constraints. *Hinge constraints* restrict the angle between two adjacent edges from going below 0 degrees or above 360 degrees, and *edge constraints* prevent edges from collapsing to zero length and possibly inverting to negative length. Both constraints are implemented as

non-linear springs that reach a high maximum force opposing constraint violation just before it is violated.

*Intersection penalties* exert equal and opposite forces, as shown in the Figure 9 examples, to correct situations where a region has overlapped itself or others. We detect intersections using the parametric line clipping method [4]. Region self-intersection is detected by testing for intersection between each edge and all other edges in the region. Interregional overlaps are identified by testing for intersection between only the edges in the set of map boundary edges with every other map boundary edge.

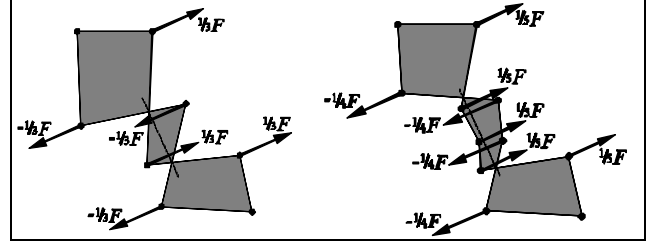


Figure 9: Examples of intersection penalty forces.

We define the magnitude of the penalty force as

$$F_p = K_p \tau \|\dot{\mathbf{x}}_{avg}\|, \quad (17)$$

where  $K_p$  is a user-defined constant,  $\tau$  is the length of time this intersection has been active, and  $\dot{\mathbf{x}}_{avg}$  is the average velocity of the vertices at the moment of the initial intersection. The velocity term applies more countering force for intersections with fast moving edges. The force is distributed equally to both sides of the intersection and then apportioned equally upon each side's vertices.

## 5 Results

In Figure 10 we show 1980 U.S. population cartograms generated by the *Pseudo-Cartogram* and *Line Integral* methods alongside an original map that serves as a reference. We do not show comparisons with the *Cellular Automaton* method since it makes no attempt to preserve shape. Results of our cartogram algorithm and those of the other algorithms are presented here at approximately the same scale to provide fair comparisons. While the shapes are preserved well in the *Pseudo-Cartogram* in Figure 10b, the 60% area error contained in this approximation is made quite obvious when viewed next to the other cartograms. The *Line Integral* cartogram in Figure 10c displays the pinching and “ballooning” typical of radial algorithm cartograms, but approaches near 1% area error. By contrast, our method successfully preserves the distinctive shapes of nearly all the states, as well as straight lines, and still achieves a high accuracy of 1.5% average area error.

In Figure 11 we demonstrate our method applied to the presentation of a 1981 equal population cartogram of British counties. Our cartogram algorithm is again successful in maintaining region shape and recognition. The average area error of 28% in our cartogram is mainly due to two extremely sparsely populated counties, with 196% error each, that cannot shrink any further without compromising our specified level of shape preservation. This is a case where there appears to be no true solution to the optimization problem as posed. Nevertheless, in cases like this our algorithm gives results that may very well be satisfactory for many applications.

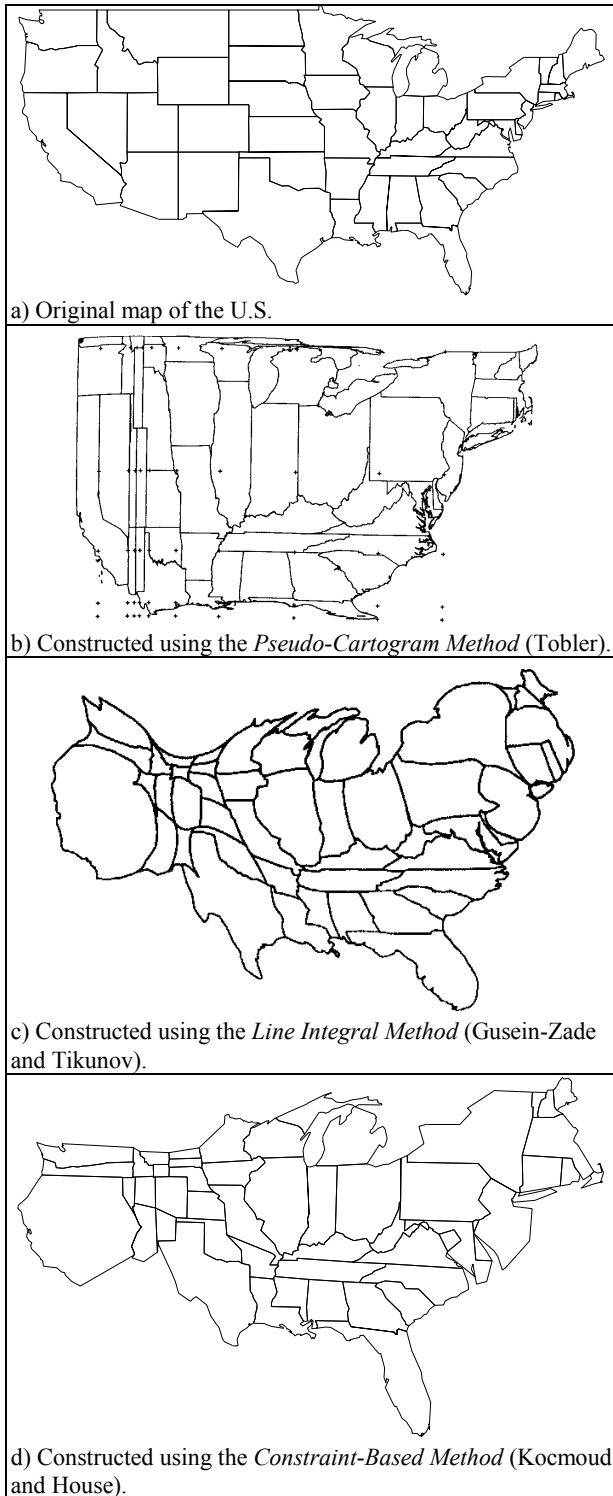


Figure 10: 1980 U.S. population cartograms (b: Reproduced with permission from [12], page 49, figure 8, © 1986 American Congress on Surveying and Mapping; c: Reproduced with permission from [5], page 172, Figure 1, © 1993 American Congress on Surveying and Mapping).

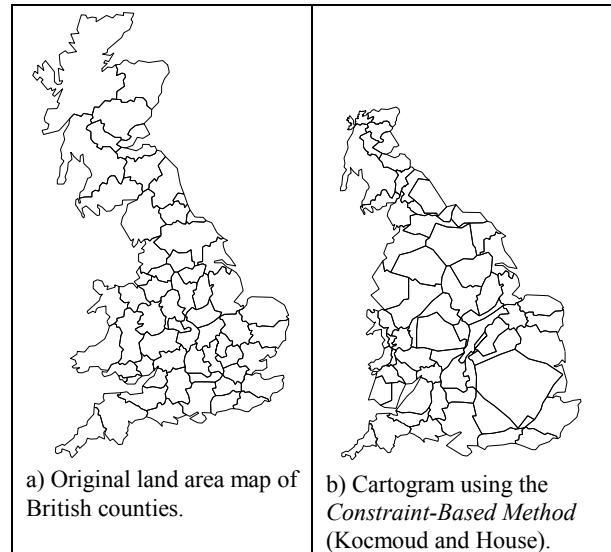


Figure 11: 1981 equal population cartogram of Britain.

## 6 Discussion

Our method was implemented in the MicroStation Development Language, a standard C programming language that compiles and executes within the MicroStation CAD program (Bentley Systems, Inc.). The map input data can be loaded from a formatted text file or converted directly from MicroStation CAD or GIS data. Our examples were created on a 300 MHz Compaq Professional Workstation with 128MB RAM. Transforming the U.S. map with 744 vertices took 6 hours for the 1980 cartogram. The 1981 population cartogram of British counties contains 983 vertices and ran for about 9 hours.

The algorithm's space complexity is easily analyzed. Let  $R$  be the number of map regions,  $E$  be the number of edges, and  $N$  the number of vertices in the fully detailed map. The map is stored as a list of regions, a list of edges and a list of vertices. Each region maintains a list of edge indices, and each edge is described by a pair of vertex indices. Each region must have at least three edges but an edge participates in at most two regions. Each vertex participates in two or more edges. Clearly  $E \geq N > R$ . The only data structure used by the algorithm that might be of more than linear complexity is the jacobian product matrix  $\mathbf{J}\mathbf{J}^T$  of equation (15). However, the jacobian  $\mathbf{J}$  is stored as a sparse matrix of size  $O(N)$ , and the product matrix need never be actually computed and stored, since the bi-conjugate gradient solution method does not use this matrix directly (see [14]). Since the algorithm must store the full map, and none of the data structures used by the algorithm is of more than linear complexity, the size complexity of the algorithm is simply  $O(E)$ .

The algorithm's computational complexity is more difficult to determine and, in fact, will require a careful experimental study. However, some progress can be made analytically. First note that the algorithm operates over a hierarchy of maps of coarser and coarser resolution. We will call the coarseness level  $c$ , where  $c = 0$  implies that the map is at full resolution. We assume that the number of edges remaining in the map at resolution level  $c$  will be  $E_c = E\alpha^c$ . In other words, at each coarsening of the map, a fixed fraction  $1/\alpha$  of the remaining edges are removed. Our method does not do exactly this, but this is roughly its effect, of course recognizing that a map cannot be coarsened beyond limits im-

posed by map topology. For the analysis, we will also make the assumption that the actual number of edges in a region does not deviate significantly from the average number, i.e. that any process that is  $O(e)$ , where  $e$  is the number of edges in a single region, will be roughly  $O(E)$  when iterated over all the regions in a map.

Given these assumptions, at coarseness level  $c$  each individual step given by the pseudocode in Figures 1, 6 and 8 can be shown to be  $O(E_c)$ , except for *DistributeAreaConstraintForces* and the *DistributeTopologicalConstraintForces*. Under our assumptions, *DistributeAreaConstraintForces*, using sparsity techniques and solving equation (15) via the bi-conjugate gradient approach, takes time  $O(E.R)$ . *DistributeTopologicalConstraintForces* does an exhaustive edge-edge crossing test within each region, giving a total time of  $RO(e_c^2) = O(E_c^2/R)$ . *DistributeTopologicalConstraintForces* also does a test for every edge on the perimeter of the map against every other perimeter edge, but for a large map this number of tests will be small compared with the total number of tests within regions. Thus, over a reasonable set of assumptions, each substep in the algorithm has complexity greater than  $O(E_c)$ , but less than  $O(E_c^2)$ . Since the number of edges in the map is reduced exponentially by increasing coarseness, the hierarchical resolution approach is lent theoretical credence by this analysis. Preliminary experiments over a very limited number of cases appear to bear this out, but further study must be done to give reliable performance numbers.

Trickier to determine is the computational complexity of the complete algorithm, since there are several nested loops whose number of iterations cannot be reasonably estimated analytically. These no doubt have to do with complexity factors other than problem size and are connected to some notion of problem difficulty. One helpful piece of preliminary experimental information is that the amount of time spent by the algorithm at each coarseness level appears to be about the same. For example, in a run similar to the one shown in Figure 2, the amount of time spent in the first 3,800 iterations at maximum coarseness was nearly identical to the amount of time spent in the next 1,800 iterations after one refinement and also to the amount of time spent in the final 400 iterations after the second refinement. Thus, while the number of edges in the map increases geometrically as coarseness decreases, the number of iterations required decreases enough to compensate. This leads to the inviting conclusion that the time complexity of the algorithm might be  $C$  times a subpolynomial function of  $E_c$  or  $E_c^C$ , where  $C$  is the maximum coarseness level, and  $E_c$  is the number of edges in the map at maximum coarseness.

A significant feature of our method is the ability to incorporate interactive aesthetic control, enabling the user to fix small problems at any time by stiffening particular springs or by modifying or pinning down vertex locations. The user can also adjust the level of area accuracy with respect to sacrifice in shape at any time. Since we wished to stress the automatic nature of our algorithm, we did not hand-adjust any of the examples shown here. However, these features could easily be integrated into an attractive interface.

We are investigating the creation of animations over time where the previous cartogram solution is used to initialize the subsequent cartogram. The timings should be very fast, since each cartogram deviates only slightly from its predecessor. The geographic data, as opposed to solution map vertices, could be interpolated between samples (i.e. between censuses) to produce intermediate frames.

## 7 Conclusion

We have considered the problem of generating continuous area cartograms as a constrained optimization problem, and demonstrated a method well suited to automatically solving this problem. The approach appears to be a significant improvement over previous cartogram methods. The technique offers easy map reproducibility as well as the opportunity for interactive aesthetic control. The algorithm runs at speeds that show promise for making it a practical everyday tool for geographic visualization.

## References

- [1] D. Dorling. *Area Cartograms: Their Use and Creation*. Department of Geography, University of Newcastle upon Tyne, Newcastle upon Tyne, England, August 1995.
- [2] J.A. Dougenik, N.R. Chrisman, and D.R. Niemeyer. An Algorithm to Construct Continuous Area Cartograms. *The Professional Geographer*, 37 (1): 75-81, 1985.
- [3] H. Edelsbrunner and R. Waupotitsch. A Combinatorial Approach to Cartograms. *Computational Geometry*, 7: 343-360, 1997.
- [4] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.
- [5] S.M. Gusein-Zade and V. Tikunov. A New Technique for Constructing Continuous Cartograms. *Cartography and Geographic Information Systems*, 20 (3): 167-173, 1993.
- [6] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220 (4598): 671-680, 1983.
- [7] Kocmoud, C.J. *Constructing Continuous Cartograms: A Constraint-Based Approach*. Masters Thesis, Visualization Laboratory, Texas A&M University, College Station, Texas, 1997. <http://www-viz.tamu.edu/students/chris/>
- [8] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. New York, Cambridge University Press, 1992.
- [9] S. Selvin, D. Merrill, S. Sacks, L. Wong, L. Bedell, and J. Schulman. *Transformations of Maps to Investigate Clusters of Disease*. Lawrence Berkeley Laboratory, University of California, No. LBL-18550, 1984.
- [10] V.T. Tikunov and S. Gusein-Zade. Map Transformations. *Geography Review*, 9 (1): 19-24, 1995.
- [11] W.R. Tobler. A Continuous Transformation Useful for Districting. *Annals of the New York Academy of Sciences*, 219 (9): 215-220, 1973.
- [12] W.R. Tobler. Pseudo-Cartograms. *The American Cartographer*, 13 (1):43-50, 1986.
- [13] J.S. Torguson. *Cartogram: A Microcomputer Program for the Interactive Construction of Value-By-Area Cartograms*. Masters Thesis, Department of Geography, University of Georgia, Athens, Georgia, 1990.
- [14] A. Witkin. Constrained Dynamics. *ACM SIGGRAPH 97 Course Notes* (Course 34: Physically Based Modeling), 1995, pp. F1-F12.