Computing Fast and Scalable Table Cartograms for Large Tables

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Mohammad Rakib Hasan

Abstract

Given an $m \times n$ table T of positive weights, and a rectangle R with an area equal to the sum of the weights, a table cartogram computes a partition of R into $m \times n$ convex quadrilateral faces such that each face has the same adjacencies as its corresponding cell in T, and has an area equal to the cell's weight. In this thesis, we explored different table cartogram algorithms for a large table with thousands of cells and investigated the potential applications of large table cartograms. We implemented Evans et al.'s [40] table cartogram algorithm that guarantees zero area error and adapted a diffusion-based cartographic transformation approach, *FastFlow* [52], to produce large table cartograms. We introduced a constraint optimization based table cartogram generation technique, *TCarto*, leveraging the concept of *force-directed layout*. We implemented *TCarto* with column-based and quadtree-based parallelization to compute table cartograms for table with thousands of cells. We presented several potential applications of large table cartograms to create the diagrammatic representations in various real-life scenarios, e.g., for analyzing spatial correlations between geospatial variables, understanding clusters and densities in scatterplots, and creating visual effects in images (i.e., expanding illumination, mosaic art effect). We presented an empirical comparison among these three table cartogram techniques with two different real-life datasets: a meteorological weather dataset and a US State-to-State migration flow dataset. *FastFlow* and *TCarto* both performed well on the weather data table. However, for US State-to-State migration flow data, where the table contained many local optima with high value differences among adjacent cells, *FastFlow* generated concave quadrilateral faces. We also investigated some potential relationships among different measurement metrics such as cartographic error (accuracy), the average aspect ratio (the readability of the visualization), computational speed, and the grid size of the table. Furthermore, we augmented our proposed *TCarto* with angle constraint to enhance the readability of the visualization, conceding some cartographic error, and also inspected the potential relationship of the restricted angles with the accuracy and the readability of the visualization. In the output of the angle constrained *TCarto* algorithm on US State-to-State migration dataset, it was difficult to identify the rows and columns for a cell upto 20° angle constraint, but appeared to be identifiable for more than 40° angle constraint.

Acknowledgements

Firstly, I would like to thank my supervisors, Dr. Debajyoti Mondal and Dr. Kevin A. Schneider for their valuable guidance. I am also grateful for the graduate funding support from the Department of Computer Science at the University of Saskatchewan and the grant from the *National Sciences and Engineering Research Council of Canada (NSERC)*. It led me to conduct this wonderful research work.

I am very much thankful to my wife and my daughters for giving me the courage to finish what I started. I am lucky to have some really good friends and lab mates who supported me all the way through.

Most importantly, I am eternally indebted to my parents, who taught me to accept failures, keep dreaming of success and never give up on something just because it is difficult.

Dedicated to my creator, my lovely wife (Zinat Ara), both of my precious daughters (Pariza Eleanor and Irem Maryam), and my parents.

Contents

List of Tables

List of Figures

List of Abbreviations

LOF	List of Figures
LOT	List of Tables
Baseline	An errorless table cartogram algorithm proven by Evans et al. [40]
FastFlow	A fast flow and diffusion based cartographic transformation algorithm proposed by Gastner et al. [52]
TCarto	Our proposed scalable table cartogram algorithm. It usually refers to TCarto DivCon approach
TCarto Parallel	TCarto algorithm with the computation of column based parallelization
TCarto DivCon	TCarto algorithm with the computation of quadtree based parallelization
MRAE	Mean Relative Area Error
RMSE	Root Mean Square Error
MQE	Mean Quadratic Error
AR	Aspect Ratio
α	Number of Concave Cell
CPU	Central Processing Unit
GPU	Graphics Processing Unit

1 Introduction

Information visualization is commonly used to analyze data and communicate information to the end users. Information visualization refers to the study of visual representations of data for enhancing human's ability to perceive information [103, 91]. It also intensifies human perception by proper visual computations with abstract information [24]. Simply, it is the process to visualize or present data in a meaningful way so that a user can easily and correctly comprehend the underlying information [116]. The area of information visualization has interacted with various fields such as interactive interface design, cartographic computation, computer graphics, psychology, and business methods. *Cartographic data representations* or *cartograms* [56, 135] combine the statistical and geographical information on a thematic map, where some statistics (e.g., population, area, income) control the areas of geographical regions (e.g., continents, countries, states) based on some predefined scale.

Area cartogram is a type of cartogram where regions of the cartogram are proportional to the variables they represent. There are various types of area based cartograms [109]. Some cartograms preserve shapes of the regions with exact area accuracy (the difference between area and the value being represented) while conceding the neighborhood adjacency. Some cartograms sacrifice area accuracy to keep the shapes of the regions and their neighborhood adjacency. A cartogram is a contiguous cartogram if it preserves the neighborhood adjacency. In Figure 1.1(a), the contiguous area cartogram preserves the shapes and the neighborhood adjacency while compromising some area accuracy. In contrast, the non-contiguous cartogram on Figure 1.1(b) holds the area accuracy and the exact shapes but does not maintain the neighborhood adjacency.

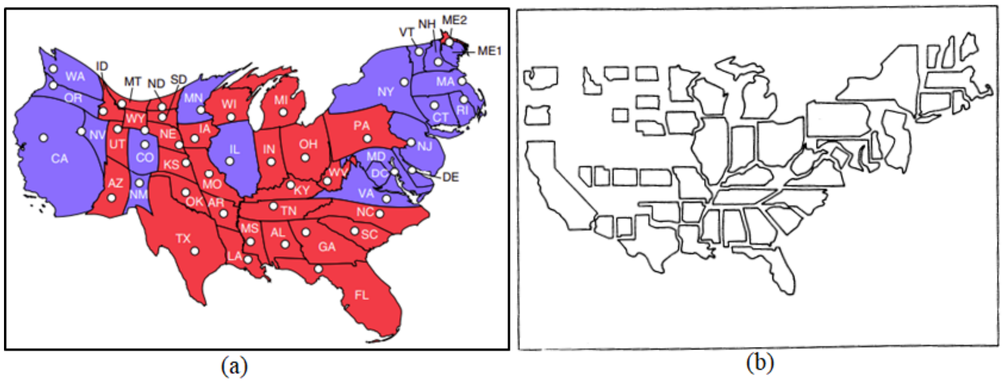


Figure 1.1: (a) The 2016 US Electoral College vote represented using a contiguous cartogram [52], (b) population of the United States in 1970 presented using non-contiguous cartogram [81].

Table cartogram, which has been proposed in 2013 [40], is an area contiguous cartogram that preserves the neighbourhood adjacency while distorting the shapes of the regions. A *table cartogram* is a cartographic representation for a two dimensional positive matrix or tabular data (See Figure 1.2). Given a positive $m \times n$ matrix of weights, a table cartogram represents each cell as a distinct convex quadrilateral with an area proportional to the corresponding cell's weight such that the cell adjacencies are preserved, the quadrilaterals form a partition of a rectangle R, and the sum of all weights is equal to the area of R rectangle keeping the outside border fixed.

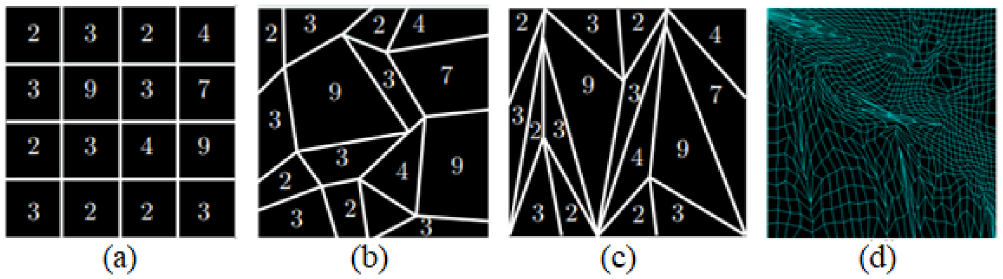


Figure 1.2: (a) A table R with 2×2 grid, (b) a table cartogram of R using constraint optimization, (c) Evans et al.'s [40] output for R (based on a theoretical proof), (d) a cartogram for a large table (32×32) based on constraint optimization.

Many researchers have conducted theoretical and application based research on cartograms. In 2018, Gastner et al. [52] proposed a diffusion-based method for density equalizing cartogram. In this approach, the population of each region is initiated with the density ($= population/area$) proportional to the weight of that region. In each iteration, populations flow to the low density regions from the high density regions, and the movements of the population deform the shape of each region. The areas of the high density regions increase, and thus their density decrease with the movement of the initial population. The opposite happens to the low density regions, and the iteration stops when all the regions have the same density. This fast approach is auspicious for traditional cartograms; however, it remains unknown how a traditional cartographic algorithm performs for a table cartogram.

In 2013, Evans et al. [40] showed that every $m \times n$ table admits a table cartogram. Their theoretical proof is based on first partitioning the table into weighted triangles, then computing cartograms for the triangles using the concept of barycentric coordinates, and finally, merging the triangles based on the necessity to obtain the final table cartogram. Although the proof guarantees to produce a table cartogram with zero cartographic error, the authors pointed out that the output may not be visually pleasing (See Figure 1.3(top-right)) and good heuristics are needed to improve the aesthetics. Recently, Inoue and Li [71] showed an optimization-based approach for the construction of table cartograms by changing the bearing angles on the edges of the polygons.

Several attempts [95, 96, 39] have been made to compute aesthetic table cartograms. Researchers have also looked at the scope for using table cartograms in practice, in particular, the data and types of tasks that are well-suited for using a table cartogram [96]. However, all prior applications of table cartograms examined only various forms of tabular data (e.g., periodic tables, temperature calendar, population demographics, line charts, etc) with a few hundred cells. Thus, table cartograms for large tables with several thousands of cells and their other applications, beyond tabular data visualization remained unexplored.

1.1 Motivation

The primary aim of this thesis is to explore table cartograms for large tables. The plan was to design a fast and scalable algorithm for computing cartograms. Initially, the table cartogram generation technique by Evans et al. [40] appeared promising because of the errorless output. Based on the theoretical proof by Evans et al. [40], we implemented the table cartogram algorithm. It could run for large tables with minimal error; however, it generated skinny triangles with a small 'average aspect ratio' (See Equation 2.8). The outputs of this algorithm for the image-based applications also compromised the neighborhood adjacencies. Figure 1.3(c)(top) shows the skinny triangles, and Figure 1.3(c)(bottom) shows that yellow-green(land) region divided the blue(ocean) region into several blue-colored holes, but blue (ocean) region is always connected in the original image. Thus, this algorithm became ineffective for image-based applications. For convenience, we refer this errorless table cartogram technique as **Baseline** [40] for the rest of the paper.

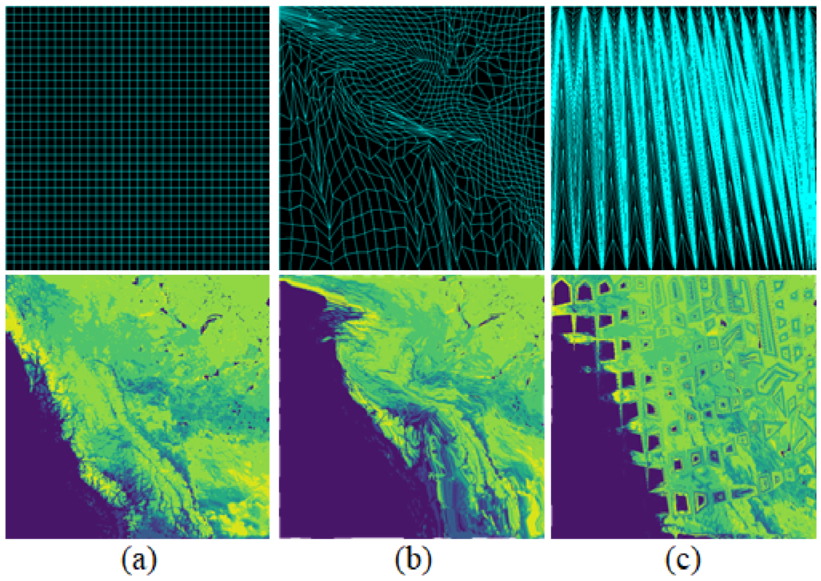


Figure 1.3: Top row represents the 32 by 32 grid corresponding to their below images. Bottom row shows, (a) a contour plot of the portion of solar energy reflected from the surface of the earth (ALBEDO) over the western provinces of Canada, (b) **TCarto** output that transforms the contour plot based on the table of soil moisture (SH20) values for 32 by 32 grid with cartographic error (See Eq. 2.2) = 0.0961 and average aspect ratio (See Eq. 2.8) = 0.6130, (c) Evans et al.'s[40] algorithm (**Baseline**) output with cartographic error = 0.0001 and average aspect ratio = 0.1689 .

Then, we decided to adapt the widely used physics-based cartogram algorithm. The diffusion-based cartogram technique by Gastner et al. [52] was promising among the existing algorithms in the literature due to its fast computation capability. Since this cartogram technique was not designed for table cartograms, we amended this algorithm by fixing the outer boundary. The output was promising for the data with no high spikes and many local optima; however, it generated concave and overlapping cells for the data distribution consisting of sudden value changes and many local maxima (See Figure 1.4). We will describe the complexity and challenges of concave cells for image-based applications shortly. Since these concave and overlapping cells mislead the visual interpretation of the image-based applications, it became a necessity to design an algorithm that can simulate large tables with thousands of cells without generating any concave and overlapping cells. For convenience, we refer to this diffusion-based cartographic approach [52] as **FastFlow** for the rest of the paper.

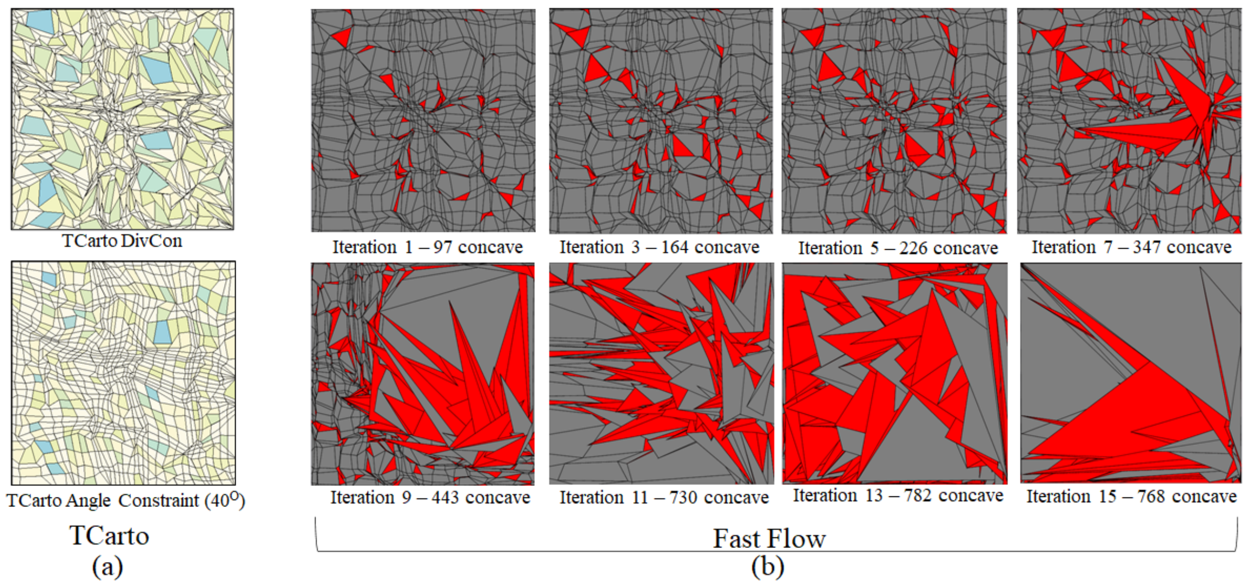

TCarto DivCon
TCarto Angle Constraint (40°)
TCarto
(a)
Iteration 1 – 97 concave
Iteration 3 – 164 concave
Iteration 5 – 226 concave
Iteration 7 – 347 concave
Iteration 9 – 443 concave
Iteration 11 – 730 concave
Iteration 13 – 782 concave
Iteration 15 – 768 concave
Fast Flow
(b)

Figure 1.4: The table cartogram outputs for the US state-to-state migration data of year 2017, (a) top is **TCarto DivCon** output, and bottom is **TCarto DivCon** with 40° angle constraint, where blue region has the high weight and yellow has the low weight, (b) **FastFlow** outputs with concave and overlapping cells at different iterations, where red cells are concave quadrilaterals.

Motivated by *Tutte's Embedding* and the concept of *force-directed layout* [47, 50], we designed an algorithm for the table cartogram, where each node had moved based on the position of its neighboring nodes and the weights of neighboring regions. Quadratic programming [62, 43] aided in finding the optimal positions of the nodes maintaining the convexity constraints, and all the nodes moved towards the globally optimized solution on each iteration. Parallelization made it possible to run the algorithm for large tables with thousands of cells, and this ability had opened up the possibility of applying the idea of table cartograms on images and geographic contour plots, and thus allowed us to explore new application areas. We proposed two approaches of this algorithm: column-based (**TCarto Parallel**) (See Algorithm 1) and quad-tree (**TCarto DivCon**)(See Algorithm 2) based parallelization. We refer **TCarto DivCon** algorithm as **TCarto** for the rest of the paper.

Table cartogram transforms the initial equal-area grid cells based on each cell's weight, sustaining the cell adjacencies and each quadrilateral cell's convexity. Because of these characteristics, it can serve as the underlying algorithm of many applications, e.g., revealing the spatial correlation between variables, understanding of clusters with their densities in scatter plot, generating the visual effect of the illumination increase of the light sources in an image. It may also assist in formulating population-density equalizing cartograms [135, 51, 52] and generating a unique-identifier matrix barcode system [92, 145, 133].

Table cartogram distorts the input cell into convex quadrilateral only and restrict distorted cells to transform a concave quadrilateral. However, traditional cartographic transformation do not follow this restriction. So, when it comes to large tables, it is natural to ask whether it is important to keep the cells as convex polygons, which makes table cartograms unique from cartographic map transformations. Figure 1.5 illustrates some image-based applications that had been explored in this thesis.

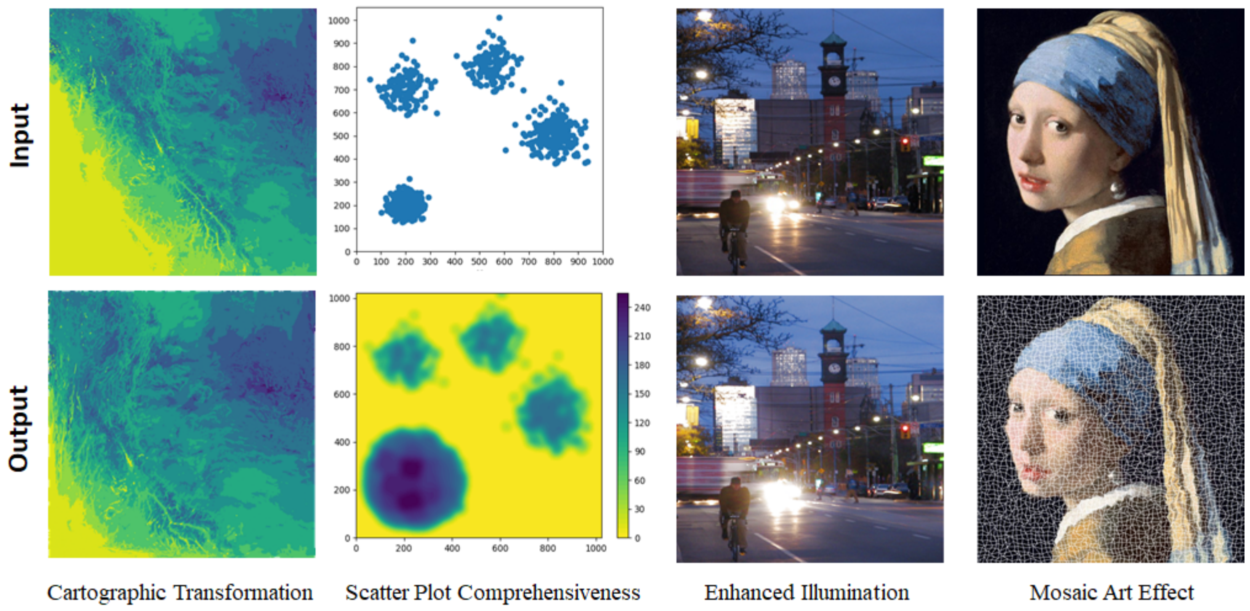


Figure 1.5: Large table cartograms applied to different applications.

Challenges with concave cell: While transforming images, concave cells may pose more challenges than the convex ones. Figure 1.6 illustrates a piecewise affine transformation [115] of a square-size image to fit into a convex and a concave polygon. An ideal situation would be that all the colors in the image maintain their pixel ratio even after the transformation, i.e., the area of the visual attributes will be scaled linearly based on the area transformation. However, for concave areas, the transformation often appears to be more distorted.

For example, Figure 1.6(a) illustrates an initial image, and Figure 1.6(b) and (c) illustrate piecewise linear transformations into a convex polygon and a concave polygon, respectively. The transformation maps the two triangles of the Delaunay triangulation of the original image to that of the destination image. The concave corner enforces the circular shape in the image to appear more distorted compared to the convex corners, e.g., the initial red circle appears to be a heart shape inside the concave polygon. Furthermore, we need to

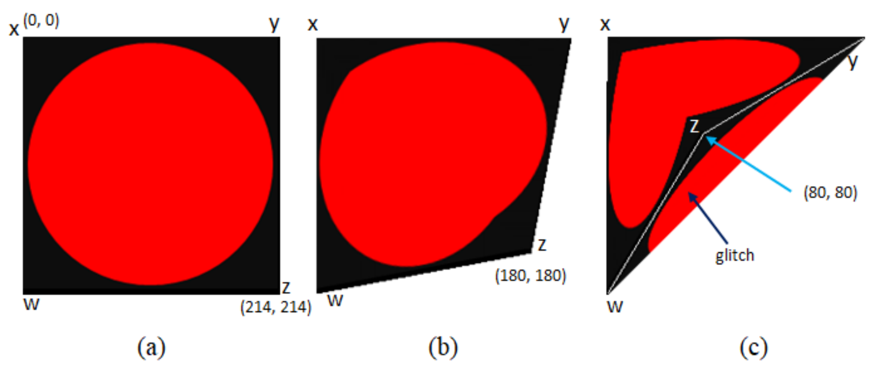


Figure 1.6: Piecewise affine transformation of (a) an image, (b) into a convex polygon, (c) into a concave polygon.

remove any glitch that may appear due to the extra triangles that appear in the Delaunay triangulation for the concave case, e.g., see the triangle marked 'glitch'.

1.2 Research Questions

The main focus of this thesis is to investigate table cartogram techniques for large tables with thousands of cells. Researchers have already investigated different applications [95, 96] with tabular data visualization of table cartogram for small table. Our plan is to advance the existing research by adapting or implementing table cartogram for large table and by exploring applications even beyond tabular data visualization. For this, we consider these two research questions.

A. Potential Applications of Large Table Cartograms

- RQ_{A1}: What are the potential applications of large table cartograms beyond visualizing tabular data?
- RQ_{A2}: How crucial is it to maintain the convexity of the cells in such applications?

Since existing approaches have already applied constraint based techniques for smaller tables, we were motivated to examine whether it can be applied to large tables. We also considered to investigate the feasibility of adapting cartographic transformation algorithms to generate table cartograms.

B. Algorithms for Computing Large Table Cartograms

- RQ_{B1}: Can constraint optimization based approach be used to compute table cartograms for large tables with small cartographic error?
- RQ_{B2}: How constraint optimization based table cartogram compares to the cartographic map transformation algorithms in various quality metrics (e.g., number of convex cells, aspect ratio, etc.)?

- RQ_{B3}: How table cartogram algorithms perform with respect to different quality metrics for different types of data distribution?

There are several major criteria such as accuracy, readability of the visualization, and processing time. Some algorithms may have better accuracy, but may generate visualizations with lower readability of the visualization. Some may be faster, but may provide lower accuracy and readability of the visualization. The appropriate algorithms are chosen based on the needs or requirements.

C. Impact of Angle Constraint

- RQ_{B4}: What are some potential trade-offs or relationships (if any) among the accuracy, processing time and other quality metrics?

Researchers have applied angle constraint on small table to generate table cartograms with better readability of the visualization [71]. Our next goal is to inspect the viability of angle constraint for large tables and also to investigate how accuracy and readability of the visualization behave with this angle constraint.

- RQ_C: Can an angle constraint based algorithm generate cartograms for large table with at least the readability to identify the columns and rows for a cell? How accuracy and this readability of the visualization relate with the threshold angle?

The above research questions and various prospects of table cartogram motivated us to design and implement the necessary algorithms, produce and analyze the results, and explore more in depth.

1.3 Contributions

We explored the existing table cartogram algorithms for a large table with thousand of cells.We implemented the **Baseline** table cartogram technique which was theoretically proven to ensure zero area error by Evans et al. [40]. But, for a large table this algorithm generated skinny triangles (See Figure 1.3). We adapted the existing **FastFlow** cartographic transformation technique by Gastner et al. [52] to generate table cartograms, which may produce cartograms with concave and overlapping cells (See Figure 1.4(b) and 1.6) for tables with high spikes in values and with many local maxima.

In this thesis, we proposed a new table cartogram generation technique that leverages *force directed layout* and overcomes the shortcomings of these existing algorithms. *Force-directed layout* [47, 50] generates an aesthetically pleasing visualizations for graphs by attracting the adjacent nodes and repelling the others. These characteristics of the *force-directed layout* motivated us to propose a new table cartogram construction technique.

Our proposed table cartogram technique, **TCarto**, is a constraint-based optimization approach, where each node is moved based on the position of its neighboring nodes and the weights of neighboring regions maintaining a minimum distance from its neighbors. **TCarto** leverages parallel computing to deform a table via local optimization using quadratic programming [62, 43]. For parallelization, we considered column-based approaches and quadtree-based approaches. The quadtree-based approach dispersed the weights faster than the column-based approach; thus, quadtree-based approach performed faster. **TCarto** can handle tables over hundred thousands of cells and maintains the convexity of each cell.

We presented several real-life applications (RQ_{A1}) of table cartograms, e.g., for analyzing correlations between geospatial variables, understanding clusters in scatterplots, and creating visual effects in images. Though **Baseline** was errorless, it generated skinny triangles with a higher aspect ratio that distorted the output images so much that this approach became impractical for the image-based applications (See Figure 1.3). We showed that both **TCarto** and **FastFlow** (and thus large table cartograms) could potentially be useful in these real-life scenarios. We also demonstrated how crucial it is to maintain the cells' convexity on these applications (RQ_{A2}).

We compared the performances of **Baseline**, **TCarto**, and **FastFlow** empirically with two different real-life datasets: a meteorological weather dataset and a US State-to-State migration flow dataset. In particular, we examined well-known metrics for measuring cartographic errors, the number of concave cells, and the average aspect ratio of the cells for RQ_{B1}, RQ_{B2}, and RQ_{B3}. **Baseline** generated outputs with minimal errors but with a lower average aspect ratio that reduced the readability of the output visualizations highly for both datasets. For the meteorological dataset, where the data distribution had no high spikes and many local optima, both **TCarto** and **FastFlow** performed well, and **FastFlow** produced only a few concave cells. For the migration dataset with many sharp local optima, both **TCarto** and **FastFlow** produced low-quality output, whereas for **FastFlow** we observed hundreds of concave cells.

We also described the potential relationships between the accuracy and the readability of the visualization for all table cartogram algorithms (RQ_{B4}). **FastFlow** and **TCarto DivCon** both performed well, but **FastFlow** is preferable when cells' convexity is not important. When this cells' convexity is crucial, **TCarto** is recommended (RQ_{B4}). We also demonstrated the relationship of the grid size with accuracy and processing speed for a constraint based table cartogram. Accuracy does not change linearly and processing speed increases exponentially by the increment of the grid size. We also showed how enforcing additional angle constraints can help enhancing the readability of the visualization and demonstrated the relationship of threshold angle with *cartographic error* and *mean aspect ratio* (RQ_C). The augmented **TCarto DivCon** with 40° angle constraint, produced table cartogram, where row and column could be easily identified for a cell.

1.4 Methodology

There are several development methodologies, e.g. waterfall model [26, 12, 112], agile methodology [12, 70], SCRUM development process [125], feature-driven development [110], and different hybrid models that we can follow as our research methodology. Each of these methodologies has its own advantages and disadvantages over others. The waterfall model and agile methodology both were suitable for our research project based on our objectives, requirements, development plan, and different risk factors. We followed the waterfall model [112] for this research project since it is suitable for projects with fixed, transparent and well-defined requirements. It is also straightforward to comprehend and use. Since this research project contains well-established requirements and few contributors, the waterfall model was chosen to carry out the research.

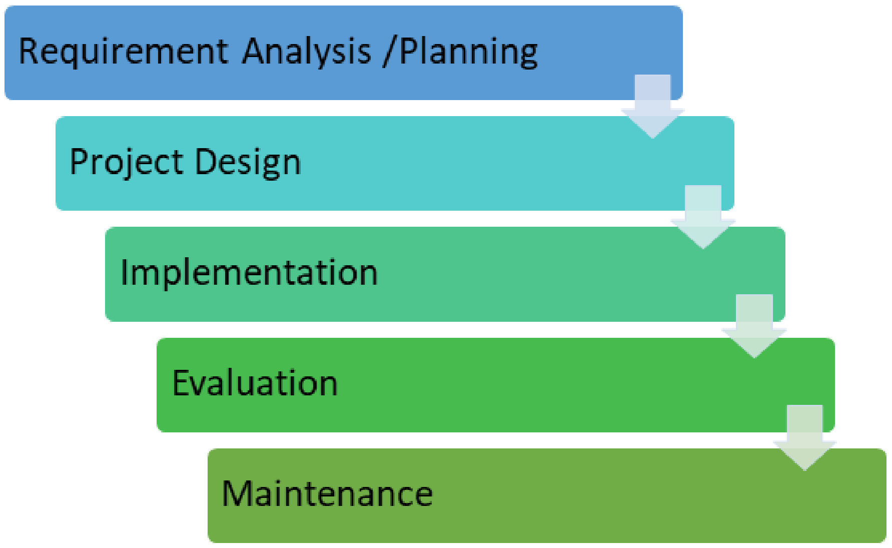


Figure 1.7: Basic diagram of waterfall model

At the beginning, we figured out all the requirements and steps to tackle the research questions. We went through the previous related works and planned the possible directions, measurements, and tasks. We also analyzed and identified the potential challenges and obstacles. In the project design phase, we planned how to implement and adapt the existing algorithms, and designed our proposed table cartogram algorithm. We also figured out various possible applications of table cartograms, and their different metrics for the evaluation. We explored several technologies and their corresponding libraries for the implementation and development of the algorithms. We selected python as a programming language because of its rich libraries and documentations. At the implementation stage, we implemented a couple of existing algorithms and our proposed algorithm as well as the possible applications for the large table cartogram. In the evaluation stage, we calculated and compared the measurement metrics for the three table cartogram algorithms and analyzed the resultant data to answer all the research questions. For this thesis project, we did not use the maintenance stage in particular; however, we always used github (`https://github.com/rakib045/TCarto`) for code versioning, documentation and project management. At the end, we prepared some tutorials for the end users.

1.5 Chapter Organization

We organized this thesis document with seven chapters. We described how we approached to the research questions by completing development tasks and analyzing the resultant outcomes in these chapters (See Figure 1.8).

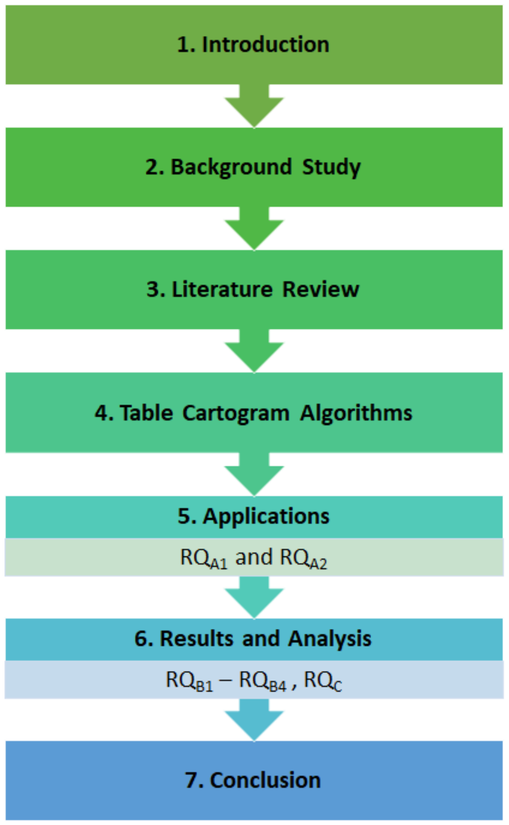


Figure 1.8: Structure of the thesis.

Chapter 1 explains the research topic, the motivations, the research questions, our approach towards them and finally a summary of the contributions.

Chapter 2 covers the background information, methods, materials and resources that anyone needs to know to comprehend the rest of this research document.

Chapter 3 provides an illustrated overview of the previous related works of different cartograms and table cartograms. It also describes the recent works of the table cartograms, different algorithmic techniques and parallel computation.

Chapter 4 illustrates the design and implementation of our proposed table cartogram algorithm (**TCarto**) with parallalization. It also includes a discussion of two of the existing algorithms that we have implemented and augmented to compute large table cartograms.

Chapter 5 demonstrates several prospective applications of large table cartograms. It also explains the importance of cells' convexity for each application.

Chapter 6 describes couple of datasets with their sources and data pre-processing workflows. It also presents the results and analysis with those datasets for different algorithms based on various performance metrics. It devises the answers and recommendations for the research questions after analyzing and visualizing the resultant data.

Finally, Chapter 7 holds the concluding remarks of this research. It describes the contributions and contains the discussions on the limitations of the research. It finishes with the possible directions of the future research.

1.6 Declaration

Throughout this research document, the term ‘we’ refers to the author and the reader following the computer science norm for scientific writing.

I confirm that I had conducted this research and written this document under the supervision of my advisors, Dr. Debajyoti Mondal and Dr. Kevin A. Schneider. Part of the results had been accepted to be published at the *16th International Symposium on Visual Computing (ISVC 2021)* [59].

1.7 Summary

Here, we narrowed down our research scope followed by a summarized description on cartogram. We discussed our motivations, defined the research questions, mentioned our contributions on this thesis and described the methodology for this thesis. At the end, we discussed the organization of this thesis document as well as necessary declarations for this thesis.

2 Background

In this chapter, we describe several topics that will be helpful to understand the following chapters of this thesis document. Table cartogram is the prime topic of this whole research work. In the first section, we define cartograms, demonstrate different types of cartograms and the table cartogram. Next, we demonstrate an image transformation techniques, piecewise affine transformation and image warping associated with that technique. Then, we describe all the quality measurement metrics that we have used in our 'Results and Analysis' chapter. At the end of this chapter, we describe different types of quadrilateral based on convexity in the miscellaneous section.

2.1 Cartogram

Cartograms refer to the thematic maps that present the visualization with the integration of statistical and geographical information to deliver better perceptions into the pattern, trends, and outliers in the real world [109]. In cartograms, the areas of geographical regions are distorted by scaling in proportion to some statistical variables. These geographical regions could be continent, country, state, or even specific portion, whereas the statistical variables could be a single statistical variable (e.g., population, income, area), or some arithmetic computation of single or multiple variables (average income, income per person). Cartograms have been studied and used since 1870 by geographers, cartographers, economists, social scientists, data analysis and visualization researchers, and many others. Researchers demonstrated and implemented different types of cartograms for achieving or improving the accuracy on statistical (cartographic error), geographical (preserving the outlines of geographic regions), and topological (keeping the correctness on neighborhood adjacencies of the geographic regions) aspects. Even newspapers in the United States used cartograms to emphasize their stories as early as in 1921 to present energy consumption in different US states [1] (See Figure 2.1).

Several systematic ways can be followed to categorize this large number of different types of cartograms. Based on the design dimensions, cartograms can be categorized into four major types: contiguous, non-contiguous, Dorling, and rectangular cartogram. We will use the notion of contiguous, non-contiguous, and rectangular cartogram in this document.

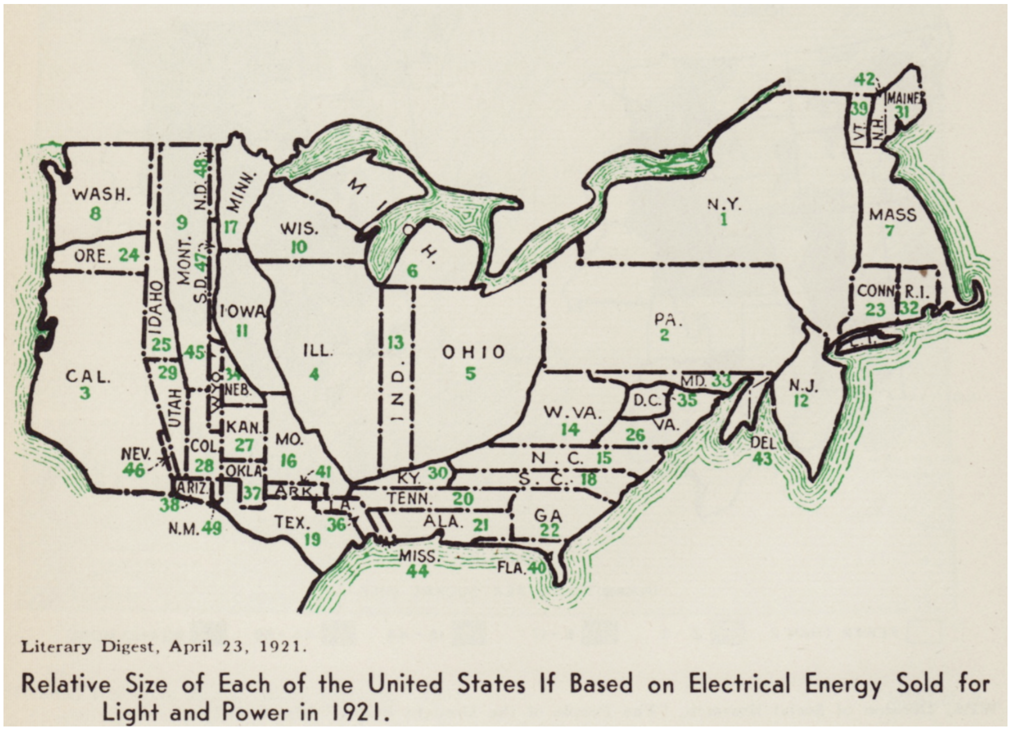

Literary Digest, April 23, 1921.
Relative Size of Each of the United States If Based on Electrical Energy Sold for Light and Power in 1921.

Figure 2.1: 1921 cartogram of the USA based on electrical energy sold for light and power in the Literary Digest [1].

2.1.1 Contiguous and Non-contiguous Cartogram

Contiguous cartograms are those that deform the geographic regions of a map based on statistics in such a way that the geographic region outlines and the neighborhood adjacencies among those regions are maintained. They are also called *deformation cartograms* [5, 52]. In these types of cartograms, the original geographic regions on the map are usually altered by zooming, pulling outside, pushing inside, or stretching the boundaries of the regions. It helps to preserve the shapes and neighborhood adjacencies of the regions (See Figure 2.2(b)). For consistency, we will mention this type of cartogram as *contiguous cartogram* in the rest of this research document. These cartograms contain higher topological accuracy, whereas a trade-off exists between statistical and geographical accuracy.

Non-contiguous cartograms transform the geographic regions based on the statistical information focusing only on preserving the original shapes of the geographic regions while ignoring the adjacencies of the regions completely. In these cartograms, the geographic regions become undistorted and only scaled up or down based on the statistical information. It makes these cartograms perfect statistical accuracy and perfect shape preservation. However, these cartograms fail to preserve the topology of the original map (See Figure 2.2(c)). This cartogram's topology is difficult to understand since the geographic regions either touch erroneously or do not contact each other at all. Sometimes, the regions may become very small, and it is very tough to comprehend without proper topological information.

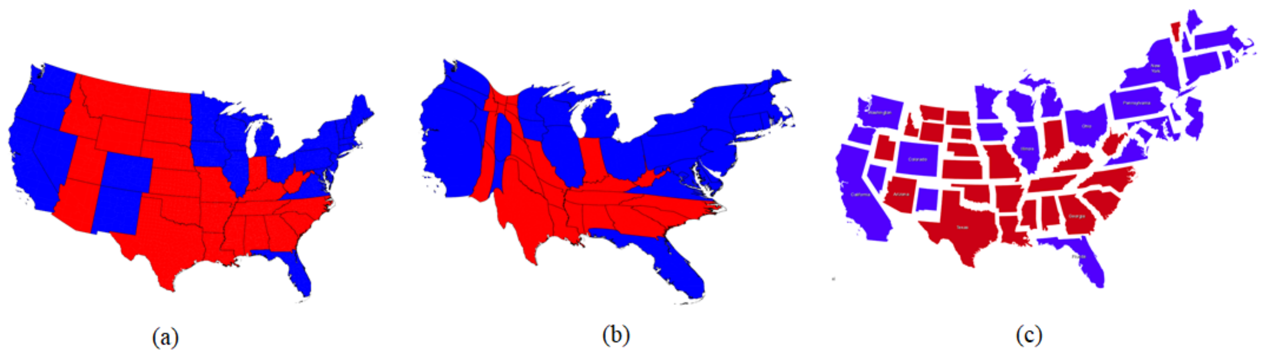


Figure 2.2: The 2012 US electoral college vote of re-electing former *President Barak Obama* (a) standard input image with red for the *Republican Party* and blue for the *Democratic Party* [28, 104], (b) college vote using a contiguous cartogram [104], (c) same election results using non-contiguous cartogram [89].

2.1.2 Rectangular Cartogram

In these cartograms, all the geographic regions are represented as rectangles. The size or area of each of those rectangles is proportional to their corresponding statistical information. The rectangular cartogram disregards the original shapes of the geographic regions. This cartogram has higher topological accuracy but can not guarantee perfect accuracy.

Firstly, an initial boundary rectangular layout is calculated from the topology of the map. Then, the appropriate coordinates of the rectangles are computed with a segment moving heuristic. The vertical and horizontal segments of that rectangular layout have been moved iteratively to increase the statistical accuracy. After that, the topology can be preserved at the cost of error, or the topology might be compromised to achieve perfect statistical accuracy. A rectangular cartogram [138] depicts the highway distances in kilometers of different states of US (See Figure 2.3).

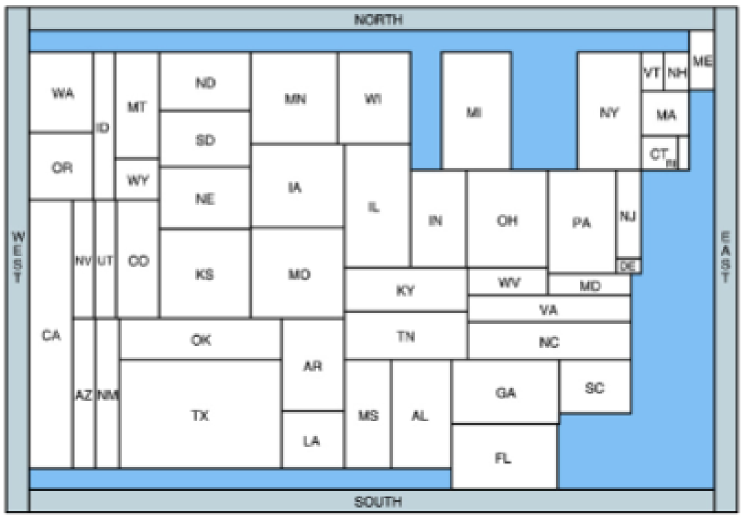
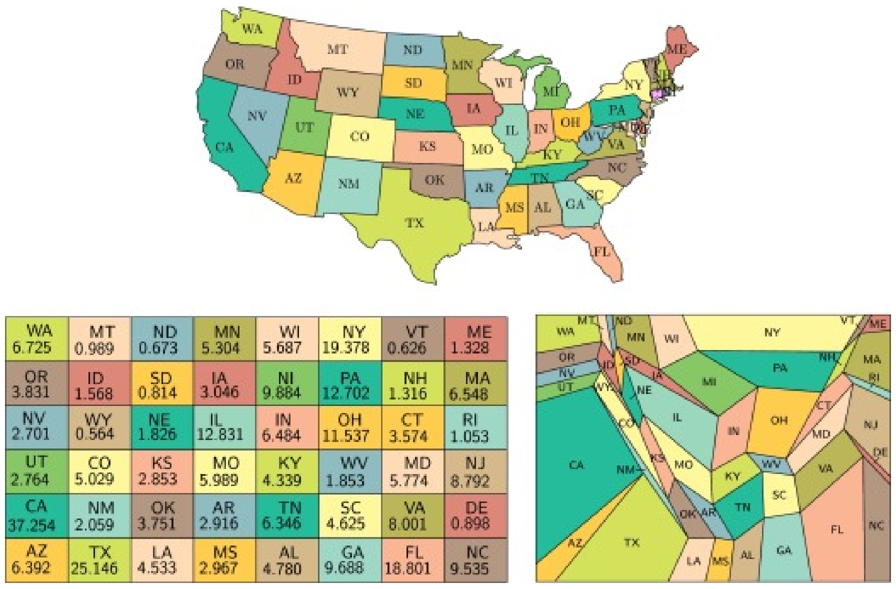


Figure 2.3: Rectangular cartogram presenting the highway distances of different states at US [138].

2.1.3 Table Cartogram

Table cartogram is an area-based contiguous cartogram with several specific rules. Table cartograms also transform the regions based on the statistical information. Table cartograms maintain the neighborhood adjacency but do not preserve the shapes of the regions. Some table cartograms guarantees no area error [41], and some sacrifice statistical accuracy [71, 96, 59] for the aesthetically pleasing visualization or maintaining the readability of the visual representations.

A table cartogram is a two-dimensional grid-like rectangular table. Initially, all the cells are equal-area rectangles, and statistical values or positive weights are assigned to each cell as input such that the total area equals the sum of all weights (See Figure 2.4(bottom-left)). This cartogram transforms the grid cells into convex quadrilaterals where the area of each cell is proportional to its corresponding weights. This table cartographic transformation also maintains the neighborhood adjacencies of the regions while keeping the outside boundary fixed. In Figure 2.4(bottom-right), the neighborhood adjacencies are maintained with precise statistical accuracy, but the shapes are compromised.



WA 6.725	MT 0.989	ND 0.673	MN 5.304	WI 5.687	NY 19.378	VT 0.626	ME 1.328
OR 3.831	ID 1.568	SD 0.814	IA 3.046	NI 9.884	PA 12.702	NH 1.316	MA 6.548
NV 2.701	WY 0.564	NE 1.826	IL 12.831	IN 6.484	OH 11.537	CT 3.574	RI 1.053
UT 2.764	CO 5.029	KS 2.853	MO 5.989	KY 4.339	WV 1.853	MD 5.774	NJ 8.792
CA 37.254	NM 2.059	OK 3.751	AR 2.916	TN 6.346	SC 4.625	VA 8.001	DE 0.898
AZ 6.392	TX 25.146	LA 4.533	MS 2.967	AL 4.780	GA 9.688	FL 18.801	NC 9.535

Figure 2.4: (top) Different states of US in the original geographic map, (bottom-left) 6 x 8 grid map of the states of US, (bottom-right) corresponding table cartogram output for the population of the US states in 2010 [41].

2.2 Image Transformation

Image transformation plays an important role in the image-based applications of table cartograms. First, we consider the input image into $m \times n$ grid cells, split the input image into smaller rectangular pieces, and then simulate the table cartogram for those grid cells to deform into convex quadrilaterals. After that, we use image transformation to transform all rectangle image pieces to the convex quadrilateral faces based on the output of the table cartogram algorithm and then merge all smaller image pieces with convex quadrilateral faces together into the output image. Image warping performs this image transformation and merging sequentially.

Image transformation is one of the common tasks in image processing. Transformation is a function that takes an input image and converts it to a different output image [3]. Let $f(x, y)$ and $G(x, y)$ are input and output image respectively, where image represents a 2D matrix with x and y are the co-ordinates. If the transformation function is T, we can consider the below equation,

$$G(x, y) = T\{f(x, y)\} \tag{2.1}$$

Linear transformations include geometrical raster transformations such as rotation, scaling, skewing, and perspective distortion. Linear transformation preserves vector addition and scalar manipulation. Matrix is usually used to implement image transformation. Affine transformations helps to achieve perspective distortion of an image.

2.2.1 Affine Transformation

Affine transformations assist to achieve scaling, rotation, and skewing [2]. We can accomplish perspective distortion using projection matrix. The below matrix can represent affine transformation:

$$\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}$$

Scaling and rotation can be performed by modifying the values of a_1, a_2, a_3 and a_4. It is called rotation matrix. So, the rotation matrix is,

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

b_1 and b_2 contributes towards the translation of the image. It is called translation matrix.

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Rotation and translation can be achieved by modifying the rotation and translation values in the same matrix. $\begin{bmatrix} c_1 & c_2 \end{bmatrix}$ is called projection matrix and we can get perspective distortion with this matrix. Let x and y be the co-ordinate values of a pixel, p of an input image. Let x' and y' be the updating values of the pixel, p after geometrical raster transformation. We can calculate the updated values by multiplying the input values with the transformation matrix.

$$\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Affine transformations preserve parallelism if projection matrix is zero (See Figure 2.5).

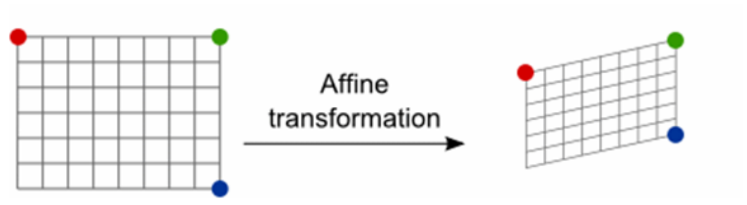


Figure 2.5: Affine image transformation.

2.2.2 Image Warping with Piecewise Affine Transformation

Piecewise affine transformation is the process that distorts an input image by image warping with the help of affine transformation based on the movement of some specific pixel points from the source position to the destination position. In piecewise affine transform, some nodes at the input image are supposed to move at their destination position on the output image. The input image has been warped using affine transformation so that those nodes fall at their destination positions. The features or shapes, or pixels within those nodes have stretched or sheared. In Figure 2.6(a), blue nodes are at the source position, and blue dots in Figure 2.6(b) are their destination positions. In this example, destination points have been calculated using the sine function. After defining the source and destination positions of the nodes, piecewise affine transformation generates the output image, Figure 2.6(b) from an input image Figure 2.6(a).

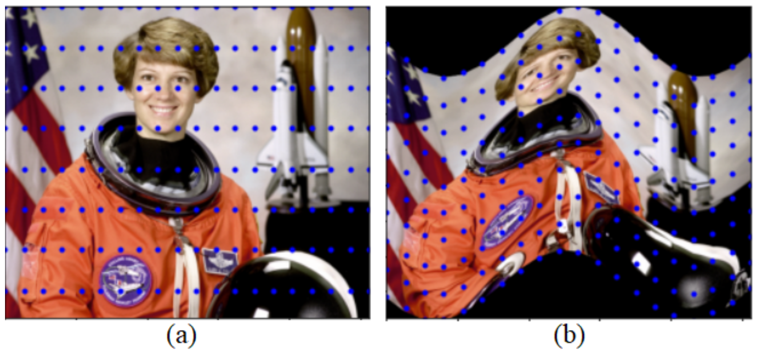


Figure 2.6: (a) Input image with blue nodes (source position) as grid points, (b) output image after applying a piecewise affine transformation with the destination point.

2.3 Quality Measurement metrics

There are several metrics for the statistical accuracy calculation. Different researchers use different error calculations based on their needs. We will use *Cartographic Error*, *Mean Relative Area Error (MRAE)*, *Root Mean Square Error (RMSE)*, and *Mean Quadratic Error (MQE)*. These metrics are important for those applications where statistical accuracy is crucial [132, 52].

2.3.1 Statistical Accuracy metrics

There are several metrics for the statistical accuracy calculation. Different researchers use different error calculations based on their needs. We will use *Cartographic Error*, *Mean Relative Area Error (MRAE)*, *Root Mean Square Error (RMSE)*, and *Mean Quadratic Error (MQE)*.

Cartographic Error

One of the standard measurements of the spatial area error of each region or polygon is defined as the *Cartographic Error*, e_i. Assume that A is an $m \times n$ table. So, there are total of $m \times n$ polygons. Let $A_i^{desired}$ be the desired area or the given weight of i polygon and A_i^{actual} be the actual area after the cartographic transformation [72], then

$$\textit{Cartographic Error, } e_i = \frac{\left|A_i^{desired} - A_i^{actual}\right|}{A_i^{desired}} \tag{2.2}$$

Thus, if $N = m \times n$, the *average cartographic error* for a set of polygon, P is defined as

$$\widetilde{e_i} = \frac{1}{N}\left[\sum_{i=1}^{N} e_i\right] \tag{2.3}$$

Average cartographic error ranges from 0 to any number, where 0 is the best case with no error and higher the error is the worse the scenario is.

Relative Area Error

Another metric for statistical accuracy calculation is the *Relative Area Error*. For previously defined $m \times n$ table A, Keim et al. [76] defined this relative area error, e_i^{rel} for i polygon as follows.

$$e_i^{rel} = \frac{\left|A_i^{desired} - A_i^{actual}\right|}{A_i^{desired} + A_i^{actual}} \tag{2.4}$$

Now, for $N = m \times n$, the *average Relative Area Error* for a set of polygon, P would be as follows,

$$\widetilde{e_i^{rel}} = \frac{1}{N}\left[\sum_{i=1}^{N} e_i^{rel}\right] \tag{2.5}$$

The range of *average relative area error* is from 0 to 1, where 0 is the best case with no error and 1 is the worst case scenario.

Mean Quadratic Error

Keim et al. [76] also formulated another error metric from relative area error and named it as *Mean Quadratic Error*. It is actually the average of root squared relative area error. Again for $m \times n$ table A, Keim et al. [76]

calculated *Mean Quadratic Error*, I_{MQE} for a set of polygon, P as below,

$$I_{MQE} = \widetilde{e_i^{rel}} = \frac{1}{N}\sqrt{\sum_{i=1}^{N} {e_i}^2} \tag{2.6}$$

I_{MQE} lies in between from 0 to 1. Here, 0 is the best case with no error and 1 is the worst case scenario.

Root Mean Square Error

Root Mean Squared Error is another metric for accuracy calculation. It is the root value of the average of squared cartographic error. If $N = m \times n$, we can define *Root Mean Squared Error*, I_{RMSE} for a set of polygon, P as below using Equation 2.2,

$$I_{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} {e_i}^2}{N}} = \sqrt{\frac{\sum_{i=1}^{N}\left(\frac{\left|A_i^{desired} - A_i^{actual}\right|}{A_i^{desired}}\right)^2}{N}} \tag{2.7}$$

The range of I_{RMSE} is also from 0 to 1, where 0 is the best case with no error and 1 is the worst scenario.

2.3.2 Comprehensiveness Measurement metrics

The readability of the cartogram output plays a crucial role. If a person could not understand the informative insight of a visualization, there is no point in generating such visualization. We considered *average aspect ratio* (average *AR*) and *concave count* for the readability of the visualization. These metrics are particularly important for those applications that use image processing or image transformation followed by the cartographic distortion output.

Aspect Ratio

Aspect Ratio (AR) refers to the ratio between height and width of the bounding box of a region or polygon. *Aspect ratio* is often used as a measure for shape distortion [16, 136] and visual quality of cartogram generation techniques [13, 138].

The range for *average Aspect Ratio* is [0, 1], where 0 is the worst and 1 is the best. For a $m \times n$ table A, let $width_i$ and $height_i$ be the width and height of the bounding box of polygon i from a set of polygon P. Now, if the total number of polygon, $N = m \times n$, the *mean AR* is as below,

$$Mean\ AR = \frac{1}{N} \cdot \sum_{i=1}^{N}\left[\frac{min(width_i, height_i)}{max(width_i, height_i)}\right] \tag{2.8}$$

The range of *mean aspect ratio* is also from 0 to 1. Here, 1 is the best case where height and width are similar and 0 is the worst case.

Concave Count

The *concave count* is simply the number of concave quadrilaterals after tabular cartographic transformation. If a quadrilateral transforms into a convex one, it follows linear transformation for all point coordinates after applying *piecewise affine transformation.* However, the concave quadrilaterals do not follow this properly and also generate glitches on the output transformed image. Thus, the applications using *piecewise affine image transformation* should not accept any concave quadrilateral. Concave count would be any positive value starting from 0. Zero concave cell is the best case, whereas the higher the count worsen the scenario.

2.4 Miscellaneous

Concave and Convex Quadrilateral

A *quadrilateral* is a polygon with four edges. The line segment connecting non-adjacent vertices is called *diagonal.* A quadrilateral is called a convex quadrilateral if all internal angles are less than or equal to 180^o. Two diagonals of a convex quadrilateral usually intersect each other. If one of the internal angles of a convex quadrilateral becomes 180^o, it turns into a triangle. So, a triangle is a degenerate convex quadrilateral. On the other hand, a quadrilateral is concave if one of its internal angles is greater than 180^o. Another way to think is that the two diagonals of a concave quadrilateral do not intersect each other.

2.5 Summary

In this chapter, we described the background information for several topics. We defined different types of cartograms related to table cartogram. Afterward, we illustrated the *piecewise affine tranformation* technique which was used in several applications of large table cartograms. We also described several important performance metrics to calculate the statistical accuracy and the readability of the cartographic visualization. All the topics in this chapter are required to understand before proceeding to the following chapters.

3 Literature Review

Researchers and scholars has drawn their attention towards cartograms and different problems of cartograms since 18th century. These problems include different fields like cartography, computational geometry, computer graphics, and many others. Different techniques and approaches has been demonstrated to tackle those various complex problems and scenarios. Our primary focus is to explore different table cartograms for large tables having thousands of cells, along with their numerous applications. In this chapter, we describe the history of cartograms, different types and techniques of cartograms to tackle the problems, and table cartograms with their applications and problems. Finally, we describe different algorithmic techniques such that *Tutte's embedding*, *force directed layout*, and *quadtree*.

3.1 Related Works

3.1.1 Cartograms

History of Cartograms: Cartographic representation was used from an early age, even before it was named as 'cartogram'. Tobler [135] mentioned that the first reference to the term 'cartogram' was used when Emile Levasseur presented cartograms in an economic geography textbook dated back to 1870. Before that, cartogram-like representations had appeared in 19th-century atlases in the US. In 1837, William C. Woodbridge presented 'comparative charts' of North America, Europe, Africa, and South America in his 'Modern atlas on a new plan to accompany the system of Universal Geography' (See Figure 3.1(a)). In the same year, the size and population of the major empires and kingdoms were visualized in 'New and Improved School Atlas' by Jesse Olney (See Figure 3.1(b)). In 1897, 'Rand McNally World Atlas' published cartogram-like representations with two circles for each empire symbolizing the area and the population of the empires [108] (See Figure 3.1(c)).

Later in 20th century, according to Fabricant [94], German election results in 1903 were visualized with cartograms. William B. Bailey showed '1911 Apportionment Map of the United States', where the size of the states is proportional to their population [82] (See Figure 3.2). In 1934, the first formal definition of rectangular cartograms was given by Raisz [119]. In the 1979 Atlas of Canada and the World, population cartogram and oil production cartograms were published [114] (See Figure 3.3). In the 90s decade, cartograms became popular and frequently described in geography and cartography textbooks [19, 128].

Surveys: Guseyn-Zade and Tikunov [56] presented a short review on several methods for cartogram generation. They introduced several desired properties to construct a cartogram such as the reproducibility

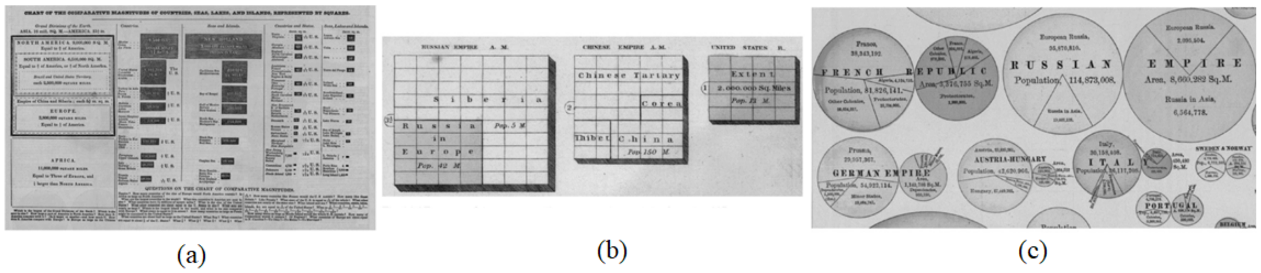


Figure 3.1: Cartogram-like representations from early ages [109, 1], (a) "Chart of the comparative magnitudes of countries" by Woodbridge in 1837, (b) figure from "New and Improved School Atlas" by Olney in 1837, (c) figure from "Rand McNally World Atlas" of 1897.

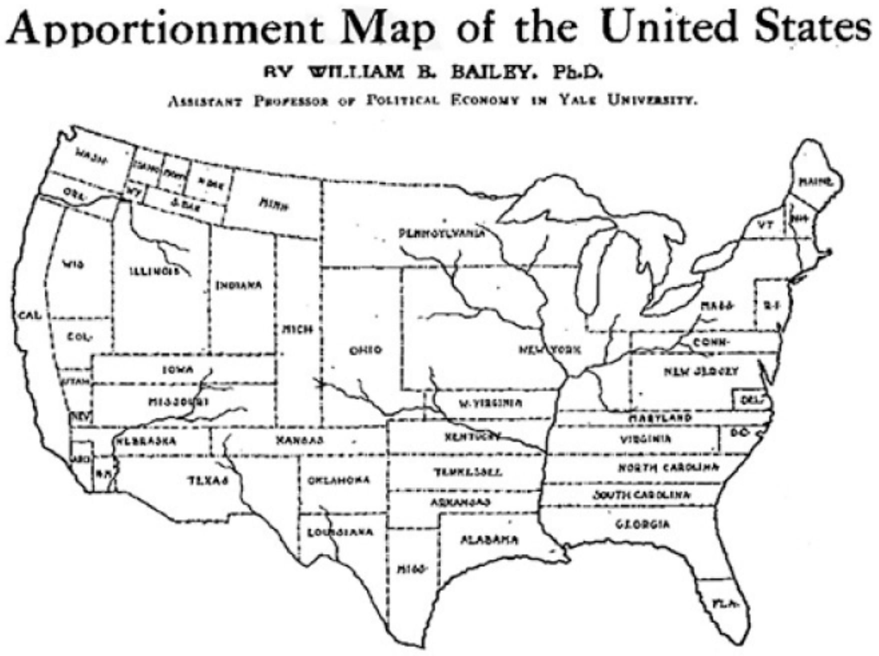


Figure 3.2: 1911 Apportionment map of the United States by William B. Bailey [1].

of the results and the recognizability of the regions by preserving original geographic shapes. Kocmoud [80] compiled another survey of cartogram generation techniques and discussed additional characteristics such as shape preservation, prevention of region overlapping, and trade-offs between area and shape accuracy. Tobler [135] compiled a comprehensive survey of cartograms, including different cartogram generation methods with their limitations. Tobler highlighted the necessity of cartogram evaluation measures mentioning three fundamental factors: (i) statistical accuracy, (ii) shape preservation, and (iii) computational efficiency. In 2016, Nusrat and Kobourov [109] conducted a detailed survey on cartograms and categorized all types of cartograms based on their design dimensions and task taxonomies. The surveys by Tobler [135] and by Nusrat and Kobourov [109] demonstrated historical progress in cartogram since an early age. Recently, Langton et al. [83] surveyed with a study design to measure how people interpret different cartograms of the same data to draw a conclusion. Hografer et al. [65] also conducted another survey on map-like visualization and

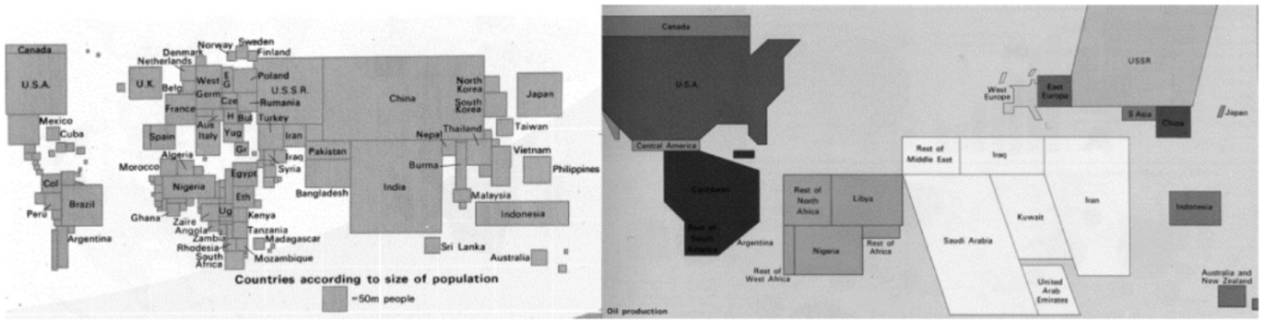


Figure 3.3: Cartograms from the 'Atlas of Canada and the World' (1979) showing (left) population, (right) oil production [108, 109].

provided a hierarchical classification of the existing techniques.

Cartographic Transformation: Cartographic representation of maps is a classic area of research. Cartographic representations can be divided into two major categories: contiguous and non-contiguous. In a contiguous area cartogram, the area of each region in the cartogram corresponds to a given measured value for that region, and all the regions together preserve the map topologies. Non-contiguous cartograms (e.g., circular [35, 29], rectangular [119, 129, 21] cartograms) do not require the map topologies to be preserved. Table cartograms can be thought of as a contiguous area cartograms.

Early algorithms were based on the metaphor that considers the original map as a rubber sheet, and defines forces on the points such that they move to realize the cartographic representation. A major challenge in such an approach is to ensure fast convergence, and preserve the shape of the regions on the map. Many algorithms [25, 36, 67] have been proposed to tackle these challenges, yet the slow convergence remained a problem.

The diffusion-based algorithm [51] is another popular method for drawing contiguous cartograms. This is inspired by the idea of diffusion in physics, where the points move from high density to the low-density regions. A diffusion-based algorithm often produces blob and spike like artifacts, and narrow corridors. Hence Cano et al. [23] proposed mosaic cartograms, where regions are represented as a set of regular tiles of the same size. The tiles help compare the regions via counting, better preserve the shape, but also have a cost of making winding region boundaries and tentacles like artifacts.

There exist many approaches beyond rubber sheet or diffusion based algorithms. For example, medial-axis based cartogram [77] that transforms the regions based on the medial axis of the map polygon, neural network based approach inspired by self-organizing map [61], etc.

Fast Computation: In 2013, Sun [132] presented a rubber-sheet inspired algorithm (Carto3F) that significantly improves the computational efficiency by using a quadtree structure, and eliminates the topological error by adding appropriate conditions while computing forces. In addition, the implementation of Carto3F exploits parallel computation. Recently, Gastner et al. [52] proposed a flow-based technique (**FastFlow**), inspired by the way particles of varying density into the water diffuse across the water over time that also leverages parallelism for fast cartogram computation.

Table Cartograms: Table cartograms [41] were proposed to visualize tabular data, where cells are restricted to be convex and to form a tiling of a rectangle. However, if we relax table cartogram constraints, then it is possible to use existing physics-inspired cartogram algorithms to create cartograms for tables. Winter [143] used the diffusion-based cartographic representation to visualize periodic table of chemical elements. However, this deforms the outer boundary of the table, as well as produces non-convex cells. Cartogram algorithms (in particular Carto3F [132] and flow-based approach [52]) often use an underlying regular grid (sometimes of size 1024×1024, e.g., Carto3F allows using a quadtree of depth 10) to transform the map, but the number of map regions makes a major difference. The existing approaches for cartographic representation focus on transforming at most a few hundred regions, while a 1024×1024 table may contain over a million cells.

Table cartogram is closely related to another well-studied problem called prescribed area drawing. Given a planar graph with positive face weights, a *prescribed area drawing* computes a planar drawing of the graph in $\mathbb{R}^2$ such that each interior face realizes a prescribed face area. In 1990, Ringel [120] introduced the prescribed area drawing problem. Since then a rich body of literature has focused on computing prescribed area drawings for various classes of planar graphs. Thus a *table cartogram* is a version of prescribed area drawing problem where the input graph is an $m \times n$ grid graph. We refer the reader to [55, 4] for the literature related to the prescribed area drawing.

Recent Works on Table Cartogram: Since the introduction of table cartogram, there have been various attempts to produce aesthetic cartograms based on optimization. Inoue and Li [71] showed an optimization-based approach for the construction of table cartograms by changing the bearing angles on the edges of the polygons. McNutt and Kindlmann [95] proposed another approach that instead of convexity constraints, puts simpler restriction such as to preserve the x and y axis ordering of the grid points. They showed that one can construct multiple table cartograms that have the same cartographic error but visually look very different. Recently, McNutt [96] has further examined table cartogram with the perspective of the data and tasks where table cartogram may be useful, and observed that table cartogram may be suited for various tasks such as understanding sorted order of cells or to find anomalies; especially when the table size is small.

3.1.2 Algorithmic Techniques

Our proposed table cartogram algorithm, **TCarto**, can simulate tables with thousands of cells. *Tutte's embedding* motivated us initially to design **TCarto**. Later, we adapted a *force-directed layout* graph model to devise our table cartogram algorithm. Parallel processing made the algorithm capable of simulating thousands of cells within reasonable processing time (5-7 minutes for 64×64 table in Figure 5.2(a)(TCarto)). We improved the task distribution technique for parallelization of **TCarto** with the concept of *quadtree.*

Tutte's embedding: Tutte's embedding or barycentric embedding is a classic algorithm in graph drawing and geometric graph theory. In 1963, Tutte showed that every simple 3-vertex-connected planar graph has a non-overlapping straight-line embedding such that the outer face is a convex polygon and interior vertex is at the barycenter of its' neighbors' positions [137]. This theorem motivated lots of research works, such as mesh processing methods [18], and force-directed graph drawing algorithms.

Force-directed layout: A force-directed graph drawing algorithm draws graphs aesthetically by clustering similar types of nodes. It is also called the spring-embedder model [53, 79, 127] or energy-based model [105, 106]. The basic idea is to define the model (node and edges) with some initial energy that converges to the minimum energy state by moving all nodes over a certain number of iterations. Non-adjacent nodes repeal each other, and adjacent nodes attract each other. The node movements become zero at the minimum energy state. This layout is used for both directed [47] and undirected [33, 46] graphs. Many researchers improved the efficiency [68] and aesthetical quality [34] of this algorithm over time. Many proposed some hybrid models with other well-known techniques such as genetic algorithm [63, 15, 78], space-filling method [73], stochastic sampling [100]. Parallelization is also applied to increase the scalability of this force-directed layout [134, 97, 149] to support thousands of nodes. GPU is also used to make this layout faster [141, 20]. Researchers also considered the nodes into multiple levels and proposed different multi-level algorithms [140, 57, 10]. The force-directed layout is used in different types of applications, such as large graph [49, 101, 90], cluster [38, 124, 130], network [93, 14, 60] visualizations, and many others.

Quadtree: A quadtree is a tree data structure with each internal node having precisely four children. Two-dimensional space is partitioned to subdivide recursively into four quadrants [42, 121, 122]. Several important surveys on quadtree or octrees [27, 6] provided different techniques and methods of the quadtree. Researchers conducted researches on making quadtree more efficient [131, 150, 32], and some focused on getting the optimal solutions [126, 85, 146]. Many researchers explored different applications of the quadtree, such as image processing [69, 102, 9], connected component labeling [123, 9], spatial queries [48, 151, 9], mesh generation [148, 111, 98, 9], and many others.

3.1.3 Parallel Processing

Parallel processing is the computation where many calculations or tasks are processed simultaneously. Initially, the tasks or instructions are distributed to several computation units such as central processing unit (CPU) [75, 113], graphic processing unit (GPU) [66, 17], and the results or outcomes of all the computations are accumulated and combined as the final output. Parallel processing can be classified into several classes such as multi-core computing, symmetric computing, distributed computing, cluster computing, and many others. The main advantage of such parallel processing is that large problems can be solved easily with a short processing time. After the invention of the Analytical Engine by Charles Babbage [87, 64], parallelism in numerical calculations was discussed first in 1958 [142, 117]. Later, Amdahl's law was devised to define the limit of speed-up due to parallelism in 1967 [142]. Since then, researchers have been relentlessly worked on improving parallelism.

Nowadays, parallel computations with central processing unit (CPU) [75, 113] and graphic processing unit

(GPU) [66, 17] are very prevalent and widely used to speed up computation in image processing [117, 37] and visualization [84, 139]. Many applications with grid like input (image or matrix) use column-based [31, 74] and quadtree-based [99, 86] parallelism. We used both techniques to parallelize our proposed algorithm **TCarto**.

3.2 Summary

In this chapter, we discussed the brief history of cartograms, several important surveys on cartograms, and different types of cartographic transformation. We also described various fast computational cartogram generation techniques and different table cartograms. We also mentioned all the recent works with table cartograms and their applications. Finally, we finished this chapter with a summarized discussion on different algorithmic techniques such as Tutte's embedding, force-directed layout, and quadtree.

4 Table Cartogram Algorithms

The primary focus of this thesis is to explore the table cartograms for a large table and adapt the existing algorithms. Evans et al. [40] designed a table cartogram algorithm and proved that it produces minimal (almost zero) error. We implemented the algorithm and referred it as **Baseline**. However, this algorithm produced some skinny triangles (See Figure 1.3) that made this algorithm impractical to our proposed applications. Then, Gastner et al. [52] designed and implemented **FastFlow**, one of the fastest known cartographic transformation in the literature. Since **FastFlow** was not developed to use for the table cartograms, we adapted this algorithm to run on large table cartograms. However, this algorithm generates concave cells (See Figure 1.6) for the data with a high number of local maxima and sudden value changes into neighboring cells. These concave cells might mislead specific applications of the table cartograms. Thus, we designed and implemented a new table cartogram generation technique, **TCarto** to simulate a large table without generating any concave cells. In this chapter, we illustrate the design of the **TCarto** and **Baseline** algorithm. Finally, we discuss the **FastFlow** algorithm and describe the details of our adaptation.

4.1 TCarto: An Optimization Based Algorithm

TCarto is different than the known optimization based algorithms [71, 95, 96] in two ways. First, it contains explicit convexity constraints (instead of considering an under-constrained version). Second, it leverages parallel computing to handle large tables. **TCarto** is inspired by the classic Tutte's approach for drawing planar graphs [137]. At each iteration of the Tutte's algorithm, every vertex moves towards the barycenter of its neighbors, and hence over time, the algorithm attempts to minimize a global energy function defined on the layout. In **TCarto**, every vertex except four at the corner moves towards the locally optimum point.

Let T be an $m \times n$ table. We normalize the cell weights $w_{i,j}$ to satisfy the following equality: $\sum W_{i,j} = m \times n$. We set the initial cartogram to be a regular $m \times n$ grid of area mn, where each cell area $A_{i,j}$ is one unit square. We can describe this with an undirected graph, $G = (V, E)$, where $V = V_{i,j}$ are the vertices ($i \in \{0, 1, ..., m\}$ and $j \in \{0, 1, ..., n\}$) and E are the edges connecting each vertex to its neighbouring vertices at its top, right, bottom and left side (Figure 4.1).

The total number of vertices is $(m + 1) \times (n + 1)$. Let, W be the weights or target values for all cells, and we normalize W. Assume the areas of the cells are $A_{i,j}$ = Area ($V_{i,j}$, $V_{i+1,j}$, $V_{i+1,j+1}$, $V_{i,j+1}$), where $i \in \{0, 1, ..., m\}$ and $j \in \{0, 1, ..., n\}$.

Initially, it is a regular ($m \times n$) equal-area grid, and the area for every cell individually is 1. We can deduct the total area of table T from Equation **??** as below,

$$\sum_{i,j} A_{i,j} = m \times n = \sum_{i,j} W_{i,j}. \tag{4.1}$$

This total area remains constant throughout the whole algorithm life-cycle. The vertices, $V_{i,j}$ changes their position based on local optimization so that all cells remain convex and,

$$\forall A_{i,j} \rightarrow \forall W_{i,j}. \tag{4.2}$$

We have used quadratic programming [7, 147] for this local optimization. This local optimization depends on the neighborhood cells' vertices, weights and areas. It is not biased on the non-neighboring attributes directly. So, this local optimization can not reach the solution for the global optimization in a single run. After several iterations of local optimization, it impacts non-neighboring cells also. The solution for the global optimization is that all $V_{i,j}$ move to certain positions so that all $A_{i,j}$ is equal to their corresponding $W_{i,j}$. The global optimization function can be formulated by the following Equation 4.3.

$$f(\psi) = \sum_{i,j} |W_{i,j} - A_{i,j}|^2, \tag{4.3}$$

where $i \in \{0, 1, ..., m-1\}$ and $j \in \{0, 1, ..., n-1\}$. The goal of the global optimization is to find the right combination ψ for the positions of all $V_{i,j}$ so that $f(\psi)$ becomes the minimum.

Three types of nodes

We can categorize the nodes into three types based on their movement. These three types are fixed nodes (red color), boundary nodes (orange color) and inner nodes (blue color) (See Figure 4.1).

Fixed Nodes (red color): Fixed nodes are the four corner nodes and their position is fixed throughout the whole life-cycle of the algorithm.

Boundary Nodes (orange color): Boundary nodes are the nodes residing on the boundary. These nodes can only move on top of the boundary. These nodes are two types - vertical boundary nodes (See Figure 4.2(a)-(b)) and horizontal boundary nodes (See Figure 4.2(c)-(d)).

In Figure 4.2(a), $V_{i,j}$ is a vertical boundary node that will move on top of vertical boundary line, $Line(V_{i,j-1}, V_{i,j+1})$. We denote by W and A as the weights and areas of the neighbouring cells, respectively. Assume that $y^{'}$ is the vertical displacement of $V_{i,j}$ towards $V_{i,j-1}$, and $V_{i,j}^{'}$ is the new position of $V_{i,j}$. If $A^{'}$ is the area of the triangle $\Delta V_{i,j} V_{i,j}^{'} V_{i+1,j}$, then

$$\frac{A_{i,j} + A^{'}}{A_{i,j-1} - A^{'}} = \frac{W_{i,j}}{W_{i,j-1}} \tag{4.4}$$

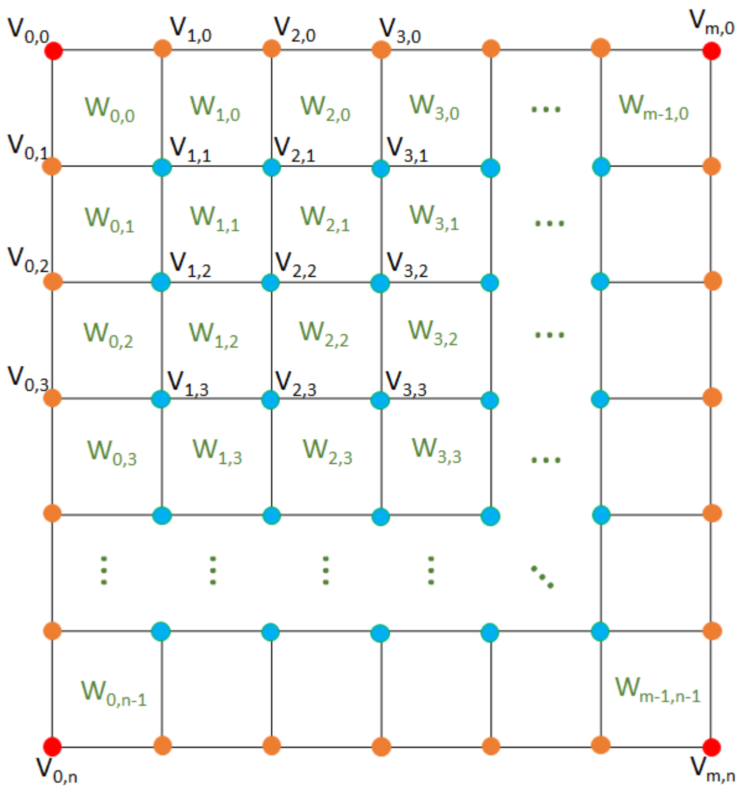


Figure 4.1: Initial $m \times n$ table T, where V, W and A are the vertices, weights and the areas of corresponding cells, respectively. There are three types of vertices - red, orange and blue vertices. Red (fixed) vertices are fixed, orange (boundary) vertices can move only towards boundary line and blue (inner) vertices can move anywhere inside so that each cell remains convex.

$$A^{'} = A_{i,j-1} - \frac{W_{i,j-1}.(A_{i,j} + A_{i,j-1})}{(W_{i,j} + W_{i,j-1})}. \tag{4.5}$$

Assume that $h^{'}$ the shortest path from $V_{i+1,j}$ to the vertical boundary line $Line(V_{i,j-1}, V_{i,j+1})$. We can calculate $y^{'}$ as follows:

$$y^{'} = \frac{2A^{'}}{h^{'}}. \tag{4.6}$$

If $y^{'}$ is positive, then $V_{i,j}$ will go upwards. It will go downwards for a negative value. Similarly, we can calculate horizontal displacement $x^{'}$ from Figure 4.2(c) as follows:

$$x^{'} = \frac{2A^{'}}{h^{'}}. \tag{4.7}$$

Due to the convexity constraint, boundary nodes can not move beyond certain region. F is the vertical and horizontal nodes in Figure 4.2(b) and (d), respectively. In both cases, F node should remain in between F_1F_2 line to keep $ABCF$ and $CDEF$ quadrilateral as convex. If F moves between AF_1, then $EDCF$ will become concave. $ABCF$ will become concave if it goes between EF_2.

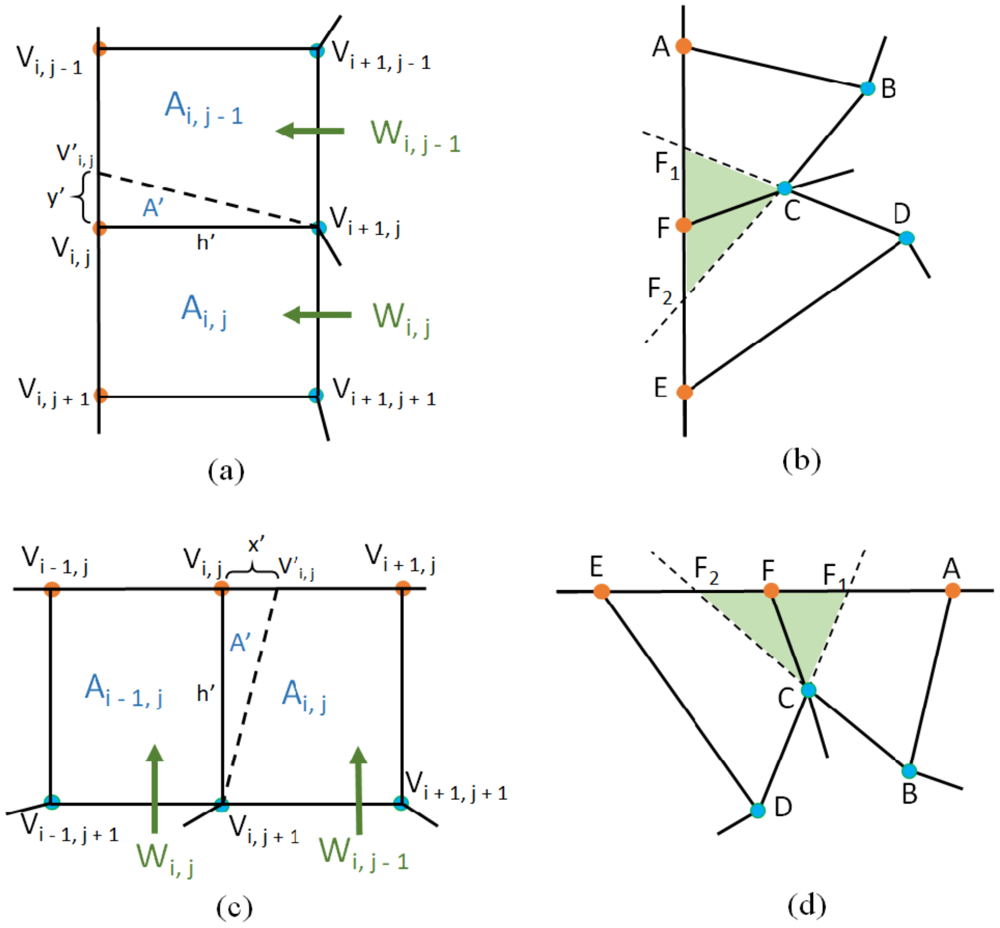


Figure 4.2: (a) Boundary node movement on vertical boundary line, (b) constraints for the movement of the vertical boundary node, F so that ABCF and CDEF cells remain convex. F can only move in between F_1 and F_2 along vertical boundary, (c) boundary node movement on horizontal boundary line, (d) constraints for the movement of the horizontal boundary node, F so that ABCF and CDEF cells remain convex. F can only move in between F_1 and F_2 along horizontal boundary.

Inner Nodes (blue color): All other nodes inside the boundary are inner nodes. We move these based on local optimization using quadratic programming. Let t be the current vertice to be moved (e.g., See Figure 4.3(c)), and consider the top left face $tpaq$ quadrangle of t. We now construct four constraints determined by the lines through pq, qs, sr, rp such that t must preserve its location relative to these lines even after the move. These four constraints alone cannot ensure that the four neighboring faces must remain convex after we move t. For example, moving t outside of the shaded region (but keeping it inside the quadrangle $pqsr$) would make one of the four faces non-convex. The reason is that for each face, the edges which are not incident to t also put constraints on where t can move. Hence, we need to add another set of eight constraints, two for each neighbor of t. Specifically, for a neighbor q, let qa and qb, where $t \notin \{a, b\}$, be the edges such that each lies on the boundary of one of the faces adjacent to the edge qt. Then the two constraints corresponding to q are determined by the lines corresponding to qa and qb. The twelve constraints together ensure that the neighboring faces of t remain convex even after the move. Since no other face is affected by the move of t, all the faces in the layout remain convex.

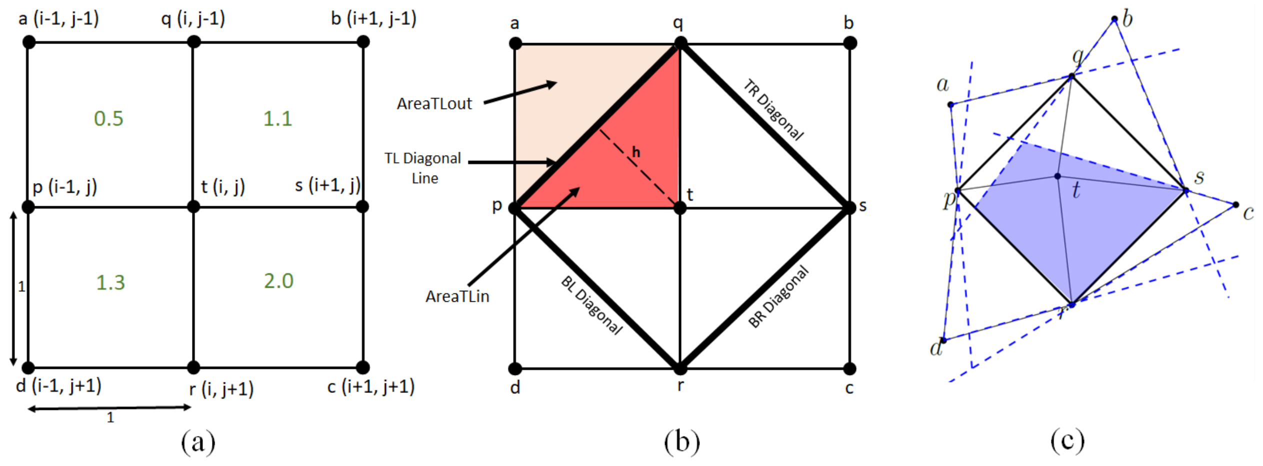


Figure 4.3: (a) Initial setup, (b) computation of the height, $h^{'}$, (c) a feasible region for moving a point, t.

We now describe the optimization function. Let $h_{t,1}, h_{t,2}, h_{t,3}, h_{t,4}$ be the perpendicular distances (heights) of t from the lines pq, qs, sr, pr, respectively. Note that these heights relate to the errors of the corresponding faces, and moving t would change these heights (e.g., See Figure 4.3(b)). We then define a set of required heights $h'_{t,i}$, where $1 \leq i \leq 4$. Here we only show how to compute $h'_{t,1}$. Denote by $W(tpaq)$ the target weight of a face $tpaq$, and let $A(tpaq)$ denote area of the same face. Assume that $\gamma = W(tpaq) - A(tpaq) - \Delta tpq$. If $\gamma > 0$, i.e., we need to add more area by moving t, then the required height is determined by the equation $\frac{1}{2}|pq|h'_{t,1} = \gamma + \Delta tpq$, where $|pq|$ is the Euclidean distance between p, q. Otherwise, $\gamma \leq 0$, i.e., we already have more than the required area. In this case we assume[1] that $h'_{t,1} = h_{t,1}$. We then minimize the error $\sum_{w \in V} \sum_{1 \leq i \leq 4} (h'_{w,i} - h_{w,i})^2$, where V is the set of grid nodes.

We can fit this minimization with quadratic programming mentioning all those twelve constraints. If $a_i x + b_i y = c_i$, where $1 \leq i \leq 4$ are the equation of four diagonal lines (pq, qs, sr, rp) and h'_i are the required heights respectively, we can define the quadratic minimization function as follows:

$$minimize \sum_{i=1}^{4} \left(\left(\frac{a_i^2 x^2}{{a_i}^2 + {b_i}^2} \right)^2 .x^2 + \left(\frac{b_i^2 y^2}{{a_i}^2 + {b_i}^2} \right)^2 .y^2 + \left(\frac{a_i.b_i}{\sqrt{{a_i}^2 + {b_i}^2}} \right) .xy + \left(\frac{a_i.(c_i - h'_i)}{\sqrt{{a_i}^2 + {b_i}^2}} \right) .x + \left(\frac{b_i.(c_i - h'_i)}{\sqrt{{a_i}^2 + {b_i}^2}} \right) .y \right) \tag{4.8}$$

subject to all twelve constraints that creates shaded region in Figure 4.3(c).

[1]Note that this case can also be handled based on the area excess, we did not consider that for simplicity.

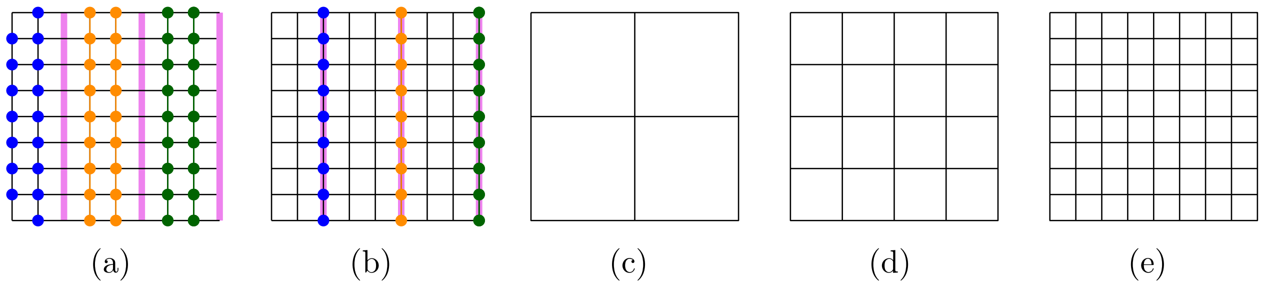


Figure 4.4: (a)–(b) Load distribution among different threads in two phases, (c)–(e) preprocessing for the DIV-CON approach.

Algorithm 1 TCarto Parallel

```
1: **Input:** A square size positive matrix $T$, iteration count $c$
2: Create a square grid and sort the vertices of the grid based on the area error of the neighboring cells
3: **for each** iteration $i$ from 0 to $c$ **do**
4:     Split the grid and assign column intervals to the threads
5:     Run each thread to move the vertices assigned to it in the precomputed order
6:     Join threads and accumulate the results
7:     Run each thread to move the vertices on the interleaved columns
8:     Join threads and accumulate the results into a layout $\Gamma$
9: **return** $\Gamma$
```

Parallel Computing

To take advantage of parallel computing, we partition the layout and distribute the load to different threads. At each iteration, we partition the columns into k regular intervals with one column gap in between (e.g., See Figure 4.4(a)). The iteration is completed in two phases. In the first phase, each thread moves their allocated points, and in the second phase the threads move the interleaved columns (e.g., See Figure 4.4(b). We will refer to this procedure as TCARTO PARALLEL. Algorithm 1 presents pseudocode of TCARTO PARALLEL.

The limitation of the TCARTO PARALLEL is that if the data density is very high in a particular interval, then it would take many iterations until the points move towards a low density region. To cope with this challenge, we take a top-down approach, as follows. Let the input be a $2^j \times 2^j$ grid. For each i from 0 to j, in the ith level we group the cells into a $2^i \times 2^i$ grid and run the procedure TCARTO PARALLEL for $j - \log(i) + 1$ iterations. Thus the major weight shift occurs early in the top levels and further refinement occurs at the bottom levels. We will refer to this approach as DIV-CON. Algorithm 2 presents a pseudocode for DIV-CON.

Implementation Details: We implemented our system in Python and used CVXOPT [8] for quadratic optimization. For parallel computing, a process-based parallelism package, multiprocessing [45], has been used for concurrent code execution. Movement of boundary points sometimes causes the loss of convexity of the grid due to the floating point error. We adjusted the computed point location to avoid non-convexity.

Algorithm 2 Div-Con

```
1: Input: A $2^j \times 2^j$ size positive matrix $T$, iteration count $c$
2: for each iteration $i$ from 0 to $j$ do
3:     create a square grid $G_i$ of size $2^i \times 2^i$
4:     for each iteration $i$ from 1 to $j$ do
5:         call TCarto Parallel for $G_i$ with $(j - \log(i) + 1)$ iterations
6:         Compute a layout $\Gamma$ by calling TCarto Parallel on $G_j$ with $c$ iterations
7: return $\Gamma$
```

We also ensured that the points do not overlap or lie on a non-incident edge by using a small threshold.

4.2 Baseline - Errorless Table Cartogram Algorithm

Evans et al. [40] proved that a table cartogram can always be achieved from a $m \times n$ table having positive weights. They construct a table cartogram with zero cartographic error and claim to generate such an errorless table cartogram every time with mathematical and geometric theory. They divide the input table in half based on the weights, make weighted triangles from those partitioned tables, finally compute cartograms for the triangles using the similar concept of barycentric coordinates. However, the authors mentioned that the generated output may not be visually pleasing and require better heuristics for aesthetically improvement. The aspect ratios of the width and height for the cells become high because of their triangle based partitioning process.

Let A be an $m \times n$ table of non-negative weights $A_{i,j}$. Let $S = \sum_{i,j} A_{i,j}$ and S_i be the sum of weights in row, i, i.e., $S_i = \sum_{1 \leq j \leq n} A_{i,j}$. Assume, $S > 4$ and R be the rectangle with height=2 and width=$\frac{S}{2}$ having four corners as $(0,0), (\frac{S}{2}, 0), (\frac{S}{2}, 2), (0, 2)$ (See Figure 4.5). We assume such height and width for an easy explanation of the algorithm; however, we set height= $\sqrt{S}$ and width= $\sqrt{S}$ in the actual implementation.

There are four major steps for this algorithm.

(a) Partition the input table and define two partitioned tables with roughly equal sum.

(b) Compute the polygonal zig-zag line Z in the rectangle R to partition it into triangles.

(c) Compute final cartogram with the idea of barycentric coordinates by splitting the triangles.

At the initial step, we need to find the splitting row k for which the cumulative row weight from top becomes larger than the half of total weight S. Then, we split A into two partitioned tables, A^t and A^b. The splitting factor λ is calculated λ from the equations (See equation 4.9 and 4.10) below.

$$\sum_{1 \leq i \leq k-1} S_i + \lambda S_k = \frac{S}{2} \tag{4.9}$$

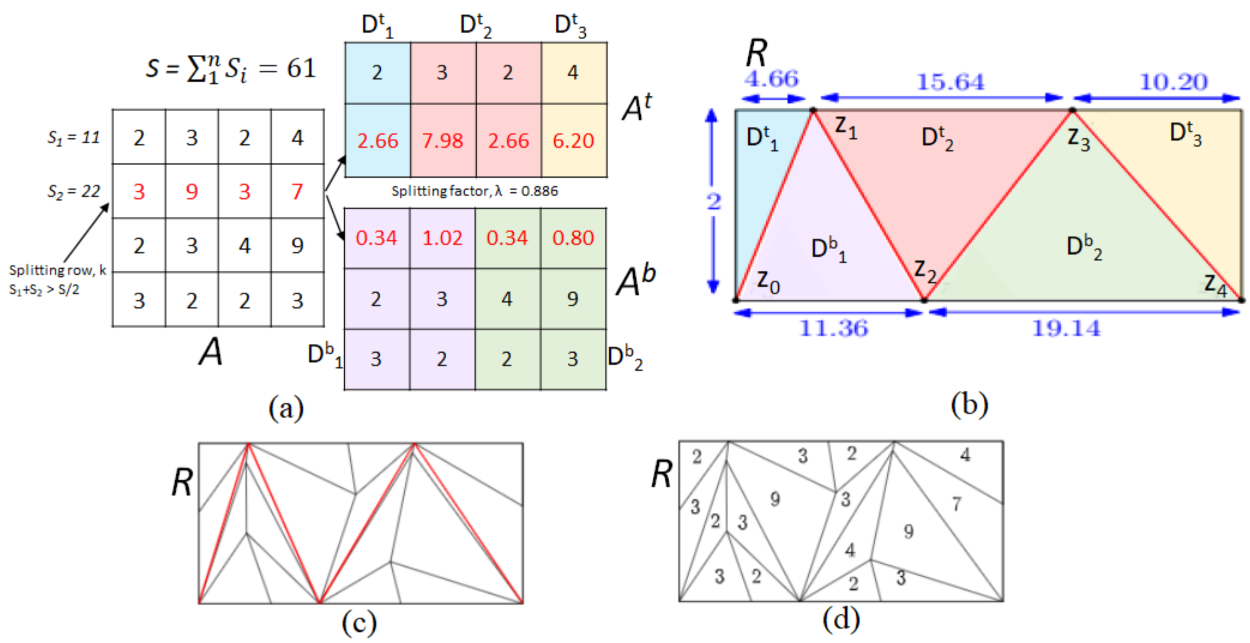


Figure 4.5: (a) Demonstration of Input table, a partitioned tables, A^t and A^b where total row, n=4, and splitting row, k =2 and splitting factor, λ = 0.886 and splitting row, (b) computation of zig-zag path, Z, (c) the subdivision of triangles, where Z is red color, (d) the final cartogram output.

$$\sum_{k+1 \le i \le n} S_i + (1-\lambda)S_k = \frac{S}{2} \tag{4.10}$$

We calculate the new weights for the split rows of A^t and A^b tables. Let D^t_p and D^b_q are two-column sums, where $p = 1, 2, ..., \frac{n}{2} + 1$, and $q = 1, 2, ..., \frac{n}{2}$. Here, D^t_1 and D^t_p have only single column. It is shown with blue and orange colors in Figure 4.5(a).

At the second step, we compute the zig-zag $Z = z_0, z_1, ..., z_n$ inside the rectangle R in such a way so that each triangle acquires the same area of their corresponding two-column sums, D^t_p and D^b_q (Shown in Figure 4.5). The first triangle has an area equal to the left column of A^t, and each subsequent triangle has an area equivalent to the next 2-columns. The last triangle may have one or two columns. The zig-zag Z always ends at one of the two rightmost corners.

At the last step, we split these triangles into subdivisional triangles so that all those smaller triangles represent the face of each cell of the input table A, with an area equal to their corresponding weights. We use barycentric coordination to calculate the smaller triangles.

We can consider $\Delta z_0 z_1 z_2$ to split into smaller triangles (See Figure 4.6). The weight of the top-left cell is assigned to $\Delta z_0 p z_1$ and top-right cell to $\Delta z_1 p z_2$. The sum of the rest is assigned into $\Delta z_0 p z_2$. Since we know the distance between z_0 and z_2, we can calculate the height, y of $\Delta z_0 p z_2$. After that, we consider a line $l_0 l_1$ parallel to $z_0 z_2$ line and point p would be on top of that line anywhere between x_{min} and x_{max}.

To find p, we do a binary search as follows. We first chose the position of point p randomly on top of line

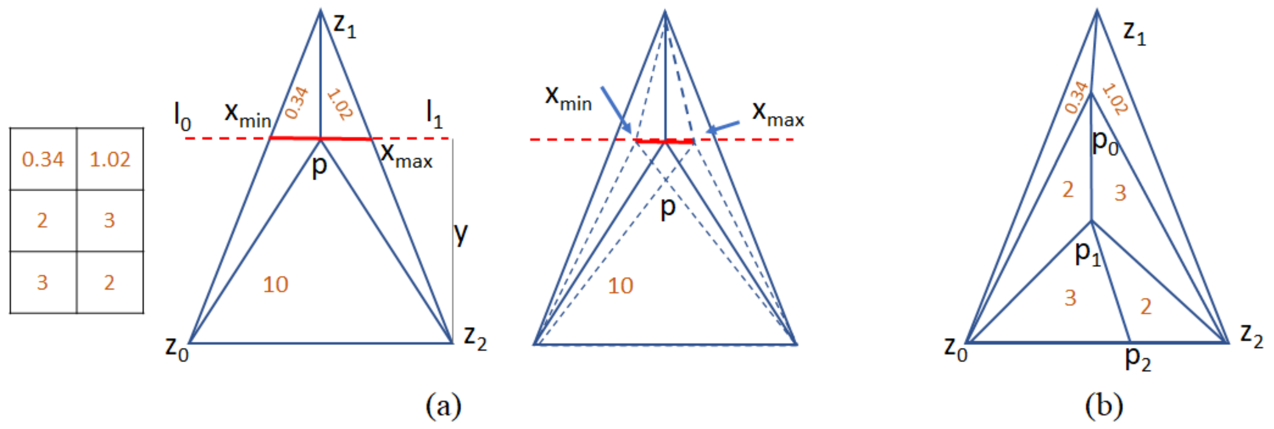


Figure 4.6: (a) Barycentric coordinates approach to subdivision the triangle, $\Delta z_0 z_1 z_2$. To make it readable and easier to understand, we have distorted the regions by pushing p point towards the bottom, (b) final output.

$l_0 l_1$ in between x_{min} and x_{max}. If $\Delta z_1 p z_0 > 0.34$, that means our point p is too much to the right. To make the search on the left, we set x_{max} at p position. If $\Delta z_1 p z_0$ is smaller than 0.34, we set x_{min} at p position. In this way, we narrow down the search region and chose point p again. The iteration ends when p position generates error within reasonable threshold.

We calculate the points p_0, p_1, p_2 at Figure 4.6(b) using a similar approach. After we remove the red zig-zag Z from Figure 4.5(c), we obtain the final cartogram output as in Figure 4.5(d).

4.3 FastFlow - Fast Flow-based Density-Equalizing Algorithm

The flow-based cartogram, introduced by Gastner et al. [52], is inspired by the density-equalization technique. This *FastFlow* algorithm is an all-coordinate cartographic transformation technique with linear equalization. Since this algorithm was not developed originally for the table cartogram technique, it does not maintain a fixed rectangular boundary and might generate empty spaces among the outer boundary and regions. Table cartogram and image-based applications require a fixed rectangular boundary and do not allow empty space inside the rectangular boundary. These adaptations are required to use it as a table cartogram technique and for image-based applications.

In our adaptation, *FastFlow* takes the regular grid table as input (See Figure 4.7(a)) where each cell is water filled. Initially, some particles or populations are placed on each region such that the density of each region is proportional to its corresponding weights (See Figure 4.7(b)). Then, the populations or particles diffuse across the entire outer-rectangular region. The populations or particles flow from higher density regions to lower density regions. Thus, higher density regions expand and lose some density, whereas lower density regions shrink to gain additional density. When the densities of all the regions are equal, the flow of populations or particles stops (See Figure 4.7(c)). Then, the final distorted regions are calculated by the boundary particles of each region, and the final output is generated (See Figure 4.7(d)).

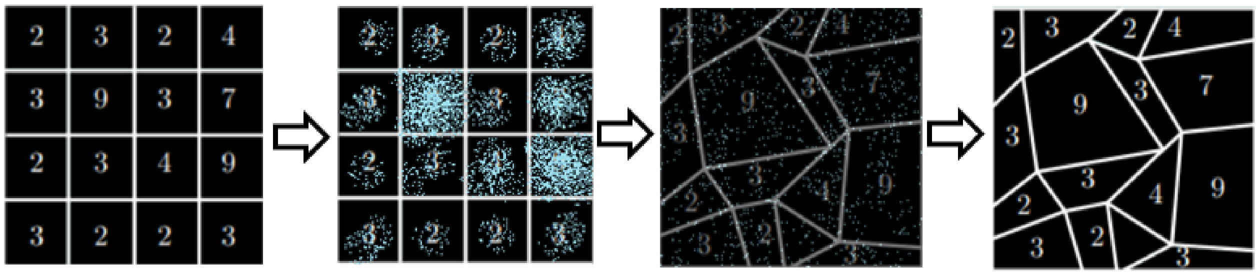


Figure 4.7: (a) Input table, (b) initial setup for FastFlow by putting particles into water-filled rectangular areas such that the density of each region is proportional to its corresponding weight, (c) equal density for all the regions, (d) the final output.

Formally, if $\rho(x, y)$ is the population density function of a small rectangular area with corners $(x \pm dx/2, y \pm dy/2)$, this rectangle contains the population $\rho(x, y)\, dx\, dy$. This rectangular area projects into a quadrilateral $\mathbb{T} = (T_x, T_y)$ in such a way that $\rho(x, y)\, dx\, dy = \overline{\rho}\, dT_x\, dT_y$, where $\overline{\rho}$ is the spatially averaged density to preserve the total map area. This transformation $\mathbb{T}$ is the density-equalizing projection. If $dx \to 0$, $dy \to 0$ and $\mathbb{T}$ is differentiable, we can get the prescribed Jacobian equation [30, 11],

$$\frac{\delta T_x}{\delta x}\frac{\delta T_y}{\delta y} - \frac{\delta T_x}{\delta y}\frac{\delta T_y}{\delta x} = \frac{\rho(x, y)}{\overline{\rho}} \tag{4.11}$$

In flow-based cartogram, the population density ρ is defined with not only position $r = (x, y)$ but also time t. This density must approach to its mean over time: $lim_{t\to\infty}\, \rho(x, y, t) = \overline{\rho}$ for all x and y. The points must flow so that the initial differences in their density become equal over time. This alone can not define transformation $\mathbb{T}$. We also need to know the velocity $v(x, y, t)$ with which (x, y) point is dragged at time t. This v satisfies the continuity equation,

$$\frac{\delta\rho}{\delta t} + \nabla.(\rho v) = 0 \tag{4.12}$$

If $v(x, y, t)$ is known for all x, y and t, we can calculate the position $r(t)$,

$$r(t) = r(0) + \int_0^t v(r(t'), t')\, dt' \tag{4.13}$$

The projection $\mathbb{T}$ will shift $r(0)$ to $lim_{t\to\infty}\, r(t)$. By integrating Fick's law, $v = -D(\nabla\rho)/\rho$ (,where D is the diffusion co-efficient) into equation 4.12, we can get $\delta\rho/\delta t = D\nabla^2\rho$. This new equation controls the evolution of ρ. If this new equation is replaced by a linear density equalization toward the mean,

$$\rho(x, y, t) = \begin{cases} (1-t)\,\rho_0\,(x, y) + t\,\overline{\rho} & \text{if } t \le 1 \\ \overline{\rho} & \text{if } t > 1 \end{cases} \tag{4.14}$$

A velocity field v exists for Eq. 4.13 such that the resulting transformation $\mathbb{T}$ satisfies Eq. 4.11 [30] that has been calculated using sine and cosine Fourier transformation.

After an affine transformation, all the coordinates are mapped inside a rectangular box with bounding coordinates $x_{min} = 0$, $y_{min} = 0$, $x_{max} = L_x$ and $y_{max} = L_y$. There is no flow at the edges of this bounding box. We compute velocity $v(r, t)$ for each grid point r and move the population with that velocity v. A predictor-corrector method is used to adapt the time step dynamically for next iteration. The iteration stops when all regions own the spatially averaged density $\overline{\rho}$.

Adaptation for table cartogram: We adapted this flow-based cartographic transformation technique to simulate large tables with thousands of cells. Since this algorithm is not developed for table cartograms, it expands or squeezes the outer boundary based on the needs in general. To make it perform similar to a table cartogram, we fix the outer boundary that no region can go beyond the outer boundary. We also ensure that there would be no empty space in between the outer boundary and the regions inside. We do not implement any restrictions for the concave quadrilateral because it takes a reasonable amount of time to verify this cells' convexity. One of the key features of this algorithm is faster processing speed, and we do not want to compromise its speed by applying convexity restriction. Thus, it might generate concave cells.

Later, when this algorithm runs the data table with high spikes in values and many local maxima, this algorithm is stuck within the predictor-corrector method to update the time step for the next iteration. It divides the time step into smaller and smaller on each iteration and eventually freezes the algorithm. So, we relaxed the rules of decreasing time steps and tweaked time step calculation to get out from the frozen state. After this modification, it can run a large table without freezing but generates overlapping cells followed by concave cells.

4.4 Summary

In this chapter, we discussed the implementation of the **Baseline** algorithm initially, and then moved to the next algorithm because of the skinny cells (See Figure 1.3). Then, we tried to adapt **FastFlow** algorithm, but it generated concave and overlapping cells for a large table with high spike data (See Figure 1.4). Later, we designed a new table cartogram, **TCarto** that could run a large table cartogram without generating any concave cells. We demonstrated the design and implementation of **TCarto** and **Baseline**. At the end, we explained the **FastFlow** algorithm along with the augmented part to make it fit for the table cartogram.

5 Applications of Table Cartograms

The first two research questions are to explore applications of large table cartograms beyond visualizing tabular data and the importance of convexity of each transformed cell in such applications. In this chapter, we propose several potential applications that use the underlying mechanism of a large table cartogram (i.e., positive evidence towards RQ_{A1}). We demonstrate how a table cartogram can be used to reveal spatial relation between a pair of variables, to understand clusters better in a scatter plot, and for different visual effects in images such as mosaic effect, expanding light illumination. For each application, we examine and discuss whether the convexity is crucial or not (i.e., RQ_{A2}). In the end, we describe how an angle constraint can be applied to produce better aesthetically pleasing visualization.

5.1 Infographics to Reveal Spatial Relation

Table cartograms for large metrics reveal the spatial correlation between pairs of geospatial variables. One variable is used for the input weights of the table cartogram and another to generate the initial contour plot. Then, the table cartogram algorithm transforms the initial regular grid to distorted grid output based on the weights. Afterward, image warping with *piece-wise affine transformation* [54, 115] is used on top of the contour plot based on the initial grid cells and the distorted grid output provided by the table cartograms. The expansion or shrinkage of the contour plot at the final output image reveals the spatial relation between those two variables.

In Figure 5.1, the two matrics representing two geospatial variables A and B. Let *variable A* be the input weight for the table cartogram, and *Contour (variable B)* be the contour plot of B. To calculate the input weight from variable A, we put the grid on top of the expected region, and then consider the average of all the values of variable A within a grid cell to calculate the input weight for that grid cell. We computed output for the table cartogram (i.e., $TCarto$, $Baseline$, and $FastFlow$) based on the input weights, *variable A*. Image warping took three inputs: *Contour (Variable B)* as the input image, the initial grid position as source nodes, and the distorted output grid generated by table cartogram algorithm as destination nodes. Image warping with piecewise affine transformation [58] generated the resulting output image that revealed the potential spatial relation between A and B.

Figure 5.2(first column) illustrates how a contour plot of ALBEDO (the reflected solar energy by the surface) is transformed based on the table cartogram for soil liquid water (SH2O) values. The blue color in the contour plot indicates a low value (lakes and ocean) and yellow indicates a high value. Since the

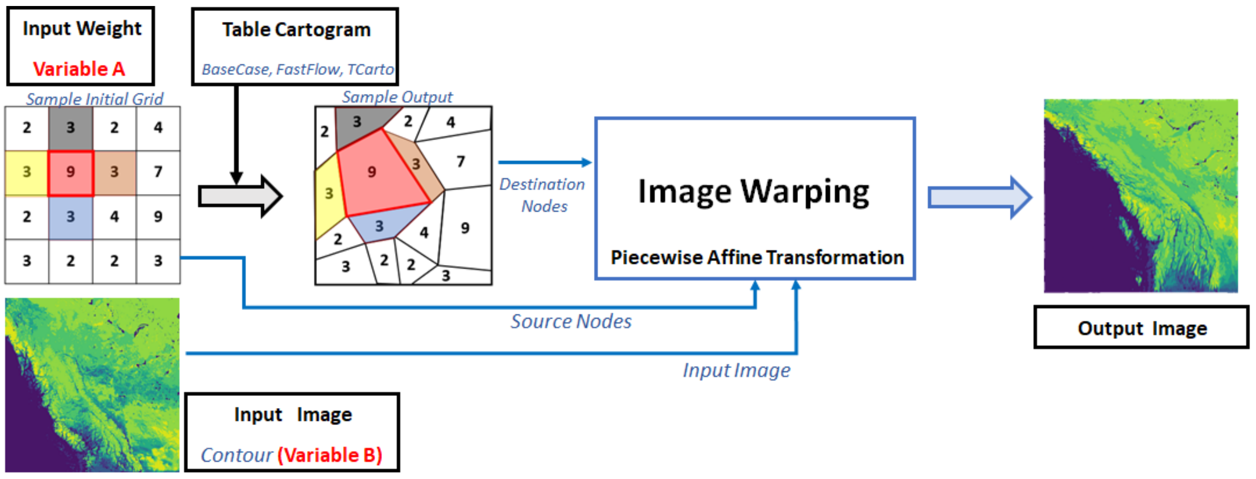


Figure 5.1: Applying table cartogram algorithm and image warping to reveal spatial relation between two variables, A and B. Here, we used soil liquid water (SH2O) as variable A and ALBEDO (the reflected solar energy by the surface) as variable B.

transformation grows the ocean, one can observe that SH2O is high in ocean (which is a low ALBEDO area).

Figure 5.2 shows the cartograms obtained using **TCarto** and **FastFlow** for various combinations of ALBEDO (Solar Energy Reflectance), TSK (Surface Skin Temperature), PBLH (Planetary Boundary Layer Height), SH2O (Soil Liquid Water), and EMISS (Surface Emissivity). In the contour plot, blue and yellow regions are the lowest and highest values consecutively. The expansion of yellow (high value) region or shrinkage of blue (low value) region indicates potential positive relation. For example, see Figure 5.2(c) for shrinkage of blue. We can infer a negative relation if the opposite happens, e.g., see Figure 5.2(a), (b) and (d) for expansion of blue. In Figure 5.2(a), SH2O as weight transforms the contour plot and expands the low valued regions of ALBEDO. It indicates that ALBEDO and SH2O are negatively correlated. From Figure 5.2(b-d), we can say that TSK and ALBEDO are negatively, EMIS and PBLH are positively, and ALBEDO and SH2O are negatively correlated. We excludes the output images for *Baseline* from Figure 5.2 on purpose since the grid output of *Baseline* generates lots of skinny cells and the output image also lacks the accurate neighborhood adjacencies (See Figure 5.3).

Discussion

Both **TCarto** and **FastFlow** showed similar transformation in regions and revealed potential relation between geospatial variables (RQ_{A1}). The cell convexity did not appear to be a crucial phenomenon since the grid size is large (RQ_{A2}). However, a close inspection of the grid showed that there exist major differences that were not readily visible in the images, but in the transformed grid. Hence overlaying the grid on the image or placing them side-by-side might help guide the interpretation when creating infographic pictures. Note that two cartograms might look very different even when their cartographic error was small (See Table 6.1). This had also been observed by McNutt [96], but for small tables. Hence such cartogram infographics were mostly useful to spark excitement among the viewers, or to covey a major concept.

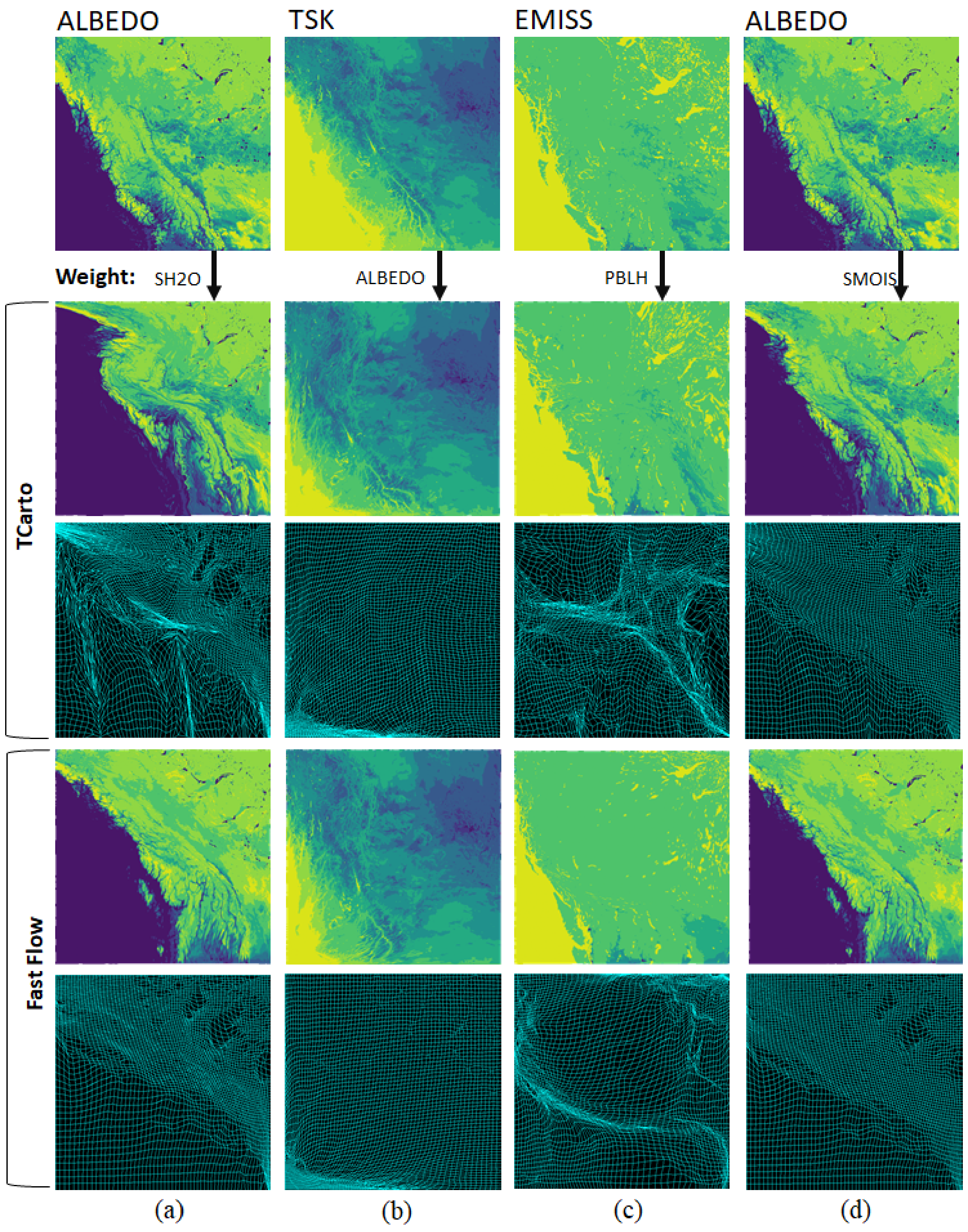


Figure 5.2: The first row shows the contour plots of different variables over western Canada (blue and yellow are low and high values). Then we have **TCarto** and **FastFlow** outputs with both the transformed images and the cartograms.

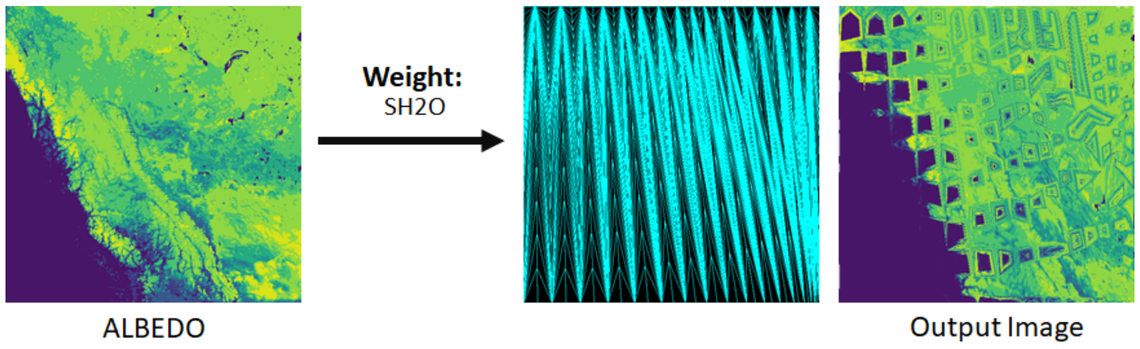


Figure 5.3: The output image for *Baseline* algorithm where ALBEDO is the contour and SH2O is the weights.

Both **FastFlow** and **TCarto** are suitable for the application to reveal the geospatial correlation between a couple of variables and convexity does not appear to be crucial for this application (RQ_{A1}, RQ_{A2}).

5.2 Different visual effects in images

Large table cartograms can potentially be used to create visual effects in images. Table cartograms expand the cells with higher weights and shrink the cells with lower weights. Because of this characteristic, large table cartograms can assist in increasing or decreasing the lightness of an image and generate different effects, such as the mosaic effect in an image. We described two such applications: increasing the light illumination and generating mosaic effect.

5.2.1 Increasing Light Illumination

A large table cartogram can embellish the illumination on the higher lightness regions and dim the lower lightness regions. To increase the illumination of light sources in an image (See Figure 5.4), we divided the input image into $m \times n$ equal area grid and calculated the weights as the average value of the lightness channel of HSL (Hue, Saturation, and Lightness) for each pixel within a single grid cell. Then, the large table cartogram algorithm transformed the grid cells based on those weights. Afterward, we generated a masked image (black and white) on the table cartogram output based on the weights and then applied *Gaussian Filtering* (Gaussian Blur with radius 10) to smooth the sharp edges. Finally, we increased the *lightness channel* of input image from 0% to a maximum of 25% based on the distorted cells and their corresponding weights to generate the final output image.

Figure 5.5 illustrates examples where the lightness channel of HSL (Hue, Saturation and Lightness) has been transformed using table cartogram to expand the light illumination from the light sources. This has been

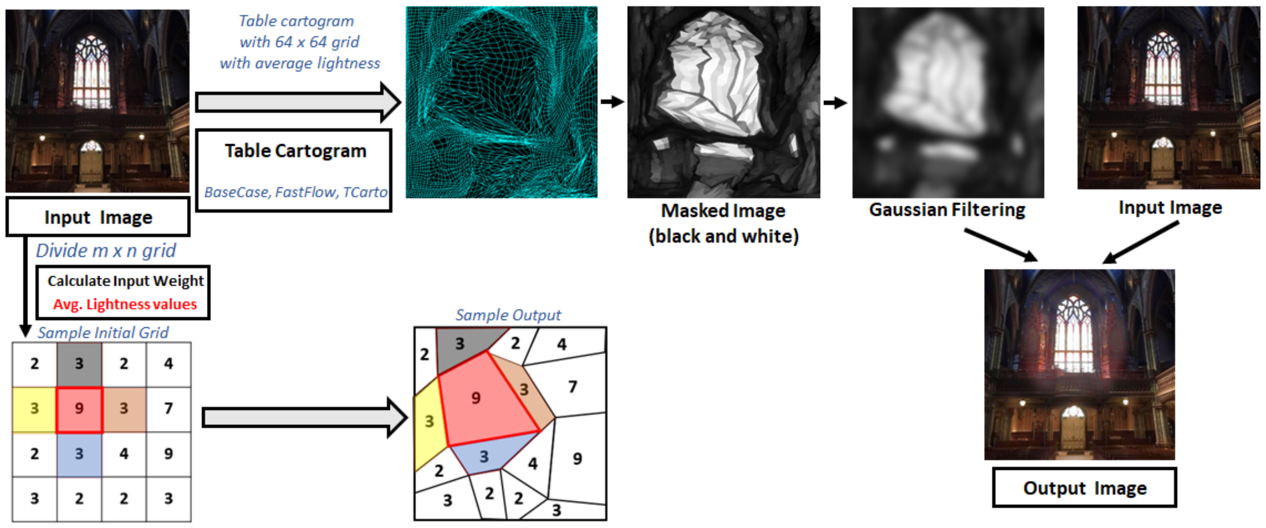


Figure 5.4: Increasing light illumination using table cartogram algorithm, image masking and image filtering.

achieved by overlaying a 64×64 grid on the image and then creating a weighted table T by accumulating the lightness values for each grid cell. Next the lightness channel has been adapted based on the table cartogram of T, which enlarges the brighter regions and compresses the darker areas.

5.2.2 Mosaic Effect

Another visual effect that we can create using large table cartograms is similar to mosaic arts. A mosaic art generates a picture using small stones or glass fragments. For the mosaic art effect (See Figure 5.6), we generated weights for $m \times n$ grid randomly and applied a table cartogram algorithm to deform the initial equal-area grid. We applied image warping with *piece-wise affine transformation* [115, 144] on the input image with the initial grid as source nodes and the table cartogram output grid as destination nodes. Then, we applied cell borders on top of the generated output to produce the final output. We can also apply different artistic effects or filters on the lines of the distorted grid or the transformed image programatically or using any photo-editor software (See Figure 5.7).

In Figure 5.8, we produce a 64×64 table T with random weights to generate the mosaic image effect. We then transform the image based on the table cartogram of T and then overlay the grid to create the mosaic effect.

Discussion

Both **TCarto** and **FastFlow** were able to create some mosaic effect (RQ_{A1}), where the cell sizes in the **FastFlow** cartogram appear to be more uniform compared to that of **TCarto**. While we only focused on potential application, it would be interesting to investigate whether one is more artistic than the other. Since the weights were chosen at random in a small interval, the cells produced were convex, with **FastFlow** cells

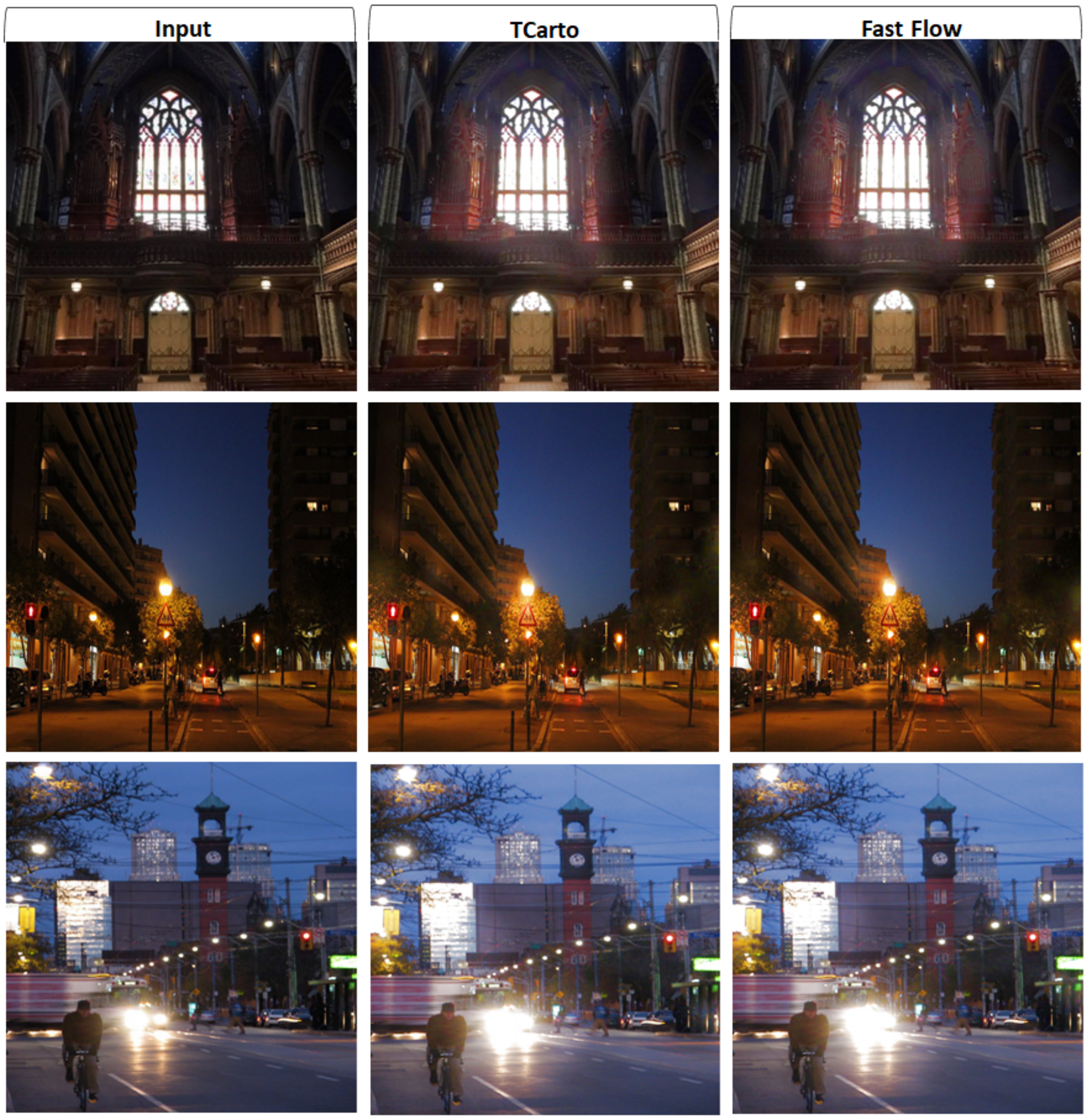


Figure 5.5: Table cartograms to expand light illumination using *64* × *64* grids, (left) input images, (middle) generated output using **TCarto** algorithm, (right) **FastFlow** algorithm.

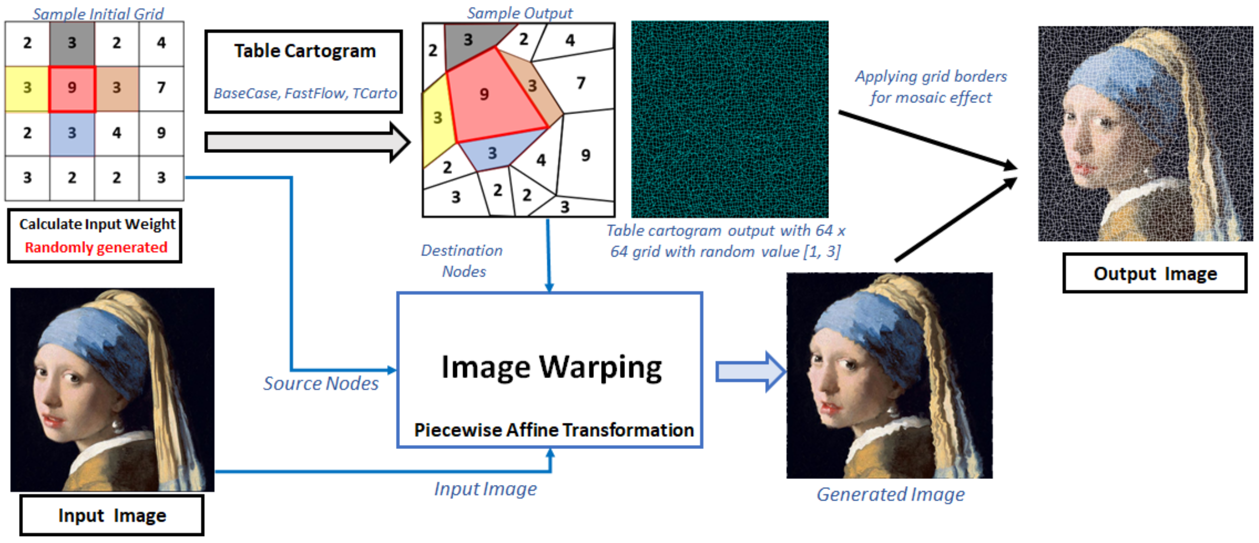


Figure 5.6: Mosaic effect generation using table cartogram algorithm, image masking and image filtering.

having better average aspect ratio (RQ_{A2}).

FastFlow outputs are slightly more uniform than **TCarto**, but both are acceptable for this type of applications and convexity is not crucial here (RQ_{A1}, RQ_{A2}).

5.3 Understanding Clusters in a Scatter-plot

Detecting clusters and density estimation of a highly dense scatter plot is always challenging due to the overlapping of data points. In 2019, Raidou et al. [118] proposed *Pixel-Relaxed Scatter Plot* as an addition to the traditional scatter-plot to detect the clusters with their corresponding densities compromising the cluster's actual positions (See Figure 5.9).

Large table cartogram may also assist in detecting clusters and interpreting their density, maintaining their coordinated actual positions on the scatter plot. Initially, we divided the scatter plot into a regular equal-area grid and calculated the weights for each cell as the number of data points residing on that cell (See Figure 5.10). We set a minimum positive value for the cells having no data point. Then, the table cartogram algorithm generated output based on the weights, where dense clusters expanded, and the other clusters shrank. We developed a mask image by coloring each cell with a sequential grey-scale color scheme based on the weights where black is the lowest and white is the highest. We applied the *Gaussian Filtering* to smooth the edges. Finally, we created a contour plot from the filtered mask image.

For example, Figure 5.11 column (a) illustrates three input scatter plots, each containing four clusters with 500 (blue), 200 (orange), 120 (red) and 120 (green) points. The density and sample number remain constant, but the position is changed to generate several scenarios such that partial overlapping (top row),

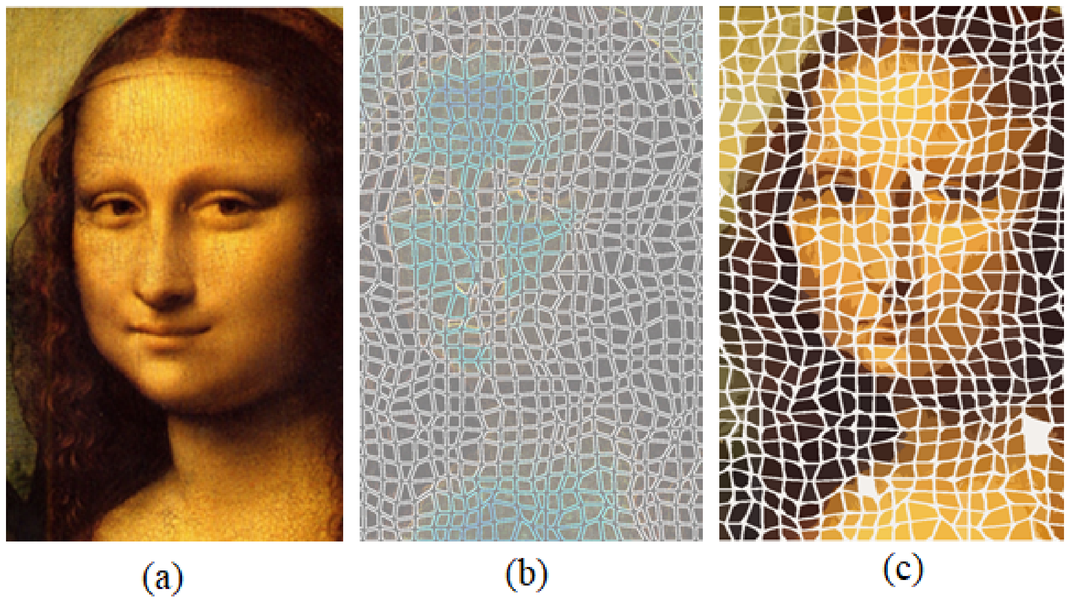


Figure 5.7: (a) Input image, (b) artistic effect on the lines of the grid transformed by **TCarto**, (c) artistic effect on the transformed image by **TCarto** using *Cutout* artistic effect in 'Microsoft PowerPoint 365'.

no overlapping (middle row) and all four overlapping (bottom row). The scatter plots has transformed into weighted tables by overlaying a 64×64 grid where the weight of each cell corresponds to the number of points in it. A table cartogram then expands the dense clusters which has been visualized using a contour plot.

While comparing the contour plot outputs by *TCarto* and *FastFlow* in Figure 5.11, the *TCarto* outputs seem expanding either vertical or horizontal way rather than growing equally to all directions. A potential reason could be the *CPU* distribution among the nodes using *TCarto DivCon* algorithm (See Algorithm 2 and Figure 4.4(c-e)). The weights are distributed depending on the *quadtree* based top-down approach. Then, all the inner nodes of those miniature quadtrees are processed in *TCarto Parallel* approach (See Algorithm 1). In contrast, the boundary nodes of those miniature quadtrees can only move to their vertical or horizontal boundaries that restrict the overall weights' distribution equally to all the directions. However, a more uniformly shaped load distribution may reduce this tendency. On the other hand, *FastFlow* outputs seem to spread along all the directions equally because of the diffusivity of the medium.

Discussion

Both **TCarto** and **FastFlow** were able to reveal the differences in the relative densities of the clusters, which was hard to depict in the original scatterplot. They even showed some detailed structures within the largest cluster. The **FastFlow** output looked more symmetric (circular) compared to that of **TCarto**.

Both **FastFlow** and **TCarto** are acceptable for this type of applications of cluster and density detection of a scatter-plot and cell convexity does not appear to be crucial here (RQ_{A1}, RQ_{A2}).

5.4 Tabular Data with Angle Constraint

Here we examine large table cartograms for general tabular data. This is a challenging case because a general table may contain many sharp local maxima and minima compared to the ones we observed in image applications or weather dataset [88]. In addition, the values between adjacent cells can be drastically different. In such scenario, the available implementation of **FastFlow** performed poorly, i.e., it generated many concave cells and also failed to converge.

Since the perception of rows and columns are important in interpreting a table, we augmented **TCarto DivCon** with angle constraints so that the corner of each cell was above a given angle threshold. In Figure 5.12, t is the previous position of a node, and t' is the optimized position of t calculated by quadratic programming. θ_1, θ_2, θ_3 and θ_4 are the angles of $pt'q$, $rt'p$, $rt's$ and $st'q$, respectively. Then, we check whether all these angles are higher than the angle threshold. If it satisfies, t' is the new position for t node. Otherwise, we calculate $t1$, which is 5% away from t' towards tt' line and check all the angles for this newly calculated $t1$ position. It continues until it finds a suitable position satisfying angle threshold, or the newly calculated node is a minimum distance away from the previous position t.

Figure 5.13(a)–(d) illustrate **TCarto** outputs for US migration data on 2018, where rows and columns represented a subset of 32 states and each cell (i, j) corresponded to the quantity that migrated from i to j. The cells were color coded based on their weights, where higher values were represented in blue. The angle constraints that had been used from left was $0°, 25°, 50°, 75°$, respectively. The available implementation of **FastFlow** did not support any angle constraints. It failed to converge and the predictor-corrector method [52] that iteratively moved the points, got stuck after 3rd iteration. It produced 152 concave cells as shown in red in Figure 5.13(e). To overcome this situation, we replaced the predictor-corrector method with uniform step size. Although it allowed us to run for more iterations, it created more concave cells and error.

Discussion

For large tabular data with many sharp local optima, **TCarto** performed way better than **FastFlow**. The rows and columns were hard to follow in a **TCarto** output, but it improved with larger angle constraints. Larger angle constraints also contribute to larger cartographic errors, and hence for real-life use cases (RQ_{A1}), it is important to choose an appropriate angle constraint that balances this potential trade-off between readability and cartographic error. This is where the convexity appears to be important than most other applications (RQ_{A2}).

TCarto is the only suitable algorithm for visualizing tabular data with angle constraint and convexity is very important for the readability of the visualization (RQ_{A1}, RQ_{A2}).

5.5 Summary

In this chapter, we demonstrated our proposed applications of large table cartograms beyond visualizing tabular data (RQ_{A1}) and explained the importance of the cells' convexity in such applications (RQ_{A2}). We described how a large table cartogram benefitted us in infographics to reveal the geospatial relation between variables, different effects in images (i.e., expanding light illumination, mosaic effect), cluster detection with densities maintaining actual coordinated positions in a scatter plot, and generating visualization for tabular data with angle constraint. We demonstrated the findings for research questions, RQ_{A1} and RQ_{A2} in this chapter.

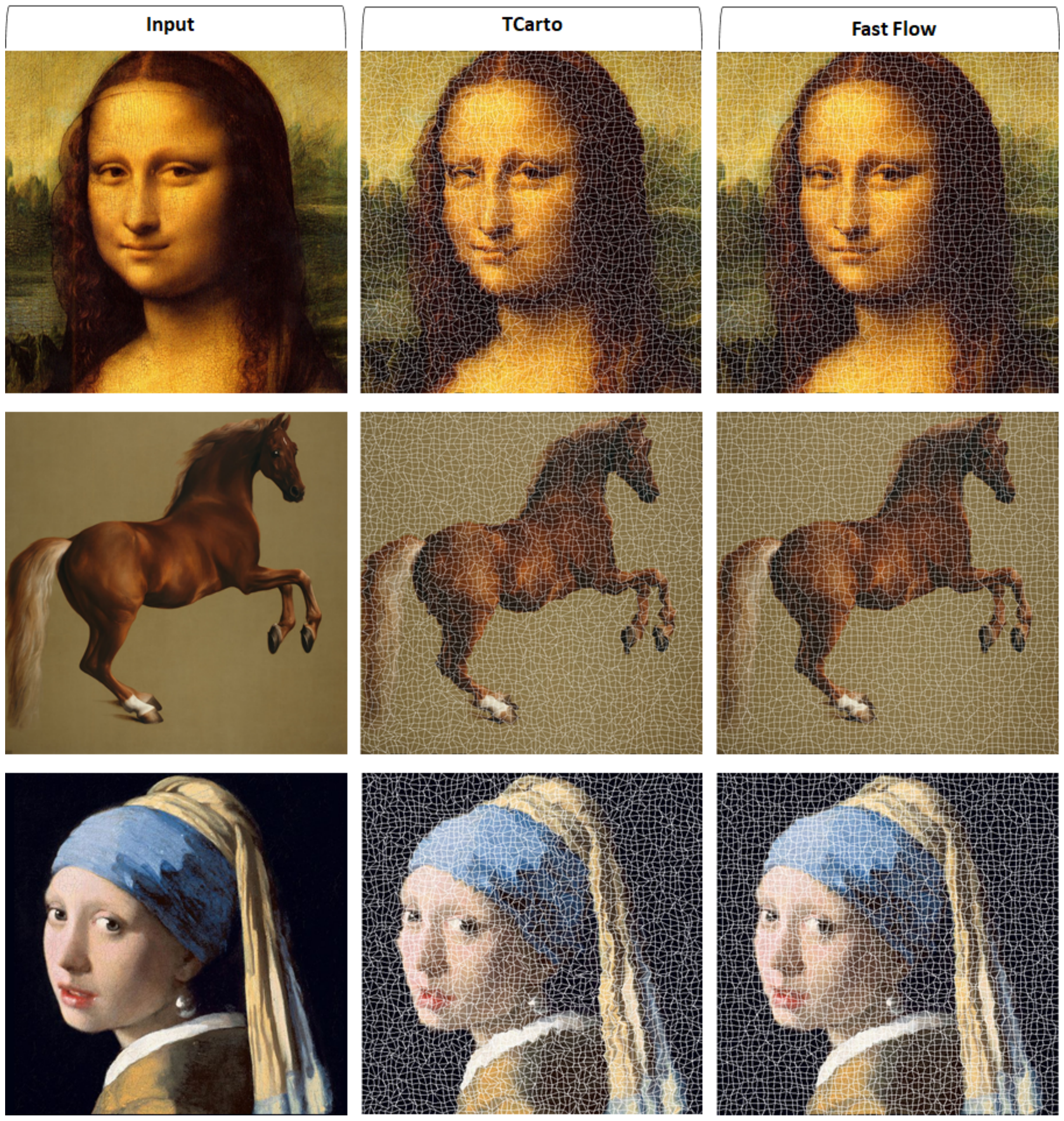


Figure 5.8: Table cartograms to create mosaic effect using 64 × 64 grid, (left) input images on first row 'Mona Lisa' by Leonardo da Vinci (1503), on second row 'Whistlejacket' by George Stubbs (1762), on third row 'Girl with a Pearl Earring' by Johannes Vermeer (1665), (middle) generated mosaic effect for **TCarto** algorithm, (right) **FastFlow** algorithm.

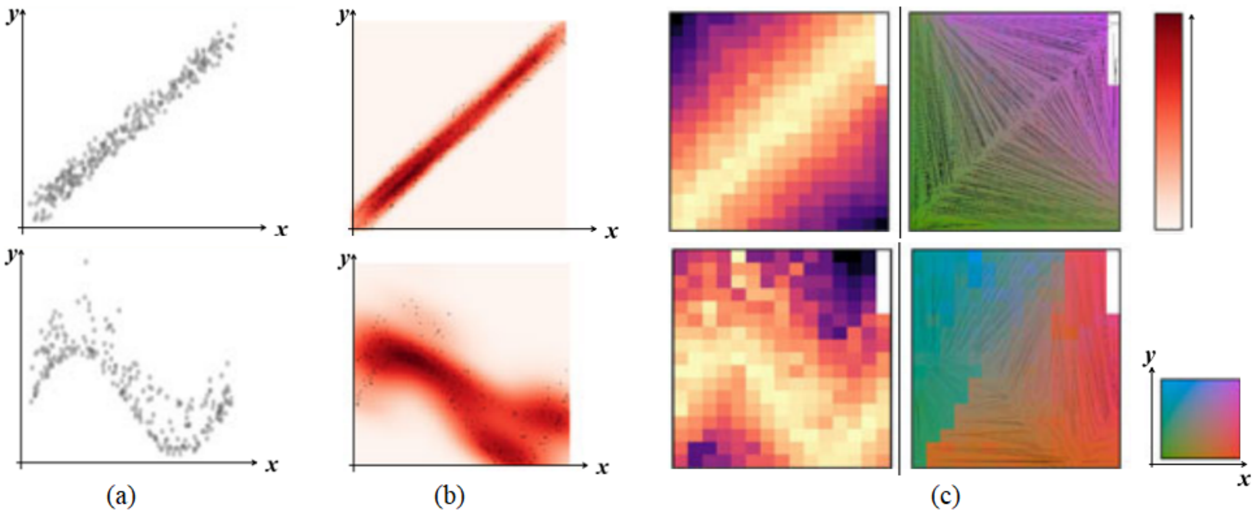


Figure 5.9: (a) Scatter plot (opacity =0.25), (b) density plot, (c) pixel-relaxed scatter plot using *linear sum assignment* [118].

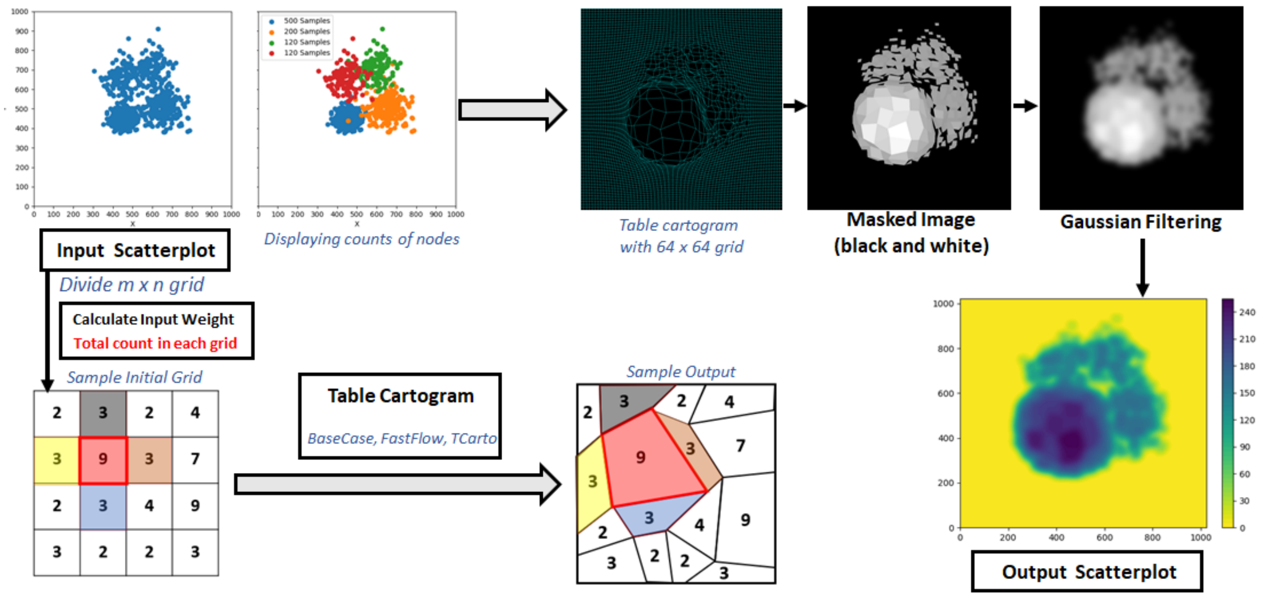


Figure 5.10: Detecting clusters with their densities using table cartogram algorithm.

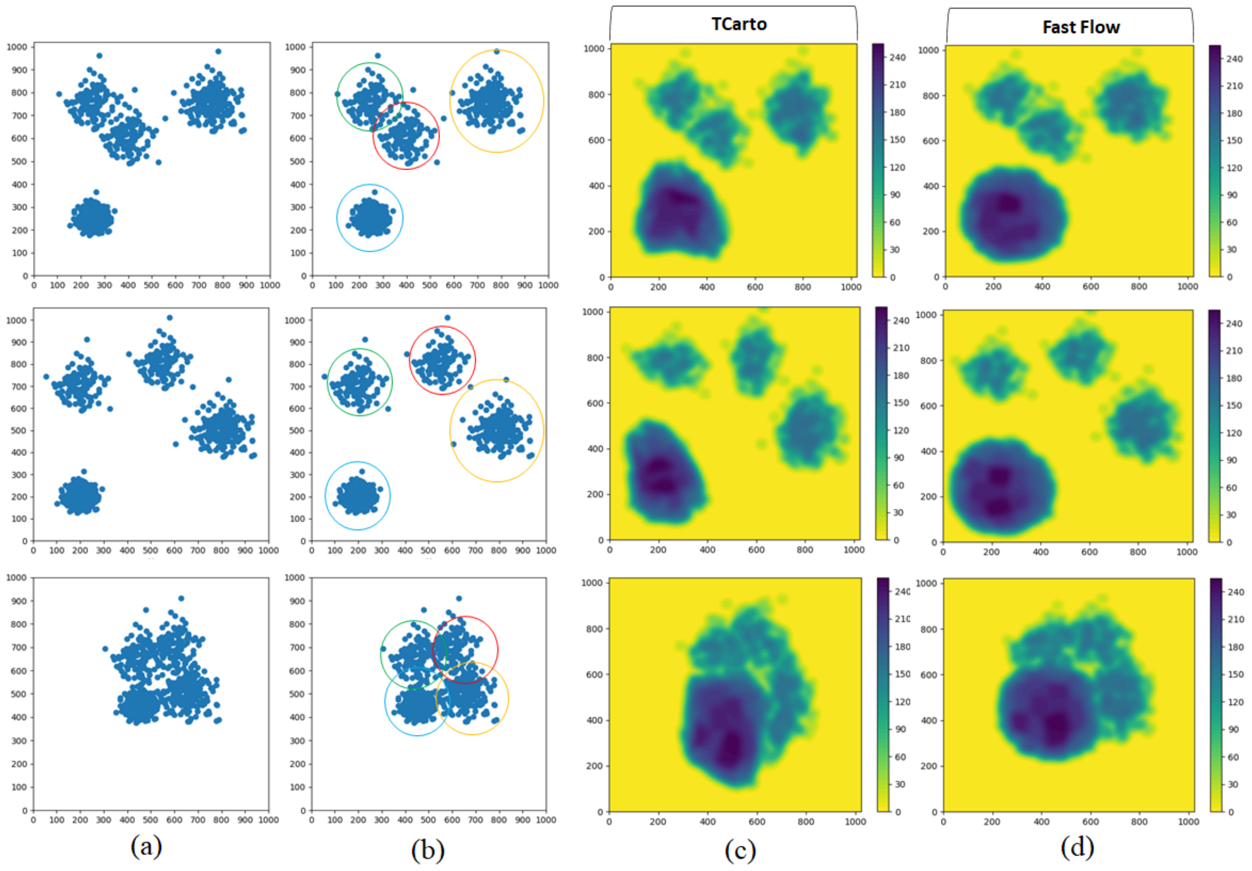


Figure 5.11: Table cartograms for interpreting clusters with their densities in a scatter plot while maintaining the clusters' position, (a) the input scatter plot, (b) marked the clusters with colored circle to describe the number of sample points. Blue, orange, red and green circle has 500, 200, 120 and 120 sample points respectively. The density and sample number remains constant, but the position is changed to generate partial overlapping(top row), no overlapping(middle row) and all four overlapping (bottom row), (c) contour plot output using **TCarto** algorithm, (d) **FastFlow** algorithm.

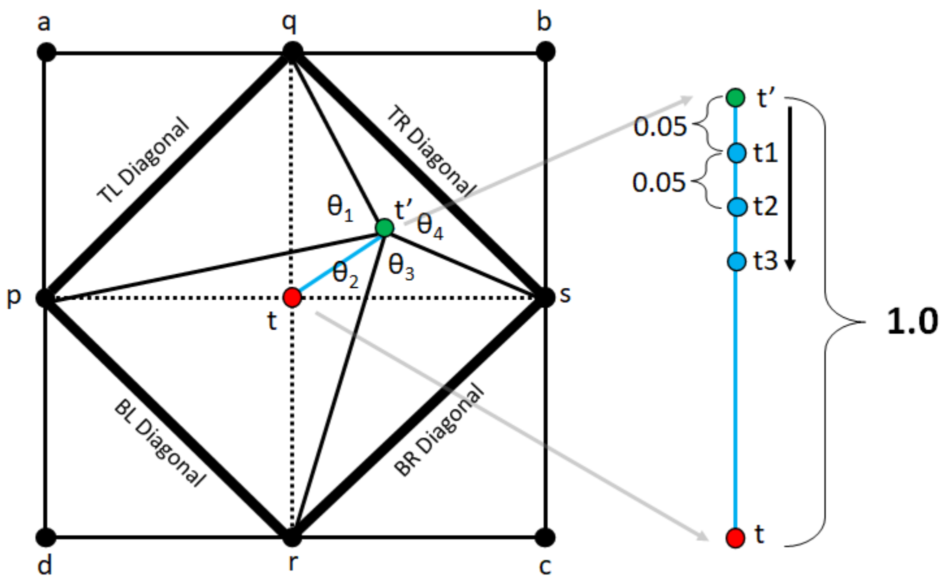


Figure 5.12: Finding new optimized position satisfying angle threshold, where t is the previous position and t' is the optimized position calculated by quadratic programming. θ_1, θ_2, θ_3, and θ_4 are $\angle pt'q$, $\angle rt'p$, $\angle st'r$, and $\angle qt's$ respectively. If all these angles are higher than threshold angle, we consider it as the new position. Otherwise, we calculate *t1* and check the angles again.

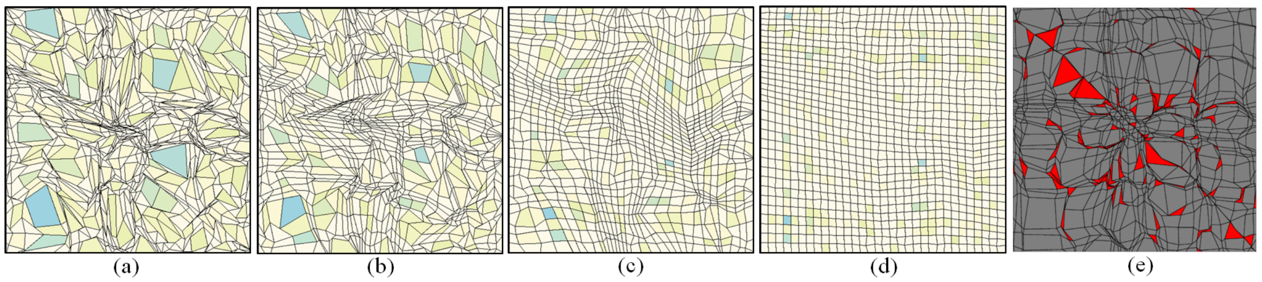


Figure 5.13: (a-d) **TCarto DivCon** output for 2018 US migration data with 0, 25, 50 and 75 angle constraint with cartographic errors of 122.01, 174.65, 175.33 and 184.71, respectively, (e) **FastFlow** output after 3rd iteration with 152 concave (red) cells (it got stuck in an infinite loop afterwards).

6 Experimental Results & Analysis

In this chapter, we describe the datasets and images that we have used in our experiments for large table cartograms. We also mention the sources, acquisition process, pre-processing steps, and various parameters of the datasets. Afterward, we define the evaluation metrics and compare the performances for all large table cartogram algorithms (i.e., **Baseline** [40], **TCarto Parallel**, **TCarto DivCon**, **FastFlow** [51]) for two different real-life datasets. We then present the experimental results. Afterward, we discuss the potential relationship among accuracy, readability of visualization, grid size and processing speed for all table cartograms. Finally, we describe the potential association of threshold angle with *cartographic error* (accuracy) and *mean aspect ratio* (the readability of the visualization) for the angle constrained table cartogram algorithms.

6.1 Experimental Setup

6.1.1 Dataset Description

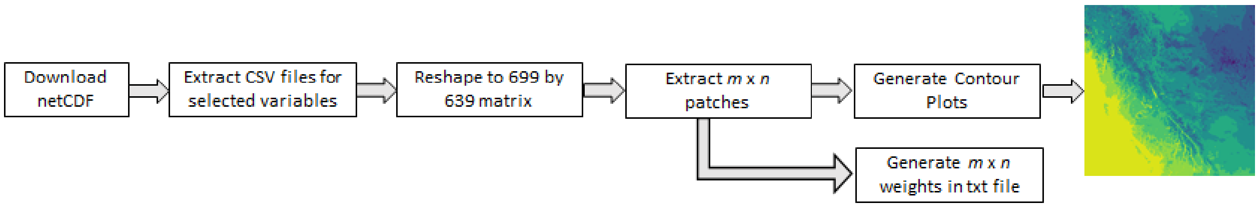
We used two datasets for our experiments: the Weather Research and Forecasting (WRF) model output for several weather parameters [44] (See Table 6.1) and the United States(US) State-to-State Migration Flows from 2015 to 2019 published by the US Census Bureau [22] (See Table 6.2).

Weather Research and Forecasting (WRF) Dataset: The Weather Research and Forecast (WRF) dataset consists of the weather data of 8 years from 2008 to 2015 [88]. Each data file contains hourly data for every day over 699 latitude and 639 longitude points of western Canada and is stored as a NetCDF file (approximately 1.3 GB) containing 10 million (10,719,864) samples. Each sample has around 36 weather parameters. The tables for this dataset does not contain any sharp spike.

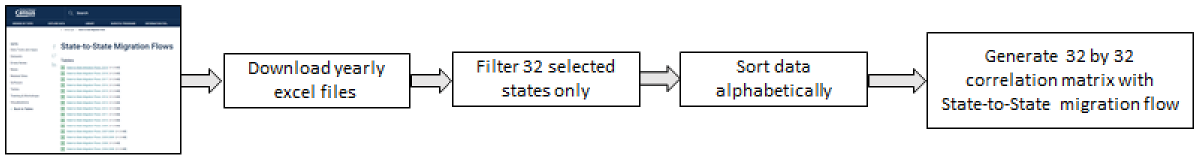
For our experiment, we chose the data for January 2013. For a missing data point, we considered the average value of its eight neighboring cells. We picked ten variables such as $ALBEDO$, TSK, $EMISS$, $SMOIS$, $LWUPB$, $PBLH$, $U10$, $PSFC$, $Q2$, $SH2O$. $ALBEDO$ measures the solar energy reflected by the earth's surface, and TSK measures the temperature of surface skin. $EMISS$ measures the emissivity of the surface, $SMOIS$ measures the water held in the surface, $LWUPB$ stands for 'Upwelling Longwave Flux', $PBLH$ stands for 'Planetary Boundary Layer Height', and $U10$ stands for the wind velocity at 10-meter height. $PSFC$ stands for surface pressure, $Q2$ stands for the ratio of water vapour mixing at 2 meter height, and $SH2O$ measures the liquid water in the surface. We downloaded the data (netCDF) files from the remote server and then extracted the CSV file. We considered the pairs of the geospatial variables for the temporal inpainting. Each temporal pair was selected randomly without repetition. Later, patches were cropped from them, maintaining the temporal order. We used perceptually uniform sequential colormap [107] to generate the contour plots from the weather dataset [88] and also calculated the weights for different grids (i.e., 16×16, 32×32, 64×64 grids). Figure 6.1(a) shows the workflow from data acquisition to the processed weight data and the contour plots.

US State-to-State Migration Dataset: The US Census Bureau [22] has published State-to-State migration data for the United States (US) since 2004. These data are stored in excel files yearly, and each yearly data file contains migration data from 52 states to 52 states. This dataset has sudden value changes to its neighboring cells and many local maxima.

For our experiment, we selected data from the year 2015 to 2019 for 32 states only (See Table 6.2). We downloaded the excel file for the 2019 year first, ordered all 52 states based on migration value in descending order (large value at top), and chose the top 32 states. Then, we downloaded all excel files from 2015 to 2019 and logged the values for our selected 32 states with the alphabetical order. Later, we generated a 32 by 32 correlation matrix with the migration data. The workflow in Figure 6.1(b) describes the data processing for the migration dataset.


Download netCDF
Extract CSV files for selected variables
Reshape to 699 by 639 matrix
Extract m x n patches
Generate Contour Plots
Generate m x n weights in txt file

(a)


State-to-State Migration Flows
Download yearly excel files
Filter 32 selected states only
Sort data alphabetically
Generate 32 by 32 correlation matrix with State-to-State migration flow

(b)

Figure 6.1: (a) Workflow for Weather Research and Forecast (WRF) [88] data processing, (b) workflow for the US State-to-State Migration Dataset [22].

6.1.2 Evaluation Metrics

The primary two assessments of the large table cartogram algorithms are the accuracy and the readability of the visualization. The accuracy of the cartogram refers to how accurately each polygon spatially transforms based on its desired weights. The readability of the visualization defines how easily a user can read or comprehenad the visualization accurately. We considered two accuracy calculations and two measures for the readability of the visualization.

Accuracy

One standard quality metric for accuracy is the *average cartographic error*, $\widetilde{e_i}$. If $A_i^{desired}$ is the desired area or the given weight of ith polygon (cell) and A_i^{actual} is the actual area after the cartographic transformation [72], then $\widetilde{e_i}$ is computed as follows: $\widetilde{e_i} = \frac{1}{N}\left[\sum_{i=1}^{N} e_i\right]$, *where* $e_i = \frac{\left|A_i^{desired} - A_i^{actual}\right|}{A_i^{desired}}$. (See Equation 2.2 for details)

The mean quadratic error I_{MQE} is another commonly used quality metric for accuracy used in the literature [76]: $I_{MQE} = \frac{1}{N}\sqrt{\sum_{i=1}^{N} {e_i}^2}$, where $e_i = \frac{\left|A_i^{desired} - A_i^{actual}\right|}{A_i^{desired} + A_i^{actual}}$. (See Equation 2.6 for details)

Comprehensiveness

We considered *mean aspect ratio* (mean AR) and *concave count* (α) to measure the readability of a visualization. The *concave count* (α) is simply the number of concave quadrilaterals in the output. The mean aspect ratio is defined as follows: $Mean\ AR = \frac{1}{N}\sum_{1 \le i \le N} \frac{\min(w_i,h_i)}{\max(w_i,h_i)}$, where w_i and h_i are the width and height of the ith cell, respectively. (See Equation 2.8 for details)

6.2 Results and Analysis

6.2.1 Performance Comparison with Weather Dataset

The weather dataset [88] contains no sudden value changes to its neighborhood cells and not many local maxima. We compared all the measurement metrics for accuracy and readability of the visualization among ***Baseline*** *(Evans et el.)* [40], ***TCarto Parallel***, ***TCarto DivCon*** and ***FastFlow*** [52] algorithms. ***Baseline*** *(Evans et el.)* and both ***TCarto*** are table cartograms, whereas ***FastFlow*** is a cartographic transformation algorithm. We fixed the outer bounding box holding all regions (polygons) to make ***FastFlow*** generate an output similar to a table cartogram. It mostly behaved like a table cartogram without following the convexity constraints. We used the large table cartogram techniques to visualize the spatial correlation of a pair of variables from the Weather Research and Forecasting (WRF) data [44]. We took a weather variable as weight and another one to generate the initial contour plot. We measured the metrics for ten different pair wise combinations of weather variables for 16×16 (See Tables A.1 and A.2), 32×32 (See Tables A.3 and A.4), 64×64 (See Tables A.5 and A.6) tabular grids. Finally, we categorized the data based on the grid size of tabular grid cells and eventually considered the average value. Table 6.1 is the summarized table of all Tables A.1, A.2, A.3, A.4, A.5, and A.6, which are provided in the Appendix.

Performance Comparison: From Table 6.1, we can observe that the **Baseline** had a very high accuracy but the average aspect ratio was too low to be used in the applications that we explored in Chapter 5.

Table 6.1 reports the results on weather datasets [88], where there is no sudden spikes and not many local maxima. Both **TCarto DivCon** and **FastFlow** had very small average cartographic error and mean quadratic error (RQ_{B1}), with **FastFlow** having the smaller average among the two. The average cell aspect

Grid	Algorithm	$\widetilde{e_i}$	I_{MQE}	*Mean AR*	*Mean* α
16 by 16	Baseline	0.00007	0.0000026	0.21039	0
	TCarto Parallel	0.28991	0.00952	0.80894	0
	TCarto DivCon	0.03421	0.00161	0.76672	0
	FastFlow	0.00763	0.00034	0.79762	0
32 by 32	Baseline	0.0004	0.0000012	0.12073	0
	TCarto Parallel	0.23297	0.00402	0.83869	0
	TCarto DivCon	0.02905	0.00076	0.82364	0
	FastFlow	0.00648	0.00014	0.85596	0.7
64 by 64	Baseline	0.00007	0.0000009	0.11617	0
	TCarto Parallel	0.49741	0.91715	0.83401	0
	TCarto DivCon	0.03632	0.00052	0.75623	0
	FastFlow	0.00817	0.000095	0.791	5

Table 6.1: Cartographic error ($\widetilde{e_i}$), mean quadratic error (I_{MQE}), mean aspect ratio (AR) and average concave count (α) for weather dataset [88].

ratio was above 0.75 for both **TCarto** and **FastFlow**, where **FastFlow** had better aspect ratio compared to **TCarto**. Only a few concave cells appeared in **FastFlow** (even for large tables).

Table 6.1 indicates that constraint based table cartogram **TCarto DivCon** can compute large table cartograms with small cartographic error (RQ_{B1}). **TCarto DivCon** and **FastFlow** both perform similarly on the data table of weather datasets [88] that have no high spike in values; however, **FastFlow** generates a few concave cells (RQ_{B2}, RQ_{B3}). Sometimes, the outputs by **TCarto Div-Con** and **FastFlow** are slightly different. However, user studies or interviews with domain expert (meteorologists for weather dataset) may potentially help to understand which technique reveals the relationship than others.

6.2.2 Performance Comparison with Migration Dataset

The US migration dataset is a 32 × 32 matrix as the weighted table and consists of a very sudden value change in neighboring cells. This dataset also has many local maxima. We made a comparison table among ***Baseline** (Evans et el.)* [40], ***TCarto Parallel***, ***TCarto DivCon*** and ***FastFlow*** [52] algorithms. For this, we ran ten iterations for both ***TCarto Parallel*** and 15 iterations for ***FastFlow***. We categorized the US migration data by year and generated the comparision table (See Table 6.2).

Performance Comparison: In Table 6.2, **Baseline** has the better accuracy than the others, but contains very low *Mean Aspect Ratio*. It made it unable to apply on the applications. Here, we observed

Year	Algorithm	$\widetilde{e_i}$	I_{MQE}	*Mean AR*	α
2019	Baseline	0.0083	0.00170	0.12322	0
	TCarto Parallel	0.9676	0.030685	0.949539	0
	TCarto DivCon	119.777	0.01297	0.563086	0
	FastFlow	5.9962	0.00813	0.539933	346
2018	Baseline	0.0081	0.00082	0.12335	0
	TCarto Parallel	0.9662	0.03066	0.95056	0
	TCarto DivCon	101.1979	0.0131	0.56454	0
	FastFlow	12134.8996	0.02895	0.591651	746
2017	Baseline	0.0081	0.00086	0.125121	0
	TCarto Parallel	0.9666	0.03068	0.94793	0
	TCarto DivCon	111.805	0.01261	0.568921	0
	FastFlow	2100.1133	0.02229	0.187013	768
2016	Baseline	0.0079	0.000845	0.122745	0
	TCarto Parallel	0.969	0.03071	0.94822	0
	TCarto DivCon	89.5294	0.01248	0.570147	0
	FastFlow	67.3776	0.016234	0.501046	513
2015	Baseline	0.0086	0.00093	0.123013	0
	TCarto Parallel	0.9656	0.0307	0.947668	0
	TCarto DivCon	131.5762	0.01229	0.571858	0
	FastFlow	2562.7078	0.02280	0.424352	752

Table 6.2: Results on US State-to-State Migration Flows from 2015 to 2019 published by the US Census Bureau [22]. On each input, **TCarto Parallel** runs until 10 iterations and **FastFlow** until 15 iterations with our modification to tackle the convergence problem.

both **TCarto** to have lower cartographic error, and better aspect ratio compared to **FastFlow**(RQ_{B2}). Furthermore, **FastFlow** produced hundreds of concave cells with inconsistent higher *cartographic error*. Though **TCarto Parallel** had better accuracy and average aspect ratio than **TCarto DivCon**, only ten iterations are not good enough for **TCarto Parallel** to disform the whole grid and to produce meaningful output. Thus, **TCarto DivCon** is preferred for the dataset having lots of spikes in values and with many local optima.

The data table of US State-to-State migration dataset [22] contains sudden value changes in adjacent cells and many local optima. For this type of dataset, **TCarto** gives better aspect ratio and often results less cartographic error, but **FastFlow** produces many concave cells. **Baseline** gives better accuracy conceding lower aspect ratio (RQ_{B2}, RQ_{B3}), but does not produce a readable output.

6.2.3 Relationship between Accuracy and Readability of Visualization

We categorized weather data of 64×64 grid from Table A.5 and Table A.6 by different large table cartogram algorithms. We plotted *cartographic error* and *mean aspect ratio* in a scatter plot with trend lines to visualize the relationship between accuracy and readability of the visualization. Figure 6.2 demonstrates that **Baseline** has the highest accuracy, but lowest *mean aspect ratio*. **TCarto Parallel** has the lowest accuracy. The scatter plot shows that both **FastFlow** and **TCarto DivCon** look promising since both of them have small amount of *cartographic error* (lower than 0.1) and higher *mean aspect ratio* (RQ_{B4}).

Though *mean aspect ratio* is similar for both **FastFlow** and **TCarto DivCon** algorithm, the accuracy of **FastFlow** is slightly better than **TCarto DivCon**. Thus, **FastFlow** is preferable for all large table cartograms when convexity is not crucial. **TCarto DivCon** is recommended for large table cartogram algorithms when cells' convexity is important (RQ_{B4}).

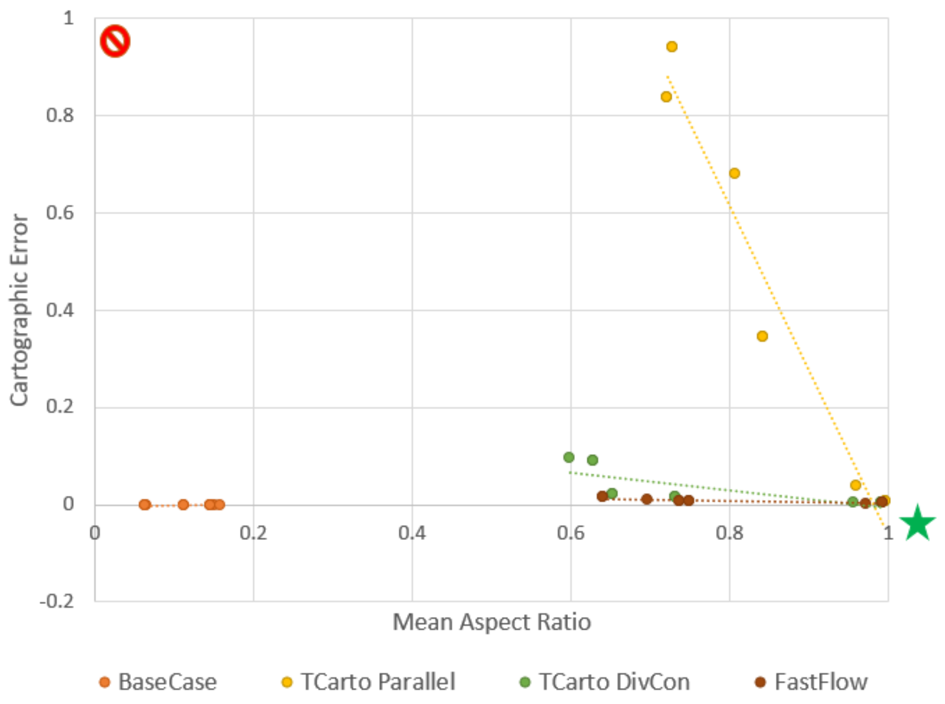


Figure 6.2: Relationship between *cartographic error* and *mean aspect ratio* for all table cartograms on weather dataset with 64×64 grid, where each data point represents the performance measurements of two weather variables (See Tables A.5 and A.6), and the straight lines represent the linear regression for their corresponding color (green star is the best case and red circular shape is worst scenario).

FastFlow is preferable for large table cartograms when both accuracy and readability of the visualization are accountable, but convexity is not crucial. If convexity is important, **TCarto DivCon** is recommended (RQ_{B4}).

6.2.4 Relationship of Grid Size with Accuracy and Processing Speed

It is interesting to investigate how accuracy changes by different grid sizes for constraint based algorithm. To analyze this, we categorized weather data for **TCarto DivCon** based on various grid sizes (i.e., 16×16, 32×32, 64×64) from Tables A.2, A.4, and A.6. We plotted a box plot with *cartographic error* where each data point is the accuracy for a pair of weather variables (one is weight and another is contour plot). Figure 6.4 shows that the minimum values of all grid sizes are similar and close to 0.05 . The mean error for 16×16 grid is the highest, whereas 32×32 grid performs better than other grids based on the accuracy. The performance of 32×32 grid looks promising since the first half of this grid is lower than 0.01 (RQ_{B4}).

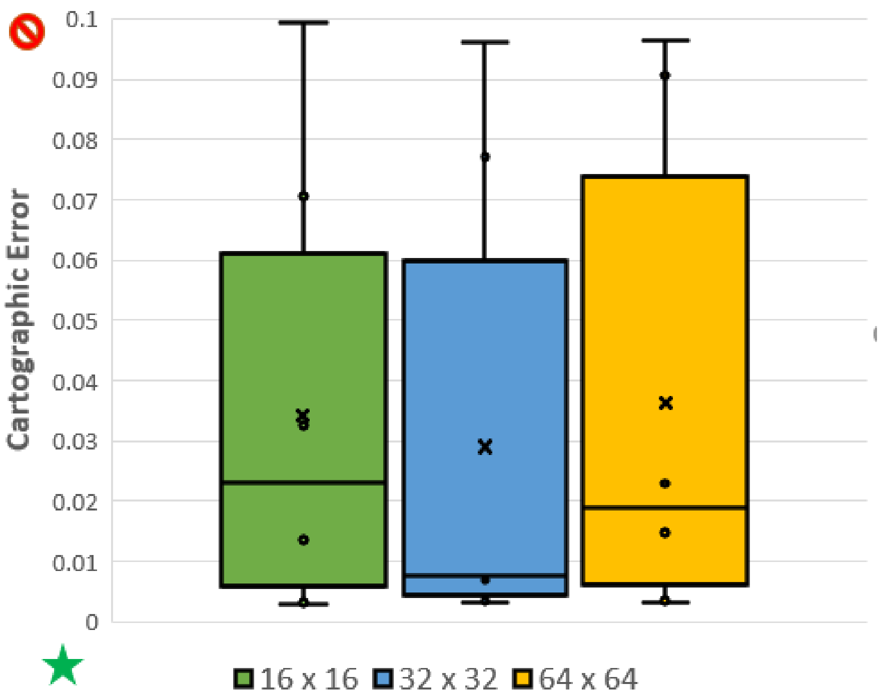


Figure 6.3: Relationship between *cartographic error* and the grid size for **TCarto DivCon** on weather dataset, where each data point represents the accuracy of two weather variables (one is weight and another is contour plot). Green star represents the best case and red circular shape shows the worst case.

Another interesting analysis would be to investigate whether the increment of grid size for **TCarto DivCon** raises the processing time linearly. Similar to earlier, we categorized the weather data based on the grid size and plotted the processing time in a box plot. Figure 6.4 shows that processing time increases with the increment of the size of the grid. The increase in processing time looks exponential rather than linear (RQ_{B4}).

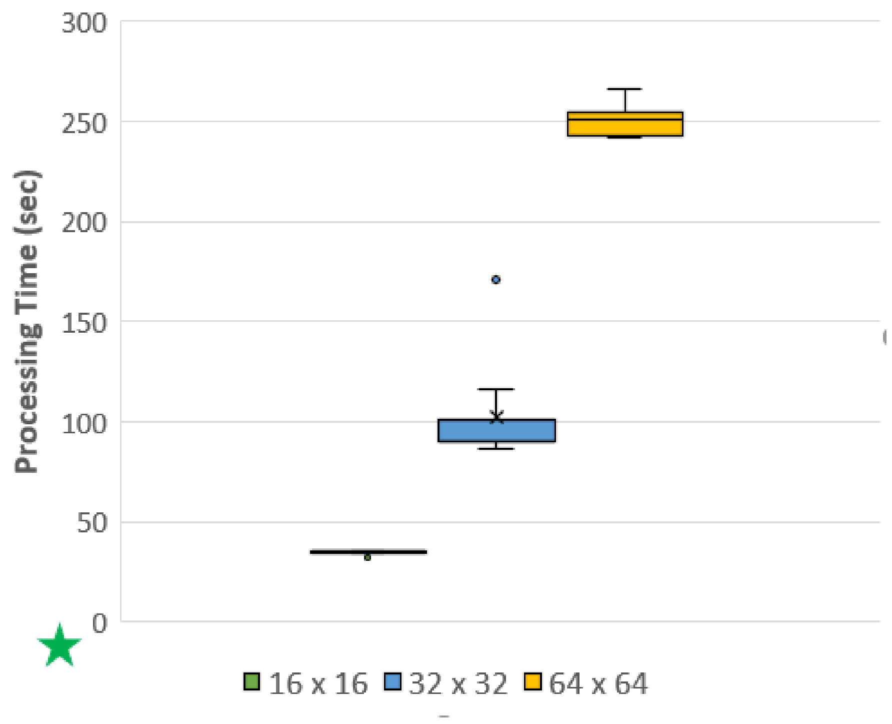


Figure 6.4: Relationship between *processing time* and the grid size for all table cartograms on weather dataset, where each data point represents the processing time for the algorithm of two weather variables (one is weight and another is contour plot). Green star represents the best case with lower processing time.

Accuracy does not change linearly by changing the grid size. **TCarto DivCon** performs better with 32×32 grid based on accuracy. However, processing time changes exponentially with the change of grid size (RQ_{B4}).

6.2.5 Relationship of Threshold Angle with Accuracy and Readability of Visualization for Angle Constraint Table Cartogram

We augmented **TCarto DivCon** with angle constraint so that the corner of each cell was above a given angle threshold. It is important to understand the corresponding rows and columns for each cell. We ran **TCarto DivCon** with angle constraints on US migration data (See Table B.1). Figure 6.5 shows all the outputs for different angle constraints such as $0°, 20°, 40°, 60°$, and $80°$ respectively (RQ_C). The outputs follow the color scheme from blue to yellow based on the weights, where blue is for high value and yellow is for the opposite. From Figure 6.5, we can observe that it is difficult to identify the row and column for a cell upto $20°$ angle constraint and it becomes easier to comprehend for more than $40°$ angle constraint.

Since the angle constraint blocks the cells to deform by enforcing additional angle limitations, it definitely compromises the accuracy. For understanding the relationship between accuracy and threshold angle, we plotted *cartographic error* (accuracy) with different angle in a line chart at Figure 6.6. It shows that *cartographic error* increases with the increment of threshold angle overall. However, the *cartographic error* lines dip for most of the cases (i.e., on 2015, 2016, and 2018) at the 40° angle constraint. We also plotted another line chart with *aspect ratio* and threshold angle (See Figure 6.7). It shows that *aspect ratio* increases exponentially with the increment of threshold angle.

While considering both line charts simultaneously from Figure 6.6 and Figure 6.7, we can observe a trade-off between the *cartographic error* and the *aspect ratio*. The *aspect ratio* improves with the increase of the threshold angle escalating the *cartographic error*. Since the readability of visualization enhances with the increase of threshold angle conceding some accuracy an appropriate threshold angle may be chosen based on the ratio or the priority of the accuracy and an aspect ratio parameters.

TCarto DivCon with angle constraint can generate outputs for large tables with the *aspect ratio* more than 0.5 . This *aspect ratio* can be increased upto 0.95 by raising threshold angle; however it also increases *cartographic error*. We can increase the threashold angle to enhance the readability of the visualization upto the acceptable *cartographic error* based on the requirement. The row and column identification for a cell is difficult upto 20° threahold angle, but it becomes easier from 40° angle restriction (See Figure 6.5)(RQ_C).

6.3 Summary

In this chapter, we explained the sources, acquisition process, data preprocessing, and all used parameters of the datasets. After mentioning the evaluation metrics, we discussed the performances of all four table cartogram algorithms by comparing the measurement metrics with two different dataset. Then, we discussed the potential relationship between accuracy and readability of visualization for all table cartograms. We also demonstrated the relationship of the grid size with accuracy and the processing time for constraint based table cartogram, **TCarto DivCon**. In the end, we presented the potential association of threshold angle with accuracy and readability of visualization for angle constraint table cartograms. Most importantly, we provided the findings and recommendations for research questions RQ_{B1} to RQ_C.

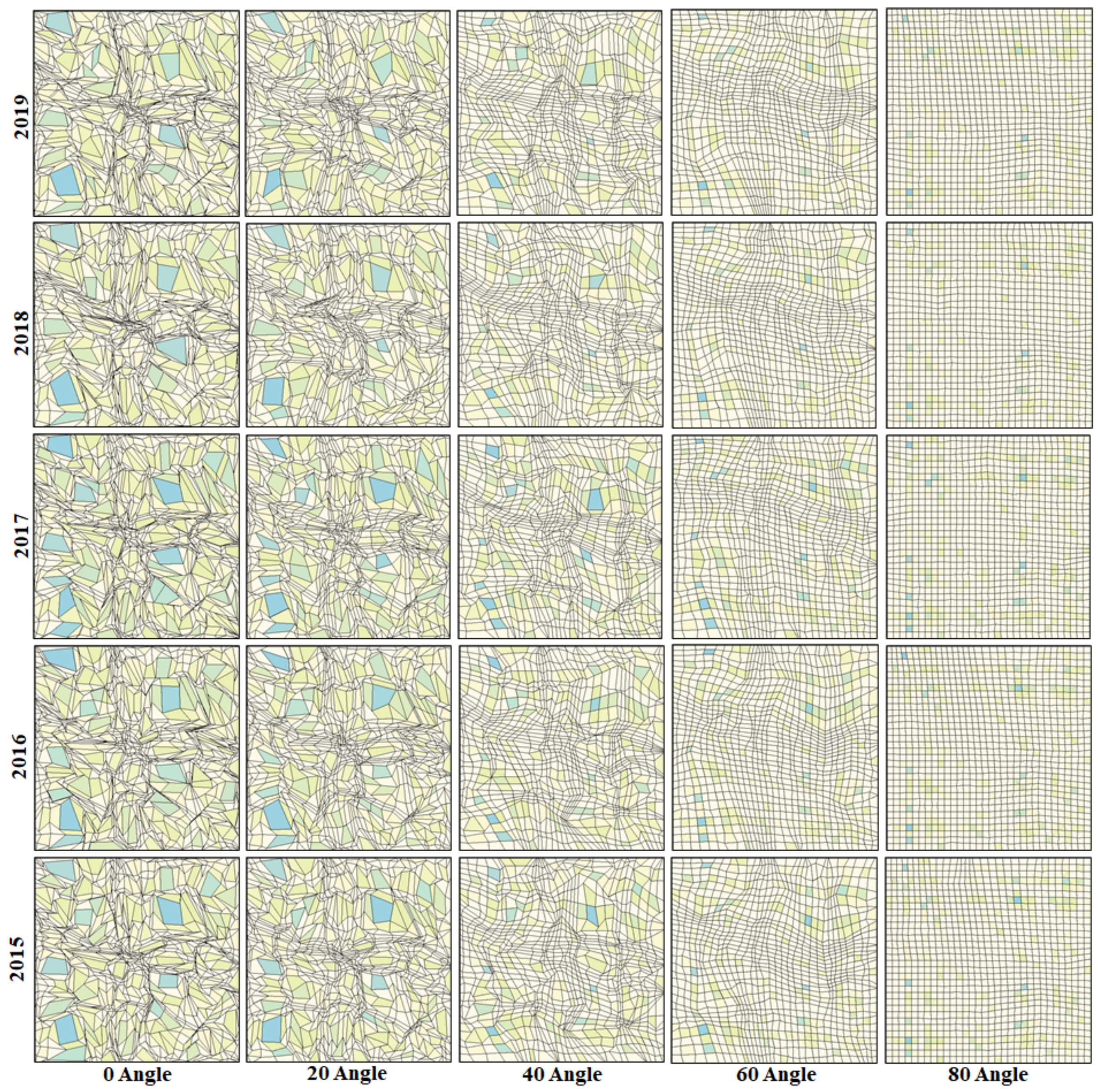


Figure 6.5: The output of angle constraint with **TCarto DivCon** for US State-to-State migration data [22] with 0, 20, 40, 60 and 80 degree angle constraints on the years from 2015 to 2019. The color and area of each cell represent the similar variable, the number of population are migrating from one state to another state. Blue and yellow color represent the highest and lowest values respectively. So, blue regions are supposed to be larger, whereas the yellow regions are smaller. Rows and columns of this visualization indicate the outgoing and incoming states. These states are Alabama, Arizona, Arkansas, California, Colorado, Connecticut, Florida, Georgia, Illinois, Indiana, Iowa, Kansas, Kentucky, Louisiana, Maryland, Massachusetts, Michigan, Minnesota, Missouri, Nevada, New Jersey, New York, North Carolina, Ohio, Oklahoma, Oregon, Pennsylvania, Tennessee, Texas, Virginia, Washington and Wisconsin in the exact order from left to right for the columns and top to bottom for the rows.

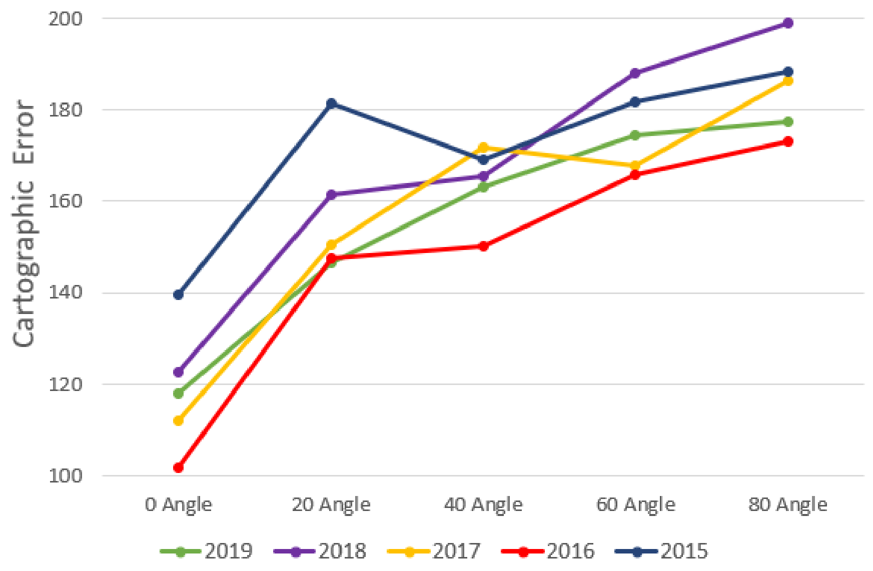


Figure 6.6: Relationship between *cartographic error* and the threshold angle for **TCarto DivCon** with angle constraint table cartogram on US State-to-State migration flow dataset [22], where each data point represents the *cartographic error* from Table B.1.

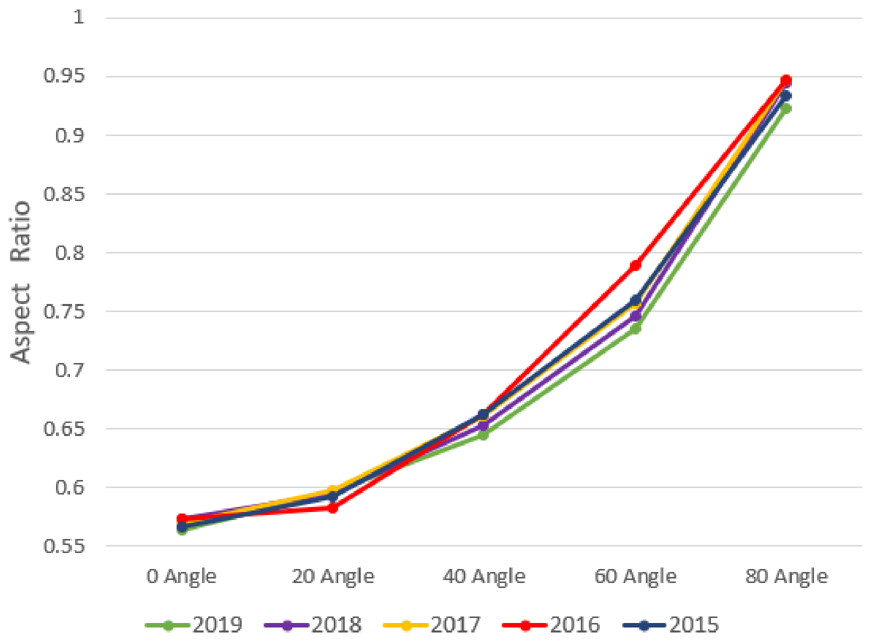


Figure 6.7: Relationship between *aspect ratio* and the threshold angle for **TCarto DivCon** with angle constraint table cartogram on US State-to-State migration flow dataset [22], where each data point represents the *cartographic error* from Table B.1 .

7 Conclusion

7.1 Summary

The primary aim of this thesis is to explore and investigate table cartogram algorithms for a large table with thousands of cells. We implemented an errorless table cartogram algorithm (**Baseline**) proven by Evans et al. [40]. Even with a high accuracy, this algorithm was unable to be used in the proposed applications because of the skinny cells that it produces (See Figure 1.3). We adapted another recent algorithm of cartographic transformation (**FastFlow**) [52] due to its fast processing time and investigated how it works as a table cartogram. Since **FastFlow** produces concave cells, we proposed a new scalable optimization-based table cartogram generation technique, **TCarto**. Later, we augmented TCarto with angle constraint for better readability with the expense of introducing cartographic error.

In this thesis, we also showed that large-scale table cartograms could potentially be applied to various applications beyond visualizing general tabular data. We used large table cartograms in creating infographics of cartographic transformations that can reveal correlations among different geospatial variables, or to generate different visual effects (i.e., increase the illumination of an image, mosaic art effect), or for better understanding of clusters with densities in a scatterplot. In all these applications, the underlying data had the property that the neighboring cells' values were similar, and the number of local maxima was small. We observed both **TCarto** and **FastFlow** to perform similarly in such scenarios. However, the cells in the **FastFlow** output appeared to be much more uniform compared to those of **TCarto**. Therefore, in the image-based applications, the images were less distorted in the **FastFlow** output.

Both the constraint-based approach and physics-inspired cartographic map transformation effectively generated large-scale cartograms (with low cartographic error, good cell aspect ratio, and few concave cells) for image and weather datasets [88]. However, for data tables with a large number of local maxima and high spikes, we observed constraint-based optimization to work better than the flow-based traditional cartographic approach. Furthermore, we investigated the possible relationships among different criteria such as accuracy, the readability of the visualization, processing speed and the grid size based on various measurement metrics (i.e., cartographic error, mean aspect ratio, and count of concave cells).

In general, for the large tabular dataset with many local optima with large spikes and high value differences between adjacent cells, neither the **TCarto** nor **FastFlow** produced appealing output. However, **TCarto** with angle constraint improved the readability of the table conceding some cartographic error. This suggested the importance of choosing an appropriate restricted angle that allowed users to follow the rows and columns, as well as to understand the relative cell areas, peaks and valley regions.

7.2 Contribution

Our main contributions to this thesis are as follows:

1. We explored different table cartograms for large tables with thousands of cells. While investigating several cartographic transformations and table cartogram algorithms, we found that some were adaptable and applicable, whereas some were not. We implemented an errorless table cartogram (**Baseline**) based on the theoretical proof by Evans et al. [40] that produced skinny triangles (See Figure 1.3). We amended another modern and fast cartographic transformation technique (**FastFlow**) by Gastner et al. [52] with a restriction at the outer boundary to adapt a large table cartogram. The outputs for this algorithm looked promising for the data tables with similar values on the neighboring cells having no high spikes; however, it produced concave and overlapping cells for the data distribution with high spikes and many local maxima (See Figure 1.4).

2. We also proposed a new constraint-based table cartogram technique, **TCarto** that preserves cell convexity yet capable of handling large tables. In this approach, each node moved based on neighboring nodes and weights via local optimization using quadratic programming [62, 43]. We proposed two approaches: column-based and quadtree-based approaches for parallelization, where the quadtree-based method performed faster than the column-based approach because of its faster weight-dispersion.

3. We demonstrated several real-life applications (RQ_{A1}) of large table cartograms, e.g., for analyzing correlations between geospatial variables, understanding clusters and their densities in scatterplots, and creating visual effects in images (i.e., expanding illumination, mosaic art effect). We illustrated that both **TCarto** and **FastFlow** could potentially be effective in these applications and discussed the importance of convexity for each application (RQ_{A2}).

4. We examined the performance of **Baseline**, **TCarto**, and **FastFlow** with two different real-life datasets: a meteorological weather dataset [88] and a US State-to-State migration flow dataset [22] by comparing several measurement metrics such as cartographic error, mean aspect ratio, number of concave cells (RQ_{B1}, RQ_{B2}, and RQ_{B3}). **Baseline** showed minimal errors with a lower aspect ratio. The data distribution of the weather dataset had no spikes and many local minima, and both **TCarto** and **FastFlow** showed impressive performance. For the migration dataset with many sharp local maxima, both **TCarto** and **FastFlow** produced low-quality output, whereas **FastFlow** generated hundreds of concave cells.

5. We also augmented **TCarto** with angle constraint to enhance the readability of the visualization. We produced outputs for migration data with better readability by increasing the restricted angle conceding

some accuracy (RQ_C). We also analyzed the potential relationships among the measurement criteria, such as accuracy, readability of the visualization, processing speed, the grid size and restricted angles (RQ_{B4}, RQ_C). When both accuracy and readability of the visualization were concerned, **FastFlow** was preferable; however, it generated concave cells. **TCarto** performed better and faster among table cartograms that do not produce concave cells.

7.3 Limitations

In this section, we discuss the limitations of this thesis.

7.3.1 Data Limitations

We simulated all the large table cartograms on two different real-life datasets: a meteorological weather dataset [88] and a US State-to-State migration flow dataset [22]. The results and analysis with datasets from further domains could provide us with better insights towards generalizability of these algorithms and their applications.

The tables for the meteorological weather dataset [88] had no high value changes in neighboring cells and had small number of local maxima. In contrast, the data characteristics of the US State-to-State migration flow dataset [22] were just the opposite, with high spikes and many local maxima. We ran all table cartograms on both types of datasets. It may be useful to run controlled study with synthetic data to gain insights into how various data features impacts the quality of the visualizations.

7.3.2 User studies

Since we devoted our focus and effort primarily to explore the scope of table cartogram for large tables, we did not consider conducting controlled user studies or interviews with the domain experts (meteorologist for weather dataset [88], artists or photographers for visual image effects) to evaluate the readability or the artistic quality of the generated outputs by **TCarto** and **FastFlow**. For the weather dataset [88], such user studies or interviews with meteorologists might potentially be helpful in understanding the best technique to reveal the relationships among various weather parameters. It would also be interesting to investigate the performance of the cartogram-based approaches and modern artificial intelligence-based techniques for creating digital arts.

Furthermore, we used two measurement criteria: mean aspect ratio and the number of concave cells for the readability of the visualization. Different people might observe such visualizations differently. Since such visual observation is subject to human interpretation, it would thus be interesting to conduct a user study on the readability of the visualizations generated by different table cartogram algorithms.

7.3.3 Evaluation metrics

In all our proposed applications, the underlying data characteristics were the small number of local maxima and the similarity among values of the neighboring cells. **TCarto** and **FastFlow** performed well and similarly in such applications. However, we observed **FastFlow** produced more uniform outputs than **TCarto**. Thus, output images were less distorted in the **FastFlow** output for the image-based applications. The measurement metrics we used were related to the accuracy or the readability of the table cartogram output. We did not consider any metric for the image warping or image distortion. It would thus be interesting to devise such performance metrics that also took the image warping or distortion into account.

7.3.4 GPU Implementation

We parallelized our proposed table cartogram algorithm, **TCarto** to compute a large table with thousands of cells. Initially, we used a column-based parallelization approach, **TCarto Parallel** (See Algorithm 1). Later, we improved the processing time with a quadtree-based parallelization approach, **TCarto DivCon** (See Algorithm 2). GPU computation had not been used to leverage any of the current implementations. Thus it would be an exciting opportunity to explore whether GPUs can be leveraged to compute table cartograms in interaction time.

7.4 Future Work

In this thesis, we explored different table cartogram techniques for large table with thousands of cells. We implemented an existing table cartogram technique and adapted a traditional cartographic algorithm so that they could run large tables. We also proposed a new constraint based table cartogram technique using quadratic programming powered by parallel computing. We also demonstrated several potential applications for large table cartograms beyond tabular data visualizations. There are still scopes to improve the table cartogram algorithm and explore various new applications. In Section 7.3, we discussed several limitations that may lead to the future works.

1. The readability of a visualization has been measured using metrics such as mean aspect ratio, number of concave cells. However, such visual interpretation varies from person to person. Thus, a user study to investigate the readability of the outputs by table cartograms, might provide better insights.
2. We simulated and run all the table cartogram algorithms on real-life data. It would be interesting to run the table cartograms on other data to generalize the algorithms, and also on synthetic or computer generated data to understand how data properties influence visualization quality.
3. A controlled study or interview with domain experts (i.e., meteorologist for weather dataset [88], artists or photographers for visual image effects) might give us better insight into the readability of the visualizations or the artistic quality of the generated outputs by different algorithms.

4. Our proposed algorithm can compute thousands of cells leveraging by parallel computation, but can not process faster enough to use it interactively. In fact, none of the algorithms has yet been leveraged with GPU implementation and can process in interaction time (in a few seconds).

5. The performance of the algorithms has not considered any metric with image distortion. It would be interesting to devise such measurement metric that can consider image distorting or image warping. While using table cartogram to reveal potential relation between a pair of geospatial variables, we only examined positive and negative relationships. It would be interesting to investigate whether one can compute the strength of such relationship from the measurement metric that considers the image distortion or image warping.

6. We have explored several applications of large table cartograms. There are still lots of applications exist that can be achieved using table cartogram such as density equalizing cartogram, unique-identifier matrix barcode system. Researchers may feel interested to explore more applications of large table cartograms.

7. The table cartograms only support the positive weights since the polygons of distorted output can occupy only the positive areas. It would be interesting to explore useful data transformations such that the table cartogram algorithms can support negative and zero values by shifting or rescaling the range of input weights.

8. Some algorithms are appropriate for better accuracy, and some are for aspect ratio. It would be interesting to design algorithms that takes user priorities as input and tunes the algorithm's parameters to produce an output that best meets the user's need.

9. The expectation is that the larger grid size might produce the less *cartographic error*. In Figure 6.4, we observed that 32×32 grid has a lower *cartographic error* than 64×64 grid. It would be interesting to investigate what restricts larger grid size to obtain better accuracy. Sometimes, load distribution with different numbers of $CPUs$ might affect the cell distortion. The potential relationship between the accuracy and the number of $CPUs$ can be examined in future.

References

[1] Cartogram history. `https://makingmaps.net/tag/cartograms-history`, 2008.

[2] Tinku Acharya and Ajoy K Ray. Affine and projective transformations. `https://www.graphicsmill.com/docs/gm/affine-and-projective-transformations.htm`.

[3] Tinku Acharya and Ajoy K Ray. *Image processing: principles and applications.* John Wiley & Sons, 2005.

[4] Md Jawaherul Alam, Therese Biedl, Stefan Felsner, Michael Kaufmann, Stephen G Kobourov, and Torsten Ueckerdt. Computing cartograms with optimal complexity. *Discrete & Computational Geometry*, 50(3):784–810, 2013.

[5] Md Jawaherul Alam, Stephen G Kobourov, and Sankar Veeramoni. Quantitative measures for cartogram generation techniques. In *Computer Graphics Forum*, volume 34, pages 351–360. Wiley Online Library, 2015.

[6] Srinivas Aluru. Quadtrees and octrees. In *Handbook of Data Structures and Applications*, pages 309–326. Chapman and Hall/CRC, 2018.

[7] Fernanda A Andalo, Gabriel Taubin, and Siome Goldenstein. Psqp: Puzzle solving by quadratic programming. *IEEE transactions on pattern analysis and machine intelligence*, 39(2):385–396, 2016.

[8] M Andersen, Joachim Dahl, and Lieven Vandenberghe. CVXOPT: Python software for convex optimization, 2013.

[9] A Angelo. A brief introduction to quadtrees and their applications. In *Style file from the 28th Canadian Conference on Computational Geometry*, 2016.

[10] Alessio Arleo, Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A distributed multilevel force-directed algorithm. In *International Symposium on Graph Drawing and Network Visualization*, pages 3–17. Springer, 2016.

[11] Albert Avinyo, Joan Sola-Morales, and Marta Valencia. On maps with given jacobians involving the heat equation. *Zeitschrift für angewandte Mathematik und Physik ZAMP*, 54(6):919–936, 2003.

[12] S Balaji and M Sundararajan Murugaiyan. Waterfall vs V-Model vs Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2(1):26–30, 2012.

[13] Michael Balzer, Oliver Deussen, and Claus Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM symposium on Software visualization*, pages 165–172, 2005.

[14] Michael J Bannister, David Eppstein, Michael T Goodrich, and Lowell Trott. Force-directed graph drawing using social gravity and scaling. In *International Symposium on Graph Drawing*, pages 414–425. Springer, 2012.

[15] André da Motta Salles Barreto and Helio JC Barbosa. Graph layout using a genetic algorithm. In *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*, pages 179–184. IEEE, 2000.

[16] Michael Behrisch, Michael Blumenschein, Nam Wook Kim, Lin Shao, Mennatallah El-Assady, Johannes Fuchs, Daniel Seebacher, Alexandra Diehl, Ulrik Brandes, Hanspeter Pfister, et al. Quality metrics for information visualization. In *Computer Graphics Forum*, volume 37, pages 625–662. Wiley Online Library, 2018.

[17] Lars Bergstrom and John Reppy. Nested data-parallelism on the gpu. In *Proceedings of the 17th ACM SIGPLAN international conference on Functional programming*, pages 247–258, 2012.

[18] Janis Born, Patrick Schmidt, and Leif Kobbelt. Layout embedding via combinatorial optimization. In *Computer Graphics Forum*, volume 40, pages 277–290. Wiley Online Library, 2021.

[19] Cynthia A Brewer. Cartography: thematic map design. *Cartographic Perspectives*, (17):26–27, 1994.

[20] Govert G Brinkmann, Kristian FD Rietveld, and Frank W Takes. Exploiting gpus for fast force-directed visualization of large-scale networks. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 382–391. IEEE, 2017.

[21] Kevin Buchin, Bettina Speckmann, and Sander Verdonschot. Evolution strategies for optimizing rectangular cartograms. In *International Conference on Geographic Information Science*, pages 29–42. Springer, 2012.

[22] U.S. Census Bureau. State-to-state migration flows. `https://www.census.gov/data/tables/time-series/demo/geographic-mobility/state-to-state-migration.html`. Last Revised Nov 2020.

[23] Rafael G Cano, Kevin Buchin, Thom Castermans, Astrid Pieterse, Willem Sonke, and Bettina Speckmann. Mosaic drawings and cartograms. In *Computer Graphics Forum*, volume 34, pages 361–370. Wiley Online Library, 2015.

[24] Mackinlay Card. *Readings in information visualization: using vision to think.* Morgan Kaufmann, 1999.

[25] Colette Cauvin and C Schneider. Cartographic transformations and the piezopleth maps method. *The Cartographic Journal*, 26(2):96–104, 1989.

[26] Thomas J Cheatham and John H Crenshaw. Object-oriented vs. waterfall software development. In *Proceedings of the 19th annual conference on Computer Science*, pages 595–599, 1991.

[27] Homer H Chen and Thomas S Huang. A survey of construction and manipulation of octrees. *Computer Vision, Graphics, and Image Processing*, 43(3):409–431, 1988.

[28] Federal Election Commission. Federal elections 2012. `https://www.fec.gov/resources/cms-content/documents/federalelections2012.pdf#page=11`, 2013.

[29] Pedro Cruz. Wrongfully right: applications of semantic figurative metaphors in information visualization. *IEEE VIS Arts Program (VISAP)*, pages 14–21, 2015.

[30] Bernard Dacorogna and Jürgen Moser. On a partial differential equation involving the jacobian determinant. In *Annales de l'Institut Henri Poincare (C) Non Linear Analysis*, volume 7, pages 1–26. Elsevier, 1990.

[31] Mehmet Deveci, Christian Trott, and Sivasankaran Rajamanickam. Multithreaded sparse matrix-matrix multiplication for many-core and gpu architectures. *Parallel Computing*, 78:33–46, 2018.

[32] LinLin Ding, Baiyou Qiao, Guoren Wang, and Chen Chen. An efficient quad-tree based index structure for cloud data management. In *International Conference on Web-Age Information Management*, pages 238–250. Springer, 2011.

[33] Ugur Dogrusoz, Erhan Giral, Ahmet Cetintas, Ali Civril, and Emek Demir. A layout algorithm for undirected compound graphs. *Information Sciences*, 179(7):980–994, 2009.

[34] Wenqiang Dong, Xingyu Fu, Guangluan Xu, and Yu Huang. An improved force-directed graph layout algorithm based on aesthetic criteria. *Computing and Visualization in Science*, 16(3):139–149, 2013.

[35] Daniel Dorling. Area cartograms: Their use and creation, vol. 59 of concepts and techniques in modern geography. *University of East Anglia: Environmental Publications*, 2, 1996.

[36] James A Dougenik, Nicholas R Chrisman, and Duane R Niemeyer. An algorithm to construct continuous area cartograms. *The Professional Geographer*, 37(1):75–81, 1985.

[37] A Downton and D Crookes. Parallel architectures for image processing. *Electronics & Communication Engineering Journal*, 10(3):139–151, 1998.

[38] Peter Eades and Mao Lin Huang. Navigating clustered graphs using force-directed methods. In *Graph Algorithms And Applications 2*, pages 191–215. World Scientific, 2004.

[39] Jared Espenant and Debajyoti Mondal. Streamtable: An area proportional visualization for tables with flowing streams. In *European Workshop on Computational Geometry*, page 28:1–28:7, 2021. `https://arxiv.org/abs/2103.15037`.

[40] William Evans, Stefan Felsner, Michael Kaufmann, Stephen G Kobourov, Debajyoti Mondal, Rahnuma Islam Nishat, and Kevin Verbeek. Table cartograms. In *European Symposium on Algorithms*, pages 421–432. Springer, 2013.

[41] William Evans, Stefan Felsner, Michael Kaufmann, Stephen G Kobourov, Debajyoti Mondal, Rahnuma Islam Nishat, and Kevin Verbeek. Table cartogram. *Computational Geometry*, 68:174–185, 2018.

[42] Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[43] R Fletcher. A general quadratic programming algorithm. *IMA Journal of Applied Mathematics*, 7(1):76–91, 1971.

[44] National Science Foundation. Weather research and forecasting model. `https://www.mmm.ucar.edu/weather-research-and-forecasting-model`. Last Accessed Jun 2019.

[45] The Python Software Foundation. Python: multiprocessing- process-based parallelism. `https://docs.python.org/3.4/library/multiprocessing.html`. Last Accessed Jun 2019.

[46] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In *International Symposium on Graph Drawing*, pages 388–403. Springer, 1994.

[47] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[48] Yu-Chen Fu, Zhi-Yong Hu, Wei Guo, and Dong-Ru Zhou. Qr-tree: a hybrid spatial index structure. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, volume 1, pages 459–463. IEEE, 2003.

[49] Pawel Gajer, Michael T Goodrich, and Stephen G Kobourov. A multi-dimensional approach to force-directed layouts of large graphs. In *International Symposium on Graph Drawing*, pages 211–221. Springer, 2000.

[50] Emden R Gansner and Stephen C North. Improved force-directed layouts. In *International Symposium on Graph Drawing*, pages 364–373. Springer, 1998.

[51] Michael T Gastner and Mark EJ Newman. Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences*, 101(20):7499–7504, 2004.

[52] Michael T Gastner, Vivien Seguy, and Pratyush More. Fast flow-based algorithm for creating density-equalizing map projections. *Proceedings of the National Academy of Sciences*, 115(10):E2156–E2164, 2018.

[53] Jennifer Golbeck and Paul Mutton. Spring-embedded graphs for semantic visualization. In *Visualizing the semantic Web*, pages 172–182. Springer, 2006.

[54] Ardeshir Goshtasby. Piecewise linear mapping functions for image registration. *Pattern Recognition*, 19(6):459–466, 1986.

[55] Jonathan L Gross and Jay Yellen. *Handbook of graph theory.* CRC press, 2004.

[56] Sabir M Guseyn-Zade and Vladimir S Tikunov. Numerical methods in the compilation of transformed images. *Mapping Sciences and Remote Sensing*, 31(1):66–85, 1994.

[57] Stefan Hachul and Michael Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *International Symposium on Graph Drawing*, pages 285–295. Springer, 2004.

[58] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2003.

[59] Mohammad Rakib Hasan, Debajyoti Mondal, Jarin Tasnim, and Kevin A Schneider. Putting table cartograms into practice.

[60] Henry Heberle, Marcelo Falsarella Carazzolle, Guilherme P Telles, Gabriela Vaz Meirelles, and Rosane Minghim. Cellnetvis: a web tool for visualization of biological networks using force-directed layout constrained by cellular components. *BMC bioinformatics*, 18(10):25–37, 2017.

[61] Roberto Henriques, Fernando Bação, and Victor Lobo. Carto-som: cartogram creation using self-organizing maps. *International Journal of Geographical Information Science*, 23(4):483–511, 2009.

[62] Clifford Hildreth et al. A quadratic programming procedure. *Naval research logistics quarterly*, 4(1):79–85, 1957.

[63] MHW Hobbs and Peter Rodgers. Representing space: A hybrid genetic algorithm for aesthetic graph layout. In *FEA'98 Frontiers in Evolutionary Algorithms in Proceedings of JCIS'98 The Fourth Joint Conference on Information Sciences*, volume 2, pages 415–418, 1998.

[64] Roger W Hockney and Chris R Jesshope. *Parallel Computers 2: architecture, programming and algorithms.* CRC Press, 2019.

[65] Marius Hogräfer, Magnus Heitzler, and Hans-Jörg Schulz. The state of the art in map-like visualization. *Computer Graphics Forum*, 39, 2020.

[66] Sunpyo Hong and Hyesoon Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 152–163, 2009.

[67] Donald H House and Christopher J Kocmoud. Continuous cartogram construction. In *Proceedings Visualization'98*, pages 197–204. IEEE, 1998.

[68] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica journal*, 10(1):37–71, 2005.

[69] Gregory Michael Hunter. *Efficient computation and data structures for graphics.* Princeton University, 1978.

[70] Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova. Analyses of an agile methodology implementation. In *Proceedings. 30th Euromicro Conference, 2004.*, pages 326–333. IEEE, 2004.

[71] Ryo Inoue and Mao Li. Optimization-based construction of quadrilateral table cartograms. *ISPRS International Journal of Geo-Information*, 9(1):43, 2020.

[72] Ryo Inoue and Eihan Shimizu. A new algorithm for continuous area cartogram construction with triangulation of regions and restriction on bearing changes of edges. *Cartography and Geographic Information Science*, 33(2):115–125, 2006.

[73] Takayuki Itoh, Chris Muelder, Kwan-Liu Ma, and Jun Sese. A hybrid space-filling and force-directed layout method for visualizing multiple-category graphs. In *2009 IEEE Pacific Visualization Symposium*, pages 121–128. IEEE, 2009.

[74] Arpan Jain, Tim Moon, Tom Benson, Hari Subramoni, Sam Adé Jacobs, Dhabaleswar K Panda, and Brian Van Essen. Super: Sub-graph parallelism for transformers. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 629–638. IEEE, 2021.

[75] Stephen Jones. Introduction to dynamic parallelism. In *GPU Technology Conference Presentation S*, volume 338, page 2012, 2012.

[76] Daniel A Keim, Stephen C North, and Christian Panse. Cartodraw: A fast algorithm for generating contiguous cartograms. *IEEE transactions on visualization and computer graphics*, 10(1):95–110, 2004.

[77] Daniel A Keim, Christian Panse, and Stephen C North. Medial-axis-based cartograms. *IEEE computer graphics and applications*, 25(3):60–68, 2005.

[78] Andrew Kennings and Kristofer P Vorwerk. Force-directed methods for generic placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2076–2087, 2006.

[79] Stephen G Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012.

[80] Christopher James Kocmoud. *Constructing continuous cartograms: a constraint-based approach.* PhD thesis, Texas A&M University, 1997.

[81] Mary Rebecca Duquette Krauss. *The relative effectiveness of the noncontiguous cartogram.* PhD thesis, Virginia Tech, 1989.

[82] John Krygier and Denis Wood. Making maps: A visual guide to map design for gis. 2011.

[83] Samuel H Langton and Reka Solymosi. Cartograms, hexograms and regular grids: Minimising misrepresentation in spatial data visualisations. *Environment and Planning B: Urban Analytics and City Science*, 48(2):348–357, 2021.

[84] Cleverson Ledur, Dalvan Griebler, Isabel Manssour, and Luiz Gustavo Fernandes. A high-level dsl for geospatial visualizations with multi-core parallelism support. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 298–304. IEEE, 2017.

[85] Jungwoo Lee. Optimized quadtree for karhunen-loeve transform in multispectral image coding. *IEEE Transactions on Image Processing*, 8(4):453–461, 1999.

[86] Zhuoran Li, Guiling Wang, Jinlong Meng, and Yao Xu. The parallel and precision adaptive method of marine lane extraction based on quadtree. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 170–188. Springer, 2018.

[87] AA Lovelace. Sketch of the Analytic Engine Invented by Charles Babbage by LF Menabrea. With Notes upon the Memoir by the Translator, Ada Augusta Countess of Lovelace. *P. Morrison and E,. Morrison (Eds), Charles Babbage and his Calculating Engines, Dover*, 1961.

[88] Juliane Mai, Kurt C Kornelsen, Bryan A Tolson, Vincent Fortin, Nicolas Gasset, Djamel Bouhemhem, David Schäfer, Michael Leahy, François Anctil, and Paulin Coulibaly. The canadian surface prediction archive (caspar): A platform to enhance environmental modeling in canada and globally. *Bulletin of the American Meteorological Society*, 101(3):E341–E356, 2020.

[89] Environmental Systems Research Institute (ESRI) Map. Non contiguous cartogram. `https://carto.maps.arcgis.com/apps/StorytellingTextLegend/index.html?appid=10fcf7bdaeac466a8662635861923b0f`.

[90] Shawn Martin, W Michael Brown, Richard Klavans, and Kevin W Boyack. Openord: an open-source toolbox for large graph layout. In *Visualization and Data Analysis 2011*, volume 7868, page 786806. International Society for Optics and Photonics, 2011.

[91] Riccardo Mazza. *Introduction to information visualization.* Springer Science & Business Media, 2009.

[92] Bob G McCullouch and Kamolwan Lueprasert. 2d bar-code applications in construction. *Journal of construction engineering and management*, 120(4):739–752, 1994.

[93] Michael J McGuffin. Simple algorithms for network visualization: A tutorial. *Tsinghua Science and Technology*, 17(4):383–398, 2012.

[94] Robert McMaster and Susanna McMaster. A history of twentieth-century american academic cartography. *Cartography and Geographic Information Science*, 29(3):305–321, 2002.

[95] A. McNutt and G. Kindlmann. A minimally constrained optimization algorithm for table cartograms. *IEEEVIS InfoVis Posters*, 2020. OSF Preprints, `https://doi.org/10.31219/osf.io/kem6j`.

[96] Andrew McNutt. What are table cartograms good for anyway? an algebraic analysis. *Eurographics Conference on Visualization (EuroVis)*, 40, 2021. To appear.

[97] Peng Mi, Maoyuan Sun, Moeti Masiane, Yong Cao, and Chris North. Interactive graph layout of a million nodes. In *Informatics*, volume 3, page 23. Multidisciplinary Digital Publishing Institute, 2016.

[98] Antonio CO Miranda and Luiz F Martha. Mesh generation on high-curvature surfaces based on a background quadtree structure. *space*, 500:N2, 2002.

[99] Pouria Mistani, Arthur Guittet, Daniil Bochkov, Joshua Schneider, Dionisios Margetis, Christian Ratsch, and Frederic Gibou. The island dynamics model on parallel quadtree grids. *Journal of Computational Physics*, 361:150–166, 2018.

[100] Alistair Morrison, Greg Ross, and Matthew Chalmers. A hybrid layout algorithm for sub-quadratic multidimensional scaling. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 152–158. IEEE, 2002.

[101] Christopher Mueller, Douglas P Gregor, and Andrew Lumsdaine. Distributed force-directed graph layout and visualization. In *EGPGV@ EuroVis/EGVE*, pages 83–90, 2006.

[102] ZF Muhsin, A Rehman, A Altameem, Tanzila Saba, and M Uddin. Improved quadtree image segmentation approach to region information. *the imaging science journal*, 62(1):56–62, 2014.

[103] Tamara Munzner. Process and pitfalls in writing information visualization research papers. In *Information visualization*, pages 134–153. Springer, 2008.

[104] Mark Newman. Maps of the 2012 us presidential election results. `http://www-personal.umich.edu/~mejn/election/2012/`, 2012.

[105] Andreas Noack. An energy model for visual graph clustering. In *International symposium on graph drawing*, pages 425–436. Springer, 2003.

[106] Andreas Noack. Energy models for graph clustering. *J. Graph Algorithms Appl.*, 11(2):453–480, 2007.

[107] Jamie R Nuñez, Christopher R Anderton, and Ryan S Renslow. Optimizing colormaps with consideration for color vision deficiency to enable accurate interpretation of scientific data. *PloS one*, 13(7):e0199239, 2018.

[108] José Jesús Reyes Nuñez. The use of cartograms in school cartography. In *Thematic cartography for the society*, pages 327–339. Springer, 2014.

[109] Sabrina Nusrat and Stephen Kobourov. The state of the art in cartograms. In *Computer Graphics Forum*, volume 35, pages 619–642. Wiley Online Library, 2016.

[110] Jianxiong Pang and Lynne Blair. Refining feature driven development-a methodology for early aspects. *Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, 86, 2004.

[111] F Pascal and JL Marechal. Fast adaptive quadtree mesh generation. *1998 International Meshing Roundtable*, 1998.

[112] Kai Petersen, Claes Wohlin, and Dejan Baca. The waterfall model in large-scale development. In *International Conference on Product-Focused Software Process Improvement*, pages 386–400. Springer, 2009.

[113] Matt Pharr and William R Mark. ispc: A spmd compiler for high-performance cpu programming. In *2012 Innovative Parallel Computing (InPar)*, pages 1–13. IEEE, 2012.

[114] G Philip. Atlas of canada and the world. *George Philip and Son, Milwaukee*, 4, 1979.

[115] Alain Pitiot, Grégoire Malandain, Eric Bardinet, and Paul M Thompson. Piecewise affine registration of biological images. In *International Workshop on Biomedical Image Registration*, pages 91–101. Springer, 2003.

[116] Helen C Purchase, Natalia Andrienko, Thomas J Jankun-Kelly, and Matthew Ward. Theoretical foundations of information visualization. In *Information Visualization*, pages 46–64. Springer, 2008.

[117] Michael J Quinn. Parallel programming. *TMH CSE*, 526:105, 2003.

[118] Renata G Raidou, M Eduard Gröller, and Martin Eisemann. Relaxing dense scatter plots with pixel-based mappings. *IEEE transactions on visualization and computer graphics*, 25(6):2205–2216, 2019.

[119] Erwin Raisz. The rectangular statistical cartogram. *Geographical Review*, 24(2):292–296, 1934.

[120] Gerhard Ringel. Equiareal graphs. *Contemporary Methods in Graph Theory, BI Wissenschaftsverlag*, pages 503–505, 1990.

[121] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[122] Hanan Samet. Hierarchical spatial data structures. In *Symposium on Large Spatial Databases*, pages 191–212. Springer, 1989.

[123] Hanan Samet and Markku Tamminen. Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE transactions on pattern analysis and machine intelligence*, 10(4):579–586, 1988.

[124] Rodrigo Santamaría, Roberto Therón, and Luis Quintales. Bicoverlapper: a tool for bicluster visualization. *Bioinformatics*, 24(9):1212–1213, 2008.

[125] Ken Schwaber. Scrum development process. In *Business object design and implementation*, pages 117–134. Springer, 1997.

[126] Clifford A Shaffer and Hanan Samet. Optimal quadtree construction algorithms. *Computer Vision, Graphics, and Image Processing*, 37(3):402–419, 1987.

[127] Frank Simon, Frank Steinbrückner, and Claus Lewerentz. *3d-spring embedder for complete graphs.* BTU, Inst. of Computer Science, 2000.

[128] Terry A Slocum, R McMaster, F Kessler, and H Howard. *Thematic cartography and visualization.* Prentice hall Upper Saddle River, NJ, 1999.

[129] Bettina Speckmann, Marc Van Kreveld, and Sander Florisson. A linear programming approach to rectangular cartograms. In *Progress in Spatial Data Handling*, pages 529–546. Springer, 2006.

[130] Marc Streit, Samuel Gratzl, Michael Gillhofer, Andreas Mayr, Andreas Mitterecker, and Sepp Hochreiter. Furby: fuzzy force-directed bicluster visualization. *BMC bioinformatics*, 15(6):1–13, 2014.

[131] Gary J Sullivan and Richard L Baker. Efficient quadtree coding of images and video. *IEEE Transactions on image processing*, 3(3):327–331, 1994.

[132] Shipeng Sun. A fast, free-form rubber-sheet algorithm for contiguous area cartograms. *International Journal of Geographical Information Science*, 27(3):567–593, 2013.

[133] Kevin D Terwilliger, Orin M Ozias, and Scott C Lauffer. Secure information handling system matrix bar code, March 31 2015. US Patent 8,997,241.

[134] Anna Tikhonova and Kwan-Liu Ma. A scalable parallel force-directed graph layout algorithm. In *Proceedings of the 8th Eurographics conference on Parallel Graphics and Visualization*, pages 25–32, 2008.

[135] Waldo Tobler. Thirty five years of computer cartograms. *ANNALS of the Association of American Geographers*, 94(1):58–73, 2004.

[136] Ying Tu and Han-Wei Shen. Visualizing changes of hierarchical data using treemaps. *IEEE transactions on visualization and computer graphics*, 13(6):1286–1293, 2007.

[137] William Thomas Tutte. How to draw a graph. volume 3, pages 743–767. Wiley Online Library, 1963.

[138] Marc Van Kreveld and Bettina Speckmann. On rectangular cartograms. *Computational Geometry*, 37(3):175–187, 2007.

[139] Adriano Vogel, Cassiano Rista, Gabriel Justo, Endrius Ewald, Dalvan Griebler, Gabriele Mencagli, and Luiz Gustavo Fernandes. Parallel stream processing with mpi for video analytics and data visualization. In *Symposium on High Performance Computing Systems*, pages 102–116. Springer, 2018.

[140] Chris Walshaw. A multilevel algorithm for force-directed graph drawing. In *International Symposium on Graph Drawing*, pages 171–182. Springer, 2000.

[141] Yong-Xian Wang, Zong-Zhe Li, Lu Yao, Wei Cao, and Zheng-Hua Wang. Two improved gpu acceleration strategies for force-directed graph layout. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, volume 13, pages V13–132. IEEE, 2010.

[142] Gregory V Wilson. The history of the development of parallel computing. *URL: http://ei. cs. vt. edu/history/Parallel. html*, 1994.

[143] Mark J. Winter. Diffusion cartograms for the display of periodic table data. *Journal of Chemical Education*, 88(11):1507–1510, 2011.

[144] TzuYen Wong, Peter Kovesi, and Amitava Datta. Projective transformations for image transition animations. In *14th International Conference on Image Analysis and Processing (ICIAP 2007)*, pages 493–500, 2007.

[145] Amichai Worms. Matrix barcode system, July 5 2012. US Patent App. 12/983,923.

[146] Jun Wu. Continuous optimization of adaptive quadtree structures. *Computer-Aided Design*, 102:72–82, 2018.

[147] Fang Yan, Yuanjie Zheng, Jinyu Cong, Liu Liu, Dacheng Tao, and Sujuan Hou. Solving jigsaw puzzles via nonconvex quadratic programming with the projected power method. *IEEE Transactions on Multimedia*, 2020.

[148] Mark Yerry and Mark Shephard. A modified quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3(01):39–46, 1983.

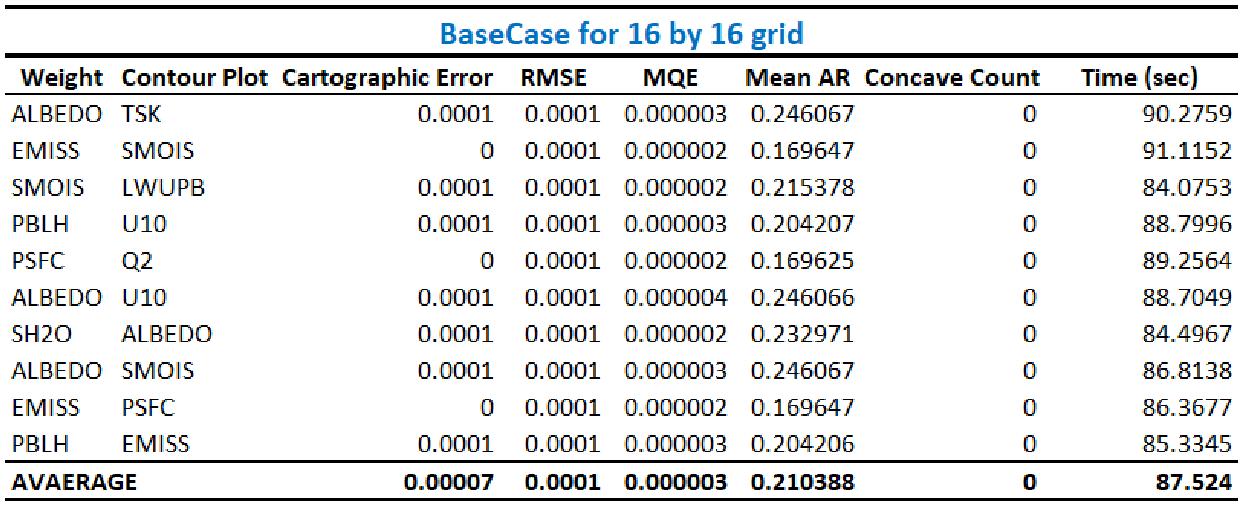
[149] Enas Yunis, Rio Yokota, and Aron Ahmadia. Scalable force directed graph layout algorithms using fast multipole methods. In *2012 11th International Symposium on Parallel and Distributed Computing*, pages 180–187. IEEE, 2012.

[150] Guangtao Zhai, Weisi Lin, Jianfei Cai, Xiaokang Yang, and Wenjun Zhang. Efficient quadtree based block-shift filtering for deblocking and deringing. *Journal of Visual Communication and Image Representation*, 20(8):595–607, 2009.

[151] XG Zhou and HS Wang. A quadtree spatial index method with inclusion relations for the incremental updating of vector landcover database. *The International Archives of the Photogrammetry, Remote Sensing and Spatial, Information Sciences*, 42:4, 2018.

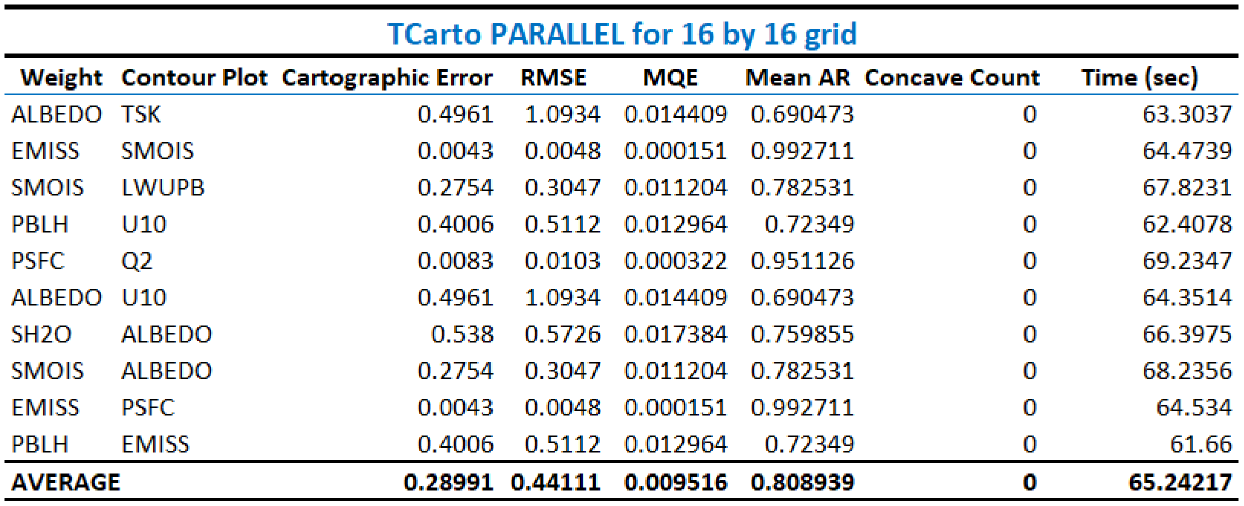
Appendix A

Generated Results with Weather Dataset



BaseCase for 16 by 16 grid

Weight	Contour Plot	Cartographic Error	RMSE	MQE	Mean AR	Concave Count	Time (sec)
ALBEDO	TSK	0.0001	0.0001	0.000003	0.246067	0	90.2759
EMISS	SMOIS	0	0.0001	0.000002	0.169647	0	91.1152
SMOIS	LWUPB	0.0001	0.0001	0.000002	0.215378	0	84.0753
PBLH	U10	0.0001	0.0001	0.000003	0.204207	0	88.7996
PSFC	Q2	0	0.0001	0.000002	0.169625	0	89.2564
ALBEDO	U10	0.0001	0.0001	0.000004	0.246066	0	88.7049
SH2O	ALBEDO	0.0001	0.0001	0.000002	0.232971	0	84.4967
ALBEDO	SMOIS	0.0001	0.0001	0.000003	0.246067	0	86.8138
EMISS	PSFC	0	0.0001	0.000002	0.169647	0	86.3677
PBLH	EMISS	0.0001	0.0001	0.000003	0.204206	0	85.3345
AVAERAGE		**0.00007**	**0.0001**	**0.000003**	**0.210388**	**0**	**87.524**



TCarto PARALLEL for 16 by 16 grid

Weight	Contour Plot	Cartographic Error	RMSE	MQE	Mean AR	Concave Count	Time (sec)
ALBEDO	TSK	0.4961	1.0934	0.014409	0.690473	0	63.3037
EMISS	SMOIS	0.0043	0.0048	0.000151	0.992711	0	64.4739
SMOIS	LWUPB	0.2754	0.3047	0.011204	0.782531	0	67.8231
PBLH	U10	0.4006	0.5112	0.012964	0.72349	0	62.4078
PSFC	Q2	0.0083	0.0103	0.000322	0.951126	0	69.2347
ALBEDO	U10	0.4961	1.0934	0.014409	0.690473	0	64.3514
SH2O	ALBEDO	0.538	0.5726	0.017384	0.759855	0	66.3975
SMOIS	ALBEDO	0.2754	0.3047	0.011204	0.782531	0	68.2356
EMISS	PSFC	0.0043	0.0048	0.000151	0.992711	0	64.534
PBLH	EMISS	0.4006	0.5112	0.012964	0.72349	0	61.66
AVERAGE		**0.28991**	**0.44111**	**0.009516**	**0.808939**	**0**	**65.24217**

Table A.1: Error and processing time for different table cartograms with the Weather Research and Forecasting (WRF) dataset, (top) **Baseline** algorithm with 16×16 grid, (bottom) **TCarto Parallel** algorithm with 16×16 grid [88].

TCarto DIVCON for 16 by 16 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.0325	0.0571	0.001660	0.662008	0	35.1892
EMISS	SMOIS	0.0032	0.0035	0.000110	0.992193	0	35.2239
SMOIS	LWUPB	0.0136	0.017	0.000528	0.737363	0	35.0065
PBLH	U10	0.0706	0.1193	0.003777	0.653522	0	34.8351
PSFC	Q2	0.003	0.0036	0.000111	0.95761	0	35.3811
ALBEDO	U10	0.0325	0.0571	0.001660	0.662008	0	35.3962
SH2O	ALBEDO	0.0993	0.1279	0.003885	0.619437	0	32.2893
SMOIS	ALBEDO	0.0136	0.017	0.000528	0.737363	0	35.5939
EMISS	PSFC	0.0032	0.0035	0.000110	0.992193	0	34.9152
PBLH	EMISS	0.0706	0.1193	0.003777	0.653522	0	34.7602
AVERAGE		**0.03421**	**0.05253**	**0.001615**	**0.766722**	**0**	**34.85906**

FastFlow for 16 by 16 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.0068	0.01013	0.000317	0.744372	0	
EMISS	SMOIS	0.0038	0.00406	0.000127	0.993098	0	
SMOIS	LWUPB	0.0105	0.01605	0.000500	0.755962	0	
PBLH	U10	0.0087	0.01172	0.000366	0.656382	0	
PSFC	Q2	0.003	0.0037	0.000116	0.97435	0	
ALBEDO	U10	0.0068	0.01013	0.000317	0.744372	0	
SH2O	ALBEDO	0.0137	0.02209	0.000686	0.702215	0	
SMOIS	ALBEDO	0.0105	0.01605	0.000500	0.755962	0	
EMISS	PSFC	0.0038	0.00406	0.000127	0.993098	0	
PBLH	EMISS	0.0087	0.01172	0.000366	0.656382	0	
AVERAGE		**0.00763**	**0.01097**	**0.000342**	**0.797619**	**0**	

Table A.2: Error and processing time for different table cartograms with the Weather Research and Forecasting (WRF) dataset, (top) **TCarto DivCon** algorithm with 16×16 grid, (bottom) **FastFlow** algorithm with 16×16 grid [88].

BaseCase for 32 by 32 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0	0.0001	0.000001	0.12391	0	646.6317
EMISS	SMOIS	0	0.0001	0.000001	0.105403	0	400.5113
SMOIS	LWUPB	0	0.0001	0.000001	0.118004	0	402.4248
PBLH	U10	0.0001	0.0001	0.000002	0.116359	0	443.3352
PSFC	Q2	0	0.0001	0.000001	0.105135	0	456.6903
ALBEDO	U10	0	0.0001	0.000001	0.12391	0	403.6109
SH2O	ALBEDO	0.0001	0.0001	0.000001	0.168931	0	433.975
ALBEDO	SMOIS	0.0001	0.0001	0.000001	0.12391	0	415.5731
EMISS	PSFC	0	0.0001	0.000001	0.105403	0	404.9429
PBLH	EMISS	0.0001	0.0001	0.000002	0.116359	0	395.3356
AVAERAGE		**0.00004**	**0.0001**	**0.000001**	**0.120732**	**0**	**440.30308**

TCarto PARALLEL for 32 by 32 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.1392	0.1937	0.002975	0.783234	0	203.6473
EMISS	SMOIS	0.0061	0.0075	0.000118	0.994482	0	183.0774
SMOIS	LWUPB	0.148	0.1943	0.004033	0.877615	0	188.3193
PBLH	U10	0.5662	0.8766	0.008612	0.720298	0	215.3847
PSFC	Q2	0.0247	0.0286	0.000448	0.953498	0	180.9087
ALBEDO	U10	0.1392	0.1937	0.002975	0.783234	0	207.8594
SH2O	ALBEDO	0.5948	0.685	0.009307	0.776501	0	202.8882
ALBEDO	SMOIS	0.1392	0.1937	0.002975	0.783234	0	206.1995
EMISS	PSFC	0.0061	0.0075	0.000118	0.994482	0	184.9958
PBLH	EMISS	0.5662	0.8766	0.008612	0.720298	0	190.68
AVAERAGE		**0.23297**	**0.32572**	**0.004017**	**0.838688**	**0**	**196.39603**

Table A.3: Error and processing time for different table cartograms with the Weather Research and Forecasting (WRF) dataset, (top) **Baseline** algorithm with 32 × 32 grid, (bottom) **TCarto Parallel** algorithm with 32 × 32 grid [88].

TCarto DIVCON for 32 by 32 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.0077	0.01	0.000155	0.84988	0	171.1602
EMISS	SMOIS	0.0035	0.0037	0.000058	0.992336	0	88.3915
SMOIS	LWUPB	0.0069	0.00014	0.000136	0.845219	0	89.7782
PBLH	U10	0.0771	0.1303	0.002305	0.643813	0	101.4747
PSFC	Q2	0.0032	0.0039	0.000061	0.95627	0	116.1137
ALBEDO	U10	0.0077	0.01	0.000155	0.84988	0	91.082
SH2O	ALBEDO	0.0961	0.1346	0.002205	0.612958	0	99.1374
ALBEDO	SMOIS	0.0077	0.01	0.000155	0.84988	0	89.8392
EMISS	PSFC	0.0035	0.0037	0.000058	0.992336	0	89.9376
PBLH	EMISS	0.0771	0.1303	0.002305	0.643813	0	86.1847
AVAERAGE		**0.02905**	**0.04366**	**0.000759**	**0.823639**	**0**	**102.30992**

FastFlow for 32 by 32 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.0032	0.00425	0.000066	0.885417	0	
EMISS	SMOIS	0.0037	0.004	0.000062	0.993086	0	
SMOIS	LWUPB	0.0041	0.00529	0.000083	0.85523	0	
PBLH	U10	0.0144	0.02023	0.000315	0.724882	2	
PSFC	Q2	0.0029	0.00346	0.000054	0.971302	0	
ALBEDO	U10	0.0032	0.00425	0.000066	0.885417	0	
SH2O	ALBEDO	0.0111	0.01602	0.000250	0.671074	3	
ALBEDO	SMOIS	0.0041	0.00529	0.000083	0.85523	0	
EMISS	PSFC	0.0037	0.004	0.000062	0.993086	0	
PBLH	EMISS	0.0144	0.02023	0.000315	0.724882	2	
AVAERAGE		**0.00648**	**0.0087**	**0.000136**	**0.855961**	**0.7**	

Table A.4: Error and processing time for different table cartograms with the Weather Research and Forecasting (WRF) dataset, (top) **TCarto DivCon** algorithm with 32×32 grid, (bottom) **FastFlow** algorithm with 32×32 grid [88].

BaseCase for 64 by 64 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.0001	0.0001	0.000001	0.145531	0	1411.6986
EMISS	SMOIS	0	0.0001	0.000000	0.063268	0	1415.6552
SMOIS	LWUPB	0.0001	0.0001	0.000001	0.151246	0	1439.5124
PBLH	U10	0.0001	0.0002	0.000002	0.112502	0	1476.9337
PSFC	Q2	0	0.0001	0.000000	0.065153	0	1523.3196
ALBEDO	U10	0.0001	0.0002	0.000001	0.145531	0	1400.7231
SH2O	ALBEDO	0.0001	0.0001	0.000001	0.157166	0	1386.6986
ALBEDO	SMOIS	0.0001	0.0002	0.000001	0.145531	0	1506.4809
EMISS	PSFC	0	0.0001	0.000000	0.063268	0	1396.4835
PBLH	EMISS	0.0001	0.0002	0.000002	0.112502	0	1445.8187
AVAERAGE		**0.00007**	**0.00014**	**0.000001**	**0.1161698**	**0**	**1440.33243**

TCarto PARALLEL for 64 by 64 grid							
Weight	**Contour Plot**	**Cartographic Error**	**RMSE**	**MQE**	**Mean AR**	**Concave Count**	**Time (sec)**
ALBEDO	TSK	0.9397	1.9667	0.005088	0.728804	0	368.1508
EMISS	SMOIS	0.0075	0.0098	0.000077	0.995572	0	544.5627
SMOIS	LWUPB	0.3447	0.3811	0.003351	0.84153	0	497.0566
PBLH	U10	0.8358	1.7899	0.005117	0.721424	0	422.0614
PSFC	Q2	0.0386	0.0436	0.000339	0.959108	0	532.8095
ALBEDO	U10	0.9397	1.9667	0.005088	0.728804	0	369.5962
SH2O	ALBEDO	0.6801	0.8329	0.005030	0.806353	0	450.3295
SMOIS	ALBEDO	0.3447	0.3811	0.003351	0.84153	0	499.3773
EMISS	PSFC	0.0075	0.0098	0.000077	0.995572	0	543.8422
PBLH	EMISS	0.8358	1.7899	0.005117	0.721424	0	422.8433
AVAERAGE		**0.49741**	**0.91715**	**0.003264**	**0.8340121**	**0**	**465.06295**

Table A.5: Error and processing time for different table cartograms with the Weather Research and Forecasting (WRF) dataset, (top) **Baseline** algorithm with 64×64 grid, (bottom) **TCarto Parallel** algorithm with 64×64 grid [88].

TCarto DIVCON for 64 by 64 grid

Weight	Contour Plot	Cartographic Error	RMSE	MQE	Mean AR	Concave Count	Time (sec)
ALBEDO	TSK	0.023	0.0429	0.000334	0.653317	0	254.6153
EMISS	SMOIS	0.0035	0.0037	0.000029	0.991587	0	266.105
SMOIS	LWUPB	0.0147	0.0209	0.000162	0.730771	0	245.6336
PBLH	U10	0.0906	0.2032	0.001433	0.628697	0	241.8784
PSFC	Q2	0.0031	0.0038	0.000030	0.955271	0	241.9801
ALBEDO	U10	0.023	0.0429	0.000334	0.653317	0	251.0729
SH2O	ALBEDO	0.0965	0.1593	0.001281	0.598307	0	253.9868
SMOIS	ALBEDO	0.0147	0.0209	0.000162	0.730771	0	250.2475
EMISS	PSFC	0.0035	0.0037	0.000029	0.991587	0	259.433
PBLH	EMISS	0.0906	0.2032	0.001433	0.628697	0	242.325
AVAERAGE		**0.03632**	**0.07045**	**0.000523**	**0.7562322**	**0**	**250.72776**

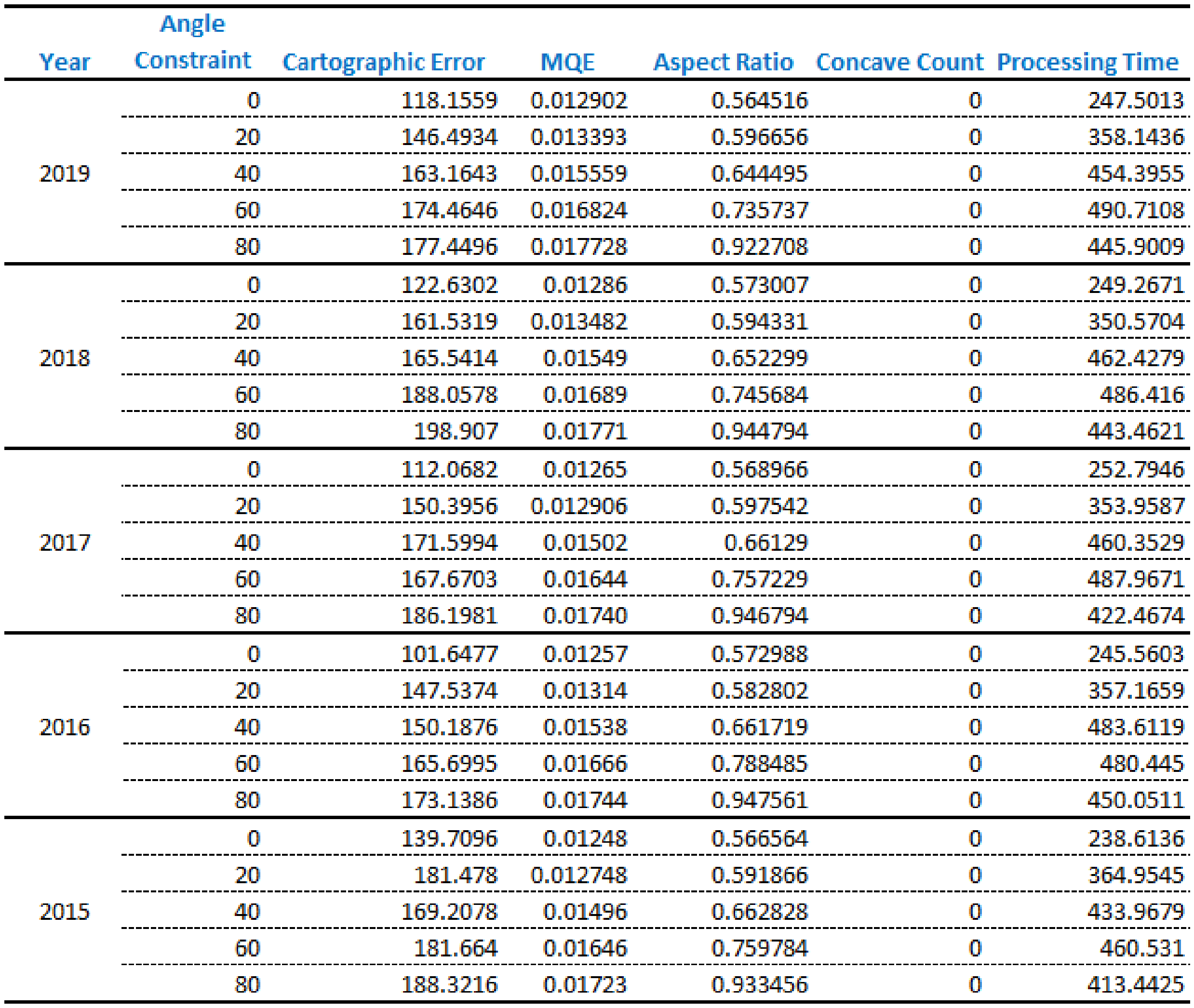
FastFlow for 64 by 64 grid

Weight	Contour Plot	Cartographic Error	RMSE	MQE	Mean AR	Concave Count	Time (sec)
ALBEDO	TSK	0.0073	0.011488	0.000090	0.737358	1	
EMISS	SMOIS	0.004	0.004171	0.000033	0.993309	0	
SMOIS	LWUPB	0.0071	0.010012	0.000078	0.749869	0	
PBLH	U10	0.0163	0.026975	0.000210	0.640149	23	
PSFC	Q2	0.0027	0.003236	0.000025	0.972509	0	
ALBEDO	U10	0.0073	0.011488	0.000090	0.737358	1	
SH2O	ALBEDO	0.0096	0.013809	0.000107	0.696102	2	
SMOIS	ALBEDO	0.0071	0.010012	0.000078	0.749869	0	
EMISS	PSFC	0.004	0.004171	0.000033	0.993309	0	
PBLH	EMISS	0.0163	0.026975	0.000210	0.640149	23	
AVAERAGE		**0.00817**	**0.012234**	**0.000095**	**0.7909981**	**5**	

Table A.6: Error and processing time for different table cartograms with the Weather Research and Forecasting (WRF) dataset, (top) **TCarto DivCon** algorithm with 64×64 grid, (bottom) **FastFlow** algorithm with 64×64 grid [88].

Appendix B

Generated Results with US Migration Data for Angle-Constrained Table Cartogram



Year	Angle Constraint	Cartographic Error	MQE	Aspect Ratio	Concave Count	Processing Time
2019	0	118.1559	0.012902	0.564516	0	247.5013
	20	146.4934	0.013393	0.596656	0	358.1436
	40	163.1643	0.015559	0.644495	0	454.3955
	60	174.4646	0.016824	0.735737	0	490.7108
	80	177.4496	0.017728	0.922708	0	445.9009
2018	0	122.6302	0.01286	0.573007	0	249.2671
	20	161.5319	0.013482	0.594331	0	350.5704
	40	165.5414	0.01549	0.652299	0	462.4279
	60	188.0578	0.01689	0.745684	0	486.416
	80	198.907	0.01771	0.944794	0	443.4621
2017	0	112.0682	0.01265	0.568966	0	252.7946
	20	150.3956	0.012906	0.597542	0	353.9587
	40	171.5994	0.01502	0.66129	0	460.3529
	60	167.6703	0.01644	0.757229	0	487.9671
	80	186.1981	0.01740	0.946794	0	422.4674
2016	0	101.6477	0.01257	0.572988	0	245.5603
	20	147.5374	0.01314	0.582802	0	357.1659
	40	150.1876	0.01538	0.661719	0	483.6119
	60	165.6995	0.01666	0.788485	0	480.445
	80	173.1386	0.01744	0.947561	0	450.0511
2015	0	139.7096	0.01248	0.566564	0	238.6136
	20	181.478	0.012748	0.591866	0	364.9545
	40	169.2078	0.01496	0.662828	0	433.9679
	60	181.664	0.01646	0.759784	0	460.531
	80	188.3216	0.01723	0.933456	0	413.4425

Table B.1: Different measurement metrics for angle (i.e., 0, 20, 40, 60, 80 degree) constraint with US migration data [22].