Rendering Dorling Cartograms on Yelp Data

Weijia Jin  
University of California, Berkeley  
Electrical Engineering and Computer Sciences  
jinweijia@berkeley.edu

Jialiang Zhang  
University of California, Berkeley  
Electrical Engineering and Computer Sciences  
jialiang@berkeley.edu

ABSTRACT
Our research is to identify important constraints and develop an algorithm for Dorling Cartograms.

Author Keywords
Dorling Cartograms; Yelp Data

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Cartogram is an abstracted and distorted type of map encoding quantitative information (1). Typical mapping data are population distribution, Gross domestic product (GDP), presidential election distribution, etc. Ever since the modern cartogram technique was well defined in the mid-1800s, area and distance cartogram became the two main popular types of cartograms (2). On the other hand, Bortins and Demers classify cartograms into three main categories: Non-contiguous, Contiguous, and Dorling Cartograms (3).

Dorling Cartogram, sometimes called Circular Cartogram or Hayes-Buday Information System, is one type of area cartograms. It is firstly introduced by Dorling in his book Area Cartograms: Their Use and Creation (4). An example of such Dorling Cartogram is provided in Figure 1. The major differences between Dorling Cartogram and other area cartograms are the fact that each enumeration elements are visualized as circles rather than its original geometric polygons. Thus, it further extracts the useful information from the visualization, filtering out excessive geographic data that might interfere with reader’s perception. Example of Dorling Cartograms application includes population distribution by Inoue (5), and epidemics patterns by Tao (6) etc. For these examples, the radius for each circle is encoded with the population or the disease counts – the data of interest. And the layout of circles is based on predefined geographical constraints, majorly linking between geographic neighbors and overlapping avoidance.

Motivations

INPUT FILES
After Dorling Cartogram is invented, other than the methodology of drawing by hand and creating by mechanical model (4), there are so few computer automated implementations in the community. Meanwhile, most of the existing implementations are adapted from Daniel Dorling’s algorithm and pseudo code from 1996, which will be discussed in related work below. The limitation of these implementations, however, is the specialized input file called shapefile (7).

Dedicated for geographic information system (GIS), shapefile (.shp) is a geospatial data format developed by ESRI (7). Since the shapefile can only be produced by specific geographic software like ArcView GIS and ARC/INFO, Dorling’s algorithm narrows its application in data visualization.

COMPUTATION COEFFICIENTS
In Dorling’s implementation, there are a number of arbitrary coefficients used to optimize the rendering results. However, they are only good for individual cases like mapping the Great Britain counties. When we transformed the algorithm onto the Yelp data, these coefficients break the algorithm due to different data scales.

FEATURE REQUIREMENTS
On the other hand, Dorling’s algorithm only accepts geographic data which has well-defined boundaries (geometric polygons) in order to calculate shared boundary length, region perimeter and list of neighbors which are all

![Figure 1: Dorling and Dorling-like Cartograms Example (by the courtesy of Cartogram Central [3])](image)
used in the algorithm. To make an algorithm to adapt more
general data, some other features need to be identified to
substitute requirements like shared boundary length in the
computation.

So these three aspects are part of the new algorithm
development and will be discussed in detail in this paper.
Specifically, in the Related Work section, we will discuss
about the evolution from Dorling’s original algorithm to
various existing online implementations. In Methods
section, we will present the detailed explanation of our
design techniques and the algorithm. The final Dorling
Cartogram rendering result will be demonstrated in Results
section, as well as evaluations to the implementations. We
will then conclude on the insights and limitations of our
algorithm and implementation, as well as future work that
can be extended on the implementation later.

RELATED WORK

Based on the paper from Dougenik et al. entitled An
Algorithm to Construct Continuous Area Cartograms (8),
Dorling developed his circular cartogram algorithm in
Pascal (9) and following in C (4). The pseudo code of his
algorithm is in Figure 2.

Later on, various implementations in Python emerged
(Johnson (10) and (11), Boyandin (12), Aisch (13), Stork
(14) and Welsh (15)). But most of them are just built on
Dorling’s algorithm in C, with minor modifications.

When the HTML technology booms, the implementations
based on JavaScript and JavaScript libraries emerged.
Based on Protovis (16), which is a JavaScript library
integrated with SVG, Bostock and Heer’s implementation of
Dorling Cartogram is the first one created based on web
technology (17). Later on, when D3 (Data-Driven
Documents) becomes popular (18), Bostock developed the
Dorling Cartogram in D3 (19).

Web technologies, specifically JavaScript and HTML, are
good for rendering the Dorling Cartogram results. So we
decided to start with the development based on JavaScript
and its libraries, and gradually improve our algorithm. The
research path is illustrated as:

JavaScript Protovis/D3 → Dorling’s source code in C →
Python translation and improvement → our final algorithm

METHODS

The Data

We will be rendering our Dorling Cartogram algorithm on
the Yelp Challenge Dataset. This dataset contains
geographic attributes and includes many interesting
information on businesses in 5 cities around the world:
Phoenix, Las Vegas, Madison, Waterloo and Edinburgh
(20). We used the business portion of the data, which is a
large json file composed of a single object type, and one
json-object (data on a single business) per-line.

---

For each region

Calculate the radius of a circle so that its area is proportional to encoding value

While the forces calculated below are not negligible

For each region (the order of calculation has no effect)

For each region which overlaps with the region

Record a force away from the overlap in proportion to it

For each region which originally neighbored the region

Record a force towards it proportional to distance away

If the forces of repulsion are greater than attraction

Scale the forces to less than the distance of the closest circle

Combine the two aggregate forces for each circle

For each region

Apply the forces recorded to be acting on each circle to its centroid

Note:
1. The radius of each region is equal to the square root of its population divided by 7,
then scaled by the total distance between neighboring regions
2. The forces of attraction are also scaled by the length of the original border between the
two regions as a proportion of the total perimeter of the region
3. The ratio to combine by is 60:40 for repulsion:attraction; but given no overlaps,
the total force is scaled to less than the distance to the closest circle
4. The forces acting on each circle are equal to a quarter of the sum of the focuses acting
on the circle at the last iteration and the newly calculated forces

Figure 2: Pseudo Code of Dorling’s Algorithm
We used Python with the pandas library to parse and aggregate the data into the desired format. The traditional cartogram examples are usually built on countries, states or counties as a single unit (circle), however in our case it will be too broad to use city or county as a basic unit as there will be too few circles and we will not get interesting results. Each business has an attribute called full_address so we decided to extract the zip code from there and aggregate the data by the list of valid zip codes. In order to accomplish this, we needed zip code specific data, such as the zip code centroids. This we obtained from the web (21) in csv format and filtered for valid zip codes, the invalid ones representing PO boxes and other special use zip codes (22). The third and final data source comes from a web database of zip code boundaries (23). The web application provides a map of an input zip code and its surrounding zip codes, which we use to manually produce a list of neighbors for each zip code region. An alternative is to estimate a list of neighbors for each zip code region by proximity to its centroid, however in order to more accurately compare our rendering results to existing Protovis/D3 implementations, we will use the actual list of neighbors for now.

Pipeline for obtaining data and rationale:

1. **Filter Yelp data by minimum reviews:** This improves reliability of results by removing businesses which has too few reviews, those could be faked reviews or very biased.

2. **Filter Yelp data by the city we want to visualize:** Narrow down the list of businesses to those in our target city, now we can get a list of city zip codes that appear in the Yelp data. Here we have the problem of missing or invalid zip codes that came with that data, refer to step 4 for the resolution.

3. **Group Yelp data by zip code:** We have many data attributes to choose from here, we decided to compute the following aggregates for each zip code: average price, average star ratings, total number of businesses, cumulative sum of reviews and complete list of business categories (without duplicates). The latter attribute could be used to generate a word cloud of the most common types of businesses within each region.

4. **Filter zip code database by the target city name and zip code type:** We observed that city names may not be unique so we use the city’s area code as a back up filter. For our project we decided to omit PO boxes and other special use zip codes.

5. **Do an inner join of Yelp data, zip code database data and zip code neighbors:** An inner join ensures that any invalid zip codes from step 2 is left out.

6. **Finally save this dataframe as a json dictionary for use by the algorithm generating script.** This data processing script is generalized such that a whole different set of data can be produced only by changing a few function variables (minimum number of reviews, city name etc.)

Figure 4: Processed Yelp Data

Las Vegas Zip Code Map

Figure 5 is the real zip code map of Las Vegas. We decided to use it as the gold standard, from where we can compare our algorithm and implementation in both neighboring and location aspects.

Protovis and D3 Implementation

Figure 6 shows our implementation of Dorling Cartogram over Yelp data based on Protovis. Each circle represents a zip code region in the city of Las Vegas, and both the radius and the color of the circle encode the average rating of all the businesses within the region.

Protovis library uses build-in functions

```python
collisionConstraint = pv.Constraintcollision(function(d) d.radius + 1),
positionConstraint = pv.Constraintposition(function(d) d.position),
linkConstraint = pv.Force.spring(100).links(borders);
```

to construct the Dorling Cartogram.

And with the comparison of the real Las Vegas Zip Code Map in Figure 3, one can conclude that the neighboring constraint is not well maintained.

Figure 7 shows our implementation of Dorling Cartogram over Yelp data based on D3.
Figure 5: Real Las Vegas Zip Code Map

Figure 6: Dorling Cartogram over Yelp Data based on Protovis

Figure 7: Dorling Cartogram over Yelp Data based on D3

Figure 8: Original Implementation of Dorling Cartogram

Figure 9: Dorling Cartogram Final Rendering Result
## RESULTS

<table>
<thead>
<tr>
<th>Constraint 1</th>
<th>Constraint 2 (w position constraint)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dorling 1996</td>
<td>26.4</td>
</tr>
<tr>
<td>Modifications v1</td>
<td>23.6</td>
</tr>
<tr>
<td>Modifications v2 (w position constraint)</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>34.4</td>
</tr>
<tr>
<td></td>
<td>32.0</td>
</tr>
</tbody>
</table>

## FUTURE WORK

- Optimize for the ratio in which forces should be combined
- Experiment with alternative metrics for attraction force between circles
- Auto-generation of a list of neighbors from circle position
- Visualization of additional data attributes
- Improve usability of user interface

## REFERENCES

### BIBLIOGRAPHY


