

CHAPTER IV

THE ALGORITHM

4.1 Overview

The objective of this chapter is to give a fundamental overview of the basic operational components of our algorithm. Three distinct operations within the algorithm are sufficiently complex to warrant their own chapters: hierarchical resolution, dynamic area constraints, and topological constraints. In addition, there are many other implementation details that are covered in Chapter VIII.

The process of creating continuous cartograms can be broken down into two distinct, and conflicting, tasks: adjusting region sizes and retaining region shapes. Our new algorithm shown in Figure 4.1 solves this geographical density equalization problem by achieving desired areas without regard to shape, and then restoring shape while attempting to hold the areas fixed. These tasks are performed within the modeling paradigm of a constraint-based physical system.

```

LoadFullResolutionMap (Vertices, Regions);
repeat
  SimplifyMap (Regions, coarseness);
  CalculateDesiredArea (Regions);
  repeat
    AchieveAreas (Regions);
    RestoreShapesWhileMaintainingAreas (Regions);
  until adverse effect of area upon shape increases;
  ReconstructMapToFullResolution (Regions, coarseness);
  coarseness := coarseness / 2;
until reached desired level of detail;
```

Figure 4.1: Pseudocode of our continuous cartogram algorithm.

Initially, the map vertices and region data are loaded from a formatted text file or converted directly from CAD data. To ensure consistent mathematical integrity of the algorithm, the traversal order of each loaded region's vertices is reversed as necessary to guarantee a clockwise ordering of vertices within each region.

A typical map dataset could easily contain several thousand lines and vertices. The processing time can often become prohibitive when every minute map detail is referenced and acted upon individually. In addition, detailed bay areas along coastlines can cause noticeable shape artifacts during resizing. This motivated a hierarchical refinement construct within the algorithm whereby the map is initially acted upon at a coarse resolution and refined later at progressively higher levels of detail. Our implementation of progressive hierarchical resolution is modeled after the simulated annealing approach [13], where gross changes in the system occur during the initial high temperatures and fine details are adjusted later at the lower temperatures.

At the heart of our method is a repetitive “relaxation process” where only one goal is actively sought while the other goal is relaxed. Once the current goal achieves a steady state, it is relaxed and the alternate goal is actively sought. In early experiments we found that if area and shape are attempted to be achieved simultaneously the system can easily stabilize in a *local* optimum, falling short of a more *global* optimum that contains less error and better shape preservation. While it is not possible to determine the global optimum for any given system, this alternating process should tend to steadily progress towards better local optima. It appears in our results that this is the case.

Therefore, we utilize the relaxation technique to alternate between the two goals of resizing regions to their correct areas and then restoring region shapes while attempting to hold the areas fixed. The resulting “jitter” within the system due to the alternation focus provides ample randomization to bounce the map out of local optima and towards a more global optimum. This relaxation process is also modeled after the simulated annealing approach [13] where higher energy states are probabilistically accepted, making outlets from local optima possible at any non-zero temperature.

It is desirable to continue alternating between the area and shape objectives as long as the resulting negative effect of one upon the other steadily decreases. In our algorithm we test only the area adjustment’s adverse impact upon shape error, halting the relaxation process when the resulting map from the area-adjusting routine contains more shape error than the preceding area-adjusted map.

Upon completion of the relaxation procedure, the map is reconstructed to its full resolution, resampled at half the current level of coarseness, and run through the relaxation process at this higher level of map detail. The entire process of simplification, reconstruction, and resampling with higher detail continues until an acceptable solution at the desired resolution is attained.

Using hierarchical resolution proves to be very efficient and effective. Instead of moving an entire detailed border outward to expand a region's area, for example, our method moves a simplified border consisting of only a few edges. This not only reduces computation time but automatically preserves shape details within each simplified edge.

4.2 Dynamics

In the algorithm we use a dynamic system paradigm in which area and shape maintaining forces act upon the map vertices. Within this paradigm, we frequently apply a strong, one-time force upon a vertex to prevent a break in map topology.

In traditional Newtonian dynamics, force impacts a point's acceleration, which alters the point's current velocity and, ultimately, changes the point's position. In our application, the resulting momentum together with the use of springs would 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. At a given moment in Aristotelian time, 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 Area

The first operation during the relaxation process is adjusting each region's size relative to its proportion of the data being visualized. One efficient approach would be to concentrate on those few key points whose movement would produce the greatest change in area with the least amount of energy. However, this causes a severe distortion

of region shape and, since only a few edges would be disoriented, the subsequent shape restoration forces would be concentrated upon those few edges to restore their initial orientations. This would cancel most of the progress just made towards achieving area. It is more effective to apply forces equally around the region such that the forces attempt to *scale* the region, thus preserving shape.

The mechanism that performs this scaling function is the area spring. This physically-based element exerts force outward along its length when compressed and inward when stretched, as demonstrated in Figure 4.2a. Area springs are situated so that they scale a region by exerting equal forces upon each vertex, as depicted in Figure 4.2b. However, as demonstrated in Figure 4.2c, adjacent regions are also exerting area forces upon shared vertices, resulting in a tug of war, with the region of greatest error having the stronger hand.

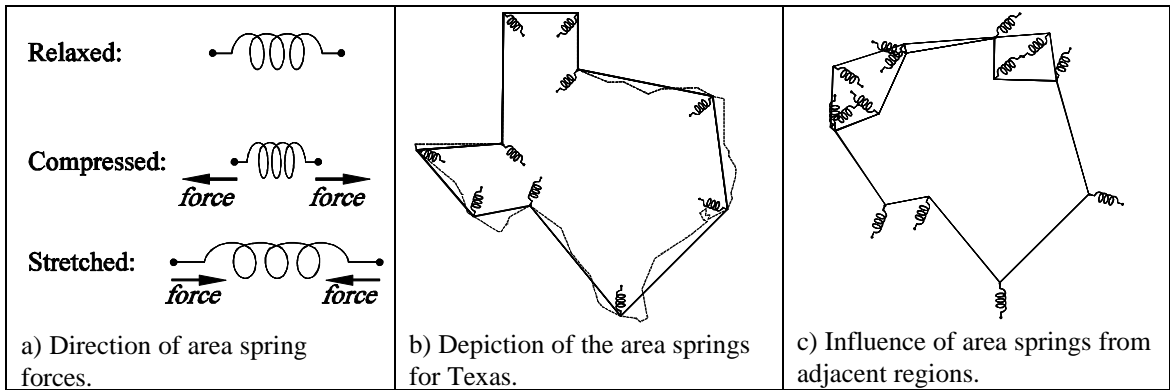


Figure 4.2: Examples of area springs.

The area of a region with n vertices (x_i, y_i) is given in [8] as

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (y_i + y_{i \oplus 1})(x_{i \oplus 1} - x_i), \quad (4.1)$$

where the operator \oplus is addition modulo n . The desired area of a region can be found proportional to the total sum of the map area,

$$A_{\text{desired}} = \frac{\text{data}}{\sum \text{data}} \sum \text{area}, \quad (4.2)$$

where *data* is the data variable we desire to visualize.

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

$$\varepsilon_{\text{area}} = 100(A - A_{\text{desired}})/A_{\text{desired}}, \quad (4.3)$$

which is the percentage difference between the current region area A and the target area A_{desired} . Note that this equation inherently favors oversized regions since the range of values is $(-100\%, +\infty\%)$. This results in greatly oversized regions shrinking first, followed by an even mixture of change as the enlarged regions descend within 100% area error.

The spring force at a vertex is exerted in a direction that bisects the interior angle formed by the two edges adjacent to the vertex. By finding unit direction vectors \mathbf{d}_1 and \mathbf{d}_2 outward from the vertex to the endpoints of each edge, we can compute the force direction as

$$\mathbf{F}_{\text{dir}} = \begin{cases} (\mathbf{d}_1 + \mathbf{d}_2)/\|\mathbf{d}_1 + \mathbf{d}_2\| & \theta < 180^\circ \\ -(\mathbf{d}_1 + \mathbf{d}_2)/\|\mathbf{d}_1 + \mathbf{d}_2\| & \theta > 180^\circ \\ R(90^\circ)\mathbf{d}_2 & \theta = 180^\circ \end{cases} \quad (4.4)$$

where θ is the angle between the two edges. Examples of each case are depicted in Figure 4.3. The positive error of oversized regions will exert a force inward whereas the negative error of undersized regions will project outward.

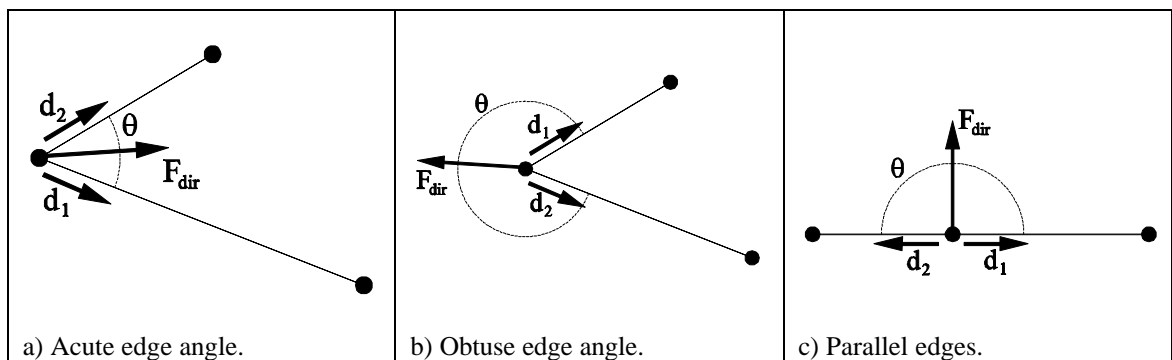


Figure 4.3: Determining the force direction for an area spring.

Having found ϵ_{area} and \mathbf{F}_{dir} , we compute the area spring force to be applied upon each vertex in the region as

$$\mathbf{F}_{\text{AS}} = \frac{K_{\text{AS}} \epsilon_{\text{area}}}{N_{\text{vertices}}} \mathbf{F}_{\text{dir}}, \quad (4.5)$$

where K_{AS} is a user-specified scaling parameter and N_{vertices} is the number of region vertices.

The pseudocode in Figure 4.4 shows the steps in resizing regions to their desired areas. Our looping construct begins with the distribution of the area spring forces, as required, to the region vertices. This is followed by several topological constraints that prevent regions from inverting and self-intersecting. The spring and constraint forces are then collectively applied upon the individual points during the dynamics procedure. Being an Aristotelian dynamical system, these forces affect point velocities which, in turn, modify point positions.

```

procedure AchieveRegionAreas (Regions);

  repeat
    DistributeAreaSpringForces (Regions);
    DistributeTopologicalConstraintForces (Regions);
    PointDynamics (Regions);
  until average area error increases;

```

Figure 4.4: Pseudocode of the process to resize regions to their desired areas.

Every iteration of this loop brings each region closer to its desired proportional area. We continue this loop until the average area error ceases to decrease. The first completion of the area adjustment for the U.S. 1990 population is shown in Figure 4.5a. This contortion rightfully should not be called a map of the United States. Likewise, it also should not be called a cartogram since it completed with 16% average area error still

remaining. We desire a process to transform these same disordered lines into something more recognizable, such as the map in Figure 4.5b.

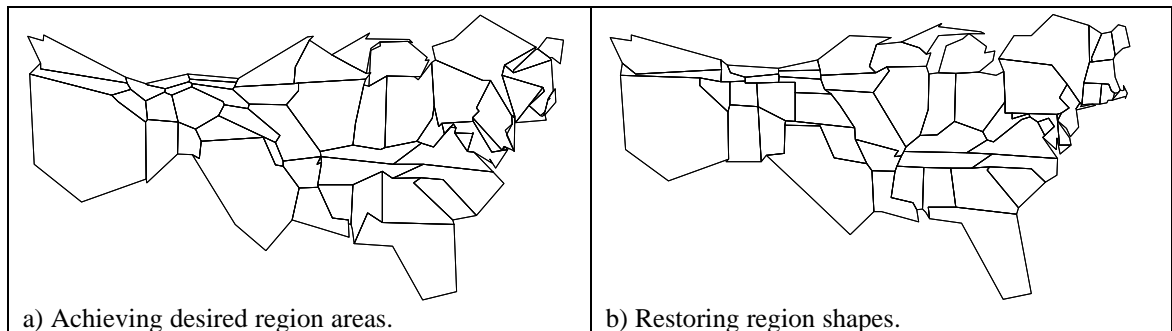


Figure 4.5: Examples of the two-part relaxation process.

4.4 Restoring Shape

Our premise is that even if region areas are correct, a cartogram will not be an effective device for communicating geographic information if the viewer cannot recognize the regions. One major difference between the two maps in Figure 4.5 is that in Figure 4.5b the lines are much closer to their original orientation. Colorado is rectangular once again, Tennessee has recaptured its parallelogram likeness, and the hunchback of New York has disappeared. Another noticeable difference is that the edge lengths of each state are more in proper proportion to each other. The panhandle's conquest over northern Texas has been suppressed and Long Island once again looks like it actually is attached to New York. Shape preservation is performed in both of these ways by two force-exerting elements: orientation springs and edge length proportionality springs.

4.4.1 Orientation Springs

The goal of orientation springs is to influence the region edges to return to their initial orientation. Determining the initial angle is a simple, but not so obvious, task. The initial orientation of a simplified edge should *not* be calculated based upon the point locations at the time of its creation. This is due to the fact that in using hierarchical resolution we can have several layers of simplification and reconstruction of our map,

each of which invariably contains some amount of unresolvable shape error. As such, the initial orientation of a simplified edge must be based upon the *initial* point locations of the *initial* map. Therefore, part of the map loading process includes the storing of initial orientations, or angles, of all edges. This information is used to cause each edge to tend to retain its orientation.

A simplified edge, by definition, is substituted in place of any number of original edges. The desired orientation for the simplified edge can be thought of as the vector average of the initial orientations of the edges contained within it. However, the orientation is actually calculated as a unit vector in the direction of the line connecting the initial locations of the two endpoints.

We attach an orientation spring to each simplified edge. When an edge has deviated from its initial orientation, equal and opposite restoring forces are applied perpendicularly on its endpoints as shown in Figure 4.6a. The angle of deviation of an edge,

$$\varepsilon = \cos^{-1}\left(\frac{\mathbf{d} \cdot \mathbf{d}_0}{\|\mathbf{d}\| \|\mathbf{d}_0\|}\right), \quad (4.6)$$

is the angular difference between the current orientation direction \mathbf{d} and the initial orientation \mathbf{d}_0 . It is desirable to have the force behave smoothly¹ as shown in Figure 4.6b, with a gradual application of force and a user-specified error threshold $\varepsilon_{\text{threshold}}$ beyond which the spring exerts a maximum amount of force. This is realized through the equation

$$\mathbf{F}_{\text{OS}} = \begin{cases} \frac{1}{2} K_{\text{OS}} \left[1 + \cos\left(\left(1 + \frac{\varepsilon}{\varepsilon_{\text{threshold}}}\right)\pi\right) \right] \mathbf{F}_{\text{dir}} & \varepsilon < \varepsilon_{\text{threshold}} \\ K_{\text{OS}} \mathbf{F}_{\text{dir}} & \varepsilon \geq \varepsilon_{\text{threshold}} \end{cases}, \quad (4.7)$$

where K_{OS} is a user-specified magnitude and \mathbf{F}_{dir} is a direction vector perpendicular to the edge. The spring force is projected perpendicularly upon the endpoints to induce a

¹ This was applicable during a previous implementation when the orientation springs and edge springs exhibited force simultaneously. A simple linearly increasing force would suffice here, such as the graph in Figure 4-7b, now that only one of the shape forces is active during any given iteration.

rotation of the edge back to its initial orientation. The reaction time and sensitivity of the spring can be controlled through the user-specified parameters. Maintaining edge orientations in this manner (especially the strong vertical and horizontal ones) greatly improves the discernibility of each region.

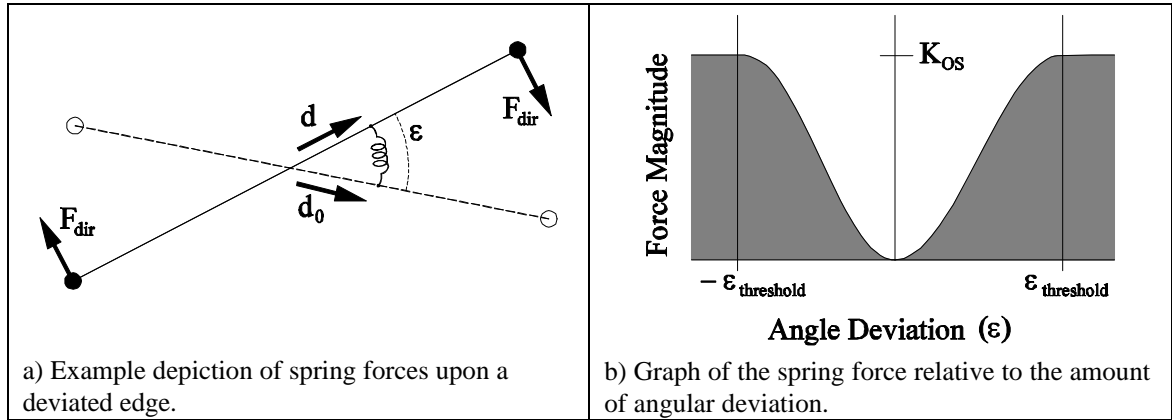


Figure 4.6: Orientation springs.

4.4.2 Edge Length Proportionality Springs

The goal of edge length proportionality springs is to influence the length of region edges to remain in the same proportion to each other with respect to the perimeter as they were in the initial perimeter. This is accomplished by attaching a spring on each edge, exhibiting force parallel the edge to adjust it nearer to the desired proportional length.

In a rationale similar to the orientation springs, the calculation of the desired proportional length of a simplified edge must be based upon the *initial* lengths of the *initial* region perimeter. Therefore, another set of information stored during the map loading process is the percentage length of every edge with respect to the region's initial perimeter. This percentage length of an edge e_k is computed as

$$P_k = \frac{\|e_k\|}{\sum_{i=1}^n \|e_i\|}, \quad (4.8)$$

where n is the number of edges in the region. Using this percentage we compute the amount of error for edge k (e_k) as

$$\varepsilon_k = 100 \left(1 - \frac{\|\mathbf{e}_k\|}{P_k \sum_{i=1}^n \|\mathbf{e}_i\|} \right), \quad (4.9)$$

which is the percentage difference between the current length and the desired proportional length. In the example in Figure 4.7a, we have an edge whose length L is shorter than the desired proportional length $L_{desired}$. The spring forces, therefore, are directed outward to lengthen the edge.

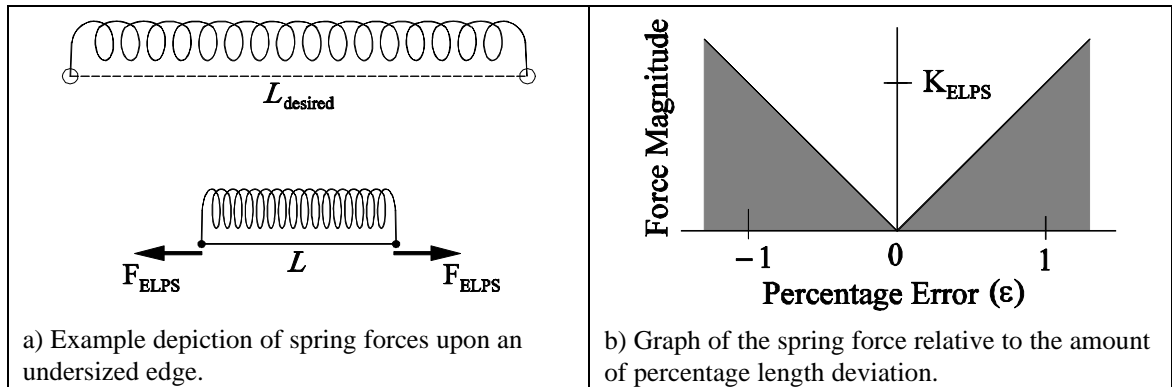


Figure 4.7: Edge length proportionality springs.

The magnitude of the spring force, as graphed in Figure 4.7b, can be given by the following simple linear equation:

$$\mathbf{F}_{ELPS} = K_{ELPS} \varepsilon_k \mathbf{F}_{dir}, \quad (4.10)$$

where K_{ELPS} is a user-specified scaling parameter and \mathbf{F}_{dir} is a unit direction vector parallel to the edge. This force is projected equally in opposite directions upon the two endpoints, resulting in an even scaling of the edge to its desired proportional size.

4.4.3. Shape Restoration Pseudocode

The pseudocode in Figure 4.8 shows the steps in restoring region shapes. Our looping construct 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 their

ground more effectively against the subsequent adversarial procedure, the area constraint.

The key component of the shape restoration process and, coincidentally, the foundational paradigm upon which this entire method is based, lies in the constraints. Without area constraints, this method would simply seesaw between an area adjusted map that sacrifices shape and a shape adjusted map that disregards area. This method, however, utilizes an area constraint to attempt to cancel those components of the shape forces that would cause changes in area. This enables shape adjustments to occur, without significant loss of accuracy in area. The subsequent topological constraints and point dynamics procedures in the pseudocode correspond directly to those described previously in the area adjusting pseudocode.

Every iteration of this loop reduces the angular and length disparity of each edge, thus bringing each region closer to its original shape. Loop termination is based on the average shape error, which can be computed as degrees of orientation error, unit lengths of edge proportion error, or a weighted mixture of both. For our implementation, we monitored only the orientation error, halting the shape loop when the average angular error ceases to decrease.

```

procedure RestoreRegionShapes (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 4.8: Pseudocode of the shape restoration process.