# A DYNAMIC APPROACH FOR VISUALIZING LOCAL AND GLOBAL INFORMATION IN GEO-SPATIAL NETWORK VISUALIZATIONS

by

Lingbo Zou

Submitted in partial fulfilment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2016

## Table of Contents

## List of Tables

## List of Figures

## Abstract

Geo-spatial network visualization is a fusion of visualization techniques and geographic information science, and has progressed rapidly in recent years. However, challenges still remain regarding the clarity of high density node-link graphs.

We present a dynamic approach for revealing the underlying information in locally cluttered areas while maintaining global edge trends around them, by creating and positioning multiple radial layouts within a geo-spatial connected graph. Moreover, two time series data-flow visualization approaches at both local and global scales are proposed respectively, namely: the peapod model and the hub model. The peapod model focuses on data flows within the local area while the hub model addresses the relations between groups of nodes across the graph. The approach is further refined by leveraging integrated circular layouts, stacked graphs, and word cloud techniques in several complementary views to support the exploration of categorical and aggregated data.

In addition to adhering to emergent design standards for geo-spatial visualization, the computational complexity and quantitative performance analysis on three different datasets were subsequently conducted to examine the scalability of the visualization model. The simulation results show that the main algorithms in our approach are able to achieve acceptable performance in real world test cases. Finally, our model's effectiveness are demonstrated by two significant case studies in different application fields, namely: US aero traffic network and GitHub collaboration graph. The case studies show that insightful knowledge can be discovered by the utilization of the dynamic approach and the support of various complementary features.

## Acknowledgements

First, I would like to express my deepest gratitude to my supervisor Dr. Stephen Brooks. His talented imagination and beautiful ideas always enlighten me when I needed help. His dedication and careful effort in revising my thesis and papers has assisted me endlessly. I appreciate his patience and understanding.

I want to thank all my lab members, especially Xihe Gao, who taught me not only about research but also the philosophy of life. Thanks to Bo Liu, Xiaoting Hong, Zezi Ai and Mohamad Salimian, for walking the greatest time in my life with me. Thanks also to Dr. Dirk Arnold and his student Ang Lu, whose diligence and responsibility kept me engaged.

Thanks are given to my thesis committee: Dr. Sageev Oore and Dr. Alex Brodsky, for their insightful comments and questions.

Finally, I would like to thank my dearest parents, for their endless love throughout my life.

# Chapter 1

# Introduction

This chapter provides an overview of the thesis. First, the motivation and background of our visualization model are provided. Then, the main contributions are summarized in brief. Finally, we introduce the structure and outline of the remainder of the thesis.

## 1.1 Overview

### 1.1.1 Geo-spatial Visualization Overview

Geo-visualization is a mixture of visualization techniques and geographic information science [1]. It visualizes geologically based information with elements such as nodes, edges, areas and text messages, and it is an essential research direction of information visualization in general. In particular, it is becoming an increasingly important area with the continuous development of software intelligence and information science in transportation related industries.

Geo-visualization systems is a research area of both breadth and depth. Many academic researchers have been focusing on visualizing geo-spatial content in recent years as well, and have achieved significant progress [2, 3, 4, 5, 6, 7, 8, 9]. Figure 1.1 shows an array of various geo-spatial visualization systems with different emphasis and focuses [2, 10, 11, 12, 13, 14]. This is simply shown to convey the variety of techniques that have been developed.

### 1.1.2 Challenges in Geo-spatial Network Visualization

Although geo-spatial visualization technologies have progressed rapidly in recent years, many challenges are still remaining [15, 16]. We summarize three typical areas that require more attention as follows.

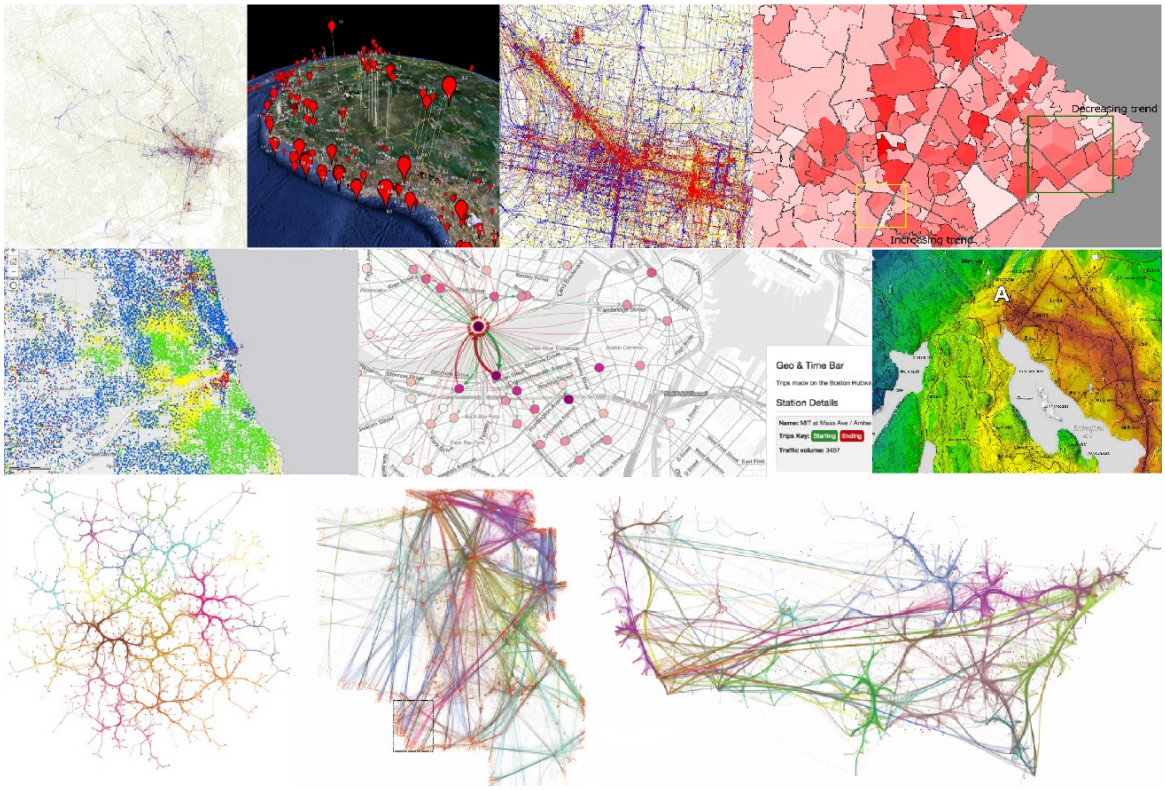



Figure 1.1: An array of various geo-spatial visualization systems with different emphasis and focuses.

First, many current geo-spatial visualization systems still lack human-centered decision making support [15]. This implies that many systems are not sufficiently supportive of data mining or illuminating the underlying information behind the data. That means, we need to develop more flexible and representational elements and forms [80] beyond nodes, edges and areas.

Second, traditional techniques often focus on static representations, for example, viewing data glyphs or graphs on a static map. However, the trend has been increasingly innovated with more dynamic and flexible tools [61].

Third, the most significant motivation for the current work derives from the existing drawbacks of current large-scale network visualization systems. One of the most difficult challenges in graph theory is the well-known clutter problem [17, 18]. The clutter problem is an important phenomenon that interferes in the process of searching for an individual item in a large graph [17]. In most geo-spatial network systems, clutter also happens frequently, and the clutter is often related to two aspects, namely, label clutter and edge

clutter. For example, Figure 1.2 demonstrates a typical case of clutter in a recent research system, which is where our initial motivation stems from.

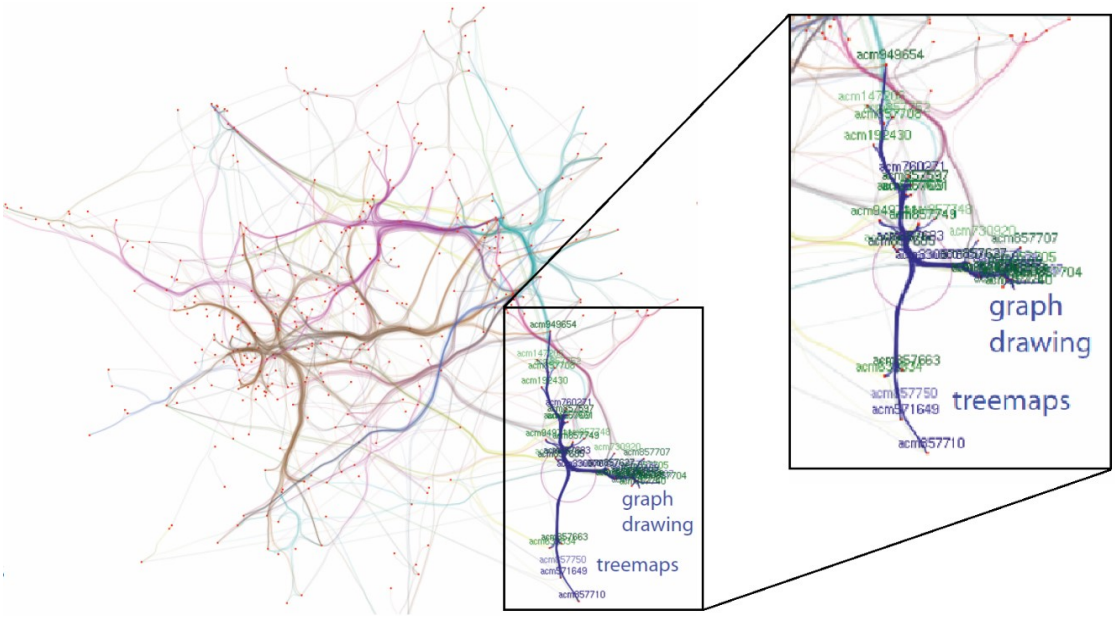



Figure 1.2: An example of label clutter problem in graph network visualization.

Figure 1.2 illustrates how the problem occurs. The main picture left shows a citations graph from literature [19] in which they use a FBEB algorithm [19] to solve edge clutter problems. We can see from the picture that there remains label clutter on the right bottom corner. We show a zoomed in view of the same clutter area on the right side to make it clear. Although we zoom the local area by 4 times, they are still overlapped and unreadable. So, the main problem is that when identifying individual labels is necessary, or when making a single node selection, the traditional layout does not meet the requirement of clarity. We can imagine that, when the data volume grows, the situation would become worse.

Researchers have been developing approaches to address clutter problems for decades. Several widely used techniques can help reducing the clutter to some extent [19, 20, 21, 22, 23, 24, 25, 26, 27, 10]. However, they each have their own restrictions and limitations, especially when utilized in geo-spatial networks. Therefore, research on new approaches for reducing visual clutter and novel dynamic techniques for browsing geo-spatial content remains necessary. We will discuss some typical traditional approaches for reducing visual clutter in more detail in Chapter 2.

### 1.1.3 Standards and Evaluation of Geo-spatial Network Systems

Although there are compelling systems in the recent years from academia [31, 32, 33, 34] that enhance various aspects of geo-spatial visualization such as trail presentation, anomaly detection and edge bundling, we believe that improvements and novel features can still be made, given the challenges stated in the previous section.

To ensure that our model adheres to the findings and design guidelines derived from prior work, while trying to resolve the challenges we stated, we refer to several representative papers regarding various aspects of geo-spatial systems, and we follow some established objectives when designing and implementing the visualization framework. Specifically, the guidelines for an informative large-scale geo-spatial network browsing system include the following:

- A system for visualizing geo-spatial networks should focus on both global and local information [81, 82]. The global picture is useful when we want to observe trends in large areas, while the local layout may strengthen the understanding of specific information in the area of interest.
- As one of the features of network systems, clutter exists in many situations, as shown in Figure 1.2. The system should facilitate the reduction of clutter particularly when local information is needed to be displayed [19, 22, 10, 31].
- The performance of a geo-spatial network system may degrade rapidly when the data size increases [29, 30]. So, effective and efficient algorithms and graph transitions should be considered carefully. Moreover, the system should have the ability to adapt to different data sets.
- The interface itself should be uncluttered and visually efficient, with the color scheme and elements following aesthetical and perceptual standards [80].
- The system should allow operation in alternate ways, and provide various control options [31]. In other words, the large scale network visualizations should be dynamic and flexible [32].
- The system language and protocols should be easy to share [28], in line with established technologies of the Internet. It should employ the most up-to-date frameworks to fit the growing volume of data and information.

We have endeavored to follow the standards defined above when designing and evaluating features to ensure that our system would satisfy the requirements that modern industry and future visualization scenarios may need.

In terms of the evaluation, there is one study that is worth highlighting is from Lam et al. [83] who did an extensive literature review of over 800 visualization publications to summarize evaluation approaches. Given the nature of these complex visualization systems they identified and argued for the benefits of a diversity of evaluation approaches, and reached the conclusion that there is no "best practice" to evaluate visualization systems and researchers should focus more on "goals" and "outputs" instead of "methods" [83]. That means that researchers should select and combine the specific appropriate approaches carefully according to specific models and goals.

In order to choose a proper evaluation scheme, as suggested in the literature [83], we examine our model and refer to two prominent categories of research papers, and leverage aspects from both categories, since we study a cross-cutting area between them. One category is general information visualization systems with a wide range of applications, the other is geo-information visualization frameworks. To provide some context for evaluation, we list several typical papers for both categories as follows.

The authors in the first category evaluate their model by a variety of approaches, and there exist no uniform evaluation criteria [83]. For example, EcoRiver [84] addresses the visual analysis of topic competition in social media and they examined two significant case studies as the evaluation to demonstrate the usefulness of their model, and they further conducted an interview session with two domain experts to collect feedback. Whisper [85] is a framework to visualize the spatiotemporal process of information diffusion on Twitter data sets, they not only examine case studies in the evaluation, but also perform a quantitative performance evaluation of the line-drawing algorithm to trace the pathways of retweets.

The second set of papers prefer to analyze their system by case studies and quantitative analysis. For example, literature [61] proposes a dynamic approach for visualizing multiscale flight data, and they studied the scalability aspect of their system in a fixed environment, since they applied high dimensional data to their model. Pang proposed an approach to visualize uncertainty in geo-spatial data [7], and evaluated the approach by

giving a detailed description of the data and providing several demonstrative data simulation results.

Taking the types of papers into consideration which most resemble the work of our own, as well as considering the features we propose, we determined that a combination of case studies and quantitative analysis would provide an appropriate evaluation for our model. We will state our evaluation results in Chapter 4, Chapter 5 and Chapter 6.

## 1.2 Contributions

First, we propose a novel dynamic approach for reducing clutter issues in geo-spatial visualization systems that incorporates circular layouts in local areas, edge bending and edge arrangement. The advantage of this approach is that we avoid overlaps and occlusion in the area of interest while maintaining the geo-spatial information to a large extent, and revealing underlying information by manipulating multiple layouts simultaneously. We also design algorithms for the better positioning of all related nodes and edges.

Second, we propose a "peapod model" for browsing local time series data flows between related nodes in one specific circular layout. A "peapod" is a visual element that connects two individual nodes on a circular layout to show flow information between them. It is a stream-like element with slender shapes on both ends with a peapod metaphor. The benefit of the peapod model is that flow information between two nodes can be readily interpreted. Furthermore, a complementary "hub stream view" that utilizes the stream graph visualization approach is implemented to visualize categorical and aggregated information about the current selection.

Third, we propose a visual design called "hub model" to examine connections between distinct groups of items. At the risk of mixing metaphors, we introduce the circular layouts as "hubs" located in a large network, and these "hubs" are connected by "timing belts" reminiscent of a "pulley system" found in automobile engines. In this context, we use "timing belts" as a metaphor for aggregated directional data flows between "hubs". This interesting layout can highlight the aggregated data flow information between multiple distinct groups of items visually, while maintaining full connectivity locally within hubs.

Moreover, a complementary “ensemble hub” is provided as a separate view to show individual direct connections between groups of hub nodes.

Finally, we augment these visualization designs with additional features to improve the overall flexibility of the system. Visual glyphs cues for the original position of the nodes on the circular layouts are provided on demand to make up the limitation brought by altering node layouts. Tag cloud views are also incorporated into the interface to demonstrate what items are included in the current selection with different levels of aggregation as well. Moreover, a detailed view that shows specific information about individual links is developed to present precise query results.

## 1.3 Thesis Outline

The remainder of this thesis is organized as follows. A brief introduction of recent theories and approaches related to our work are listed and discussed in Chapter 2. We divide the various methods into categories and discuss the merits and limitations for each of them. Chapter 3 illustrates the main proposed model and algorithm for visualizing large-scale geo-spatial networks. Our quantitative evaluation of the model is described in Chapter 4. Moreover, two case studies that utilize different data sets are examined in Chapter 5 and Chapter 6, separately. In Chapter 7, we summarize the thesis while highlighting the contributions and remaining drawbacks, and suggesting possible directions for the future work.

# Chapter 2

# Related Work

This chapter introduces and discusses methods that were proposed for visualizing and exploring large-scale geo-spatial connected networks. We start with a set of approaches for reducing visual clutter in networks with a high edge density or data volume in Section 2.1. Then we move to some interesting frameworks for browsing geo-spatial connected graphs with user interaction or automated algorithms in Section 2.2. Finally, in Section 2.3, we discuss some recent techniques for visualizing data flows between items.

The reason why we review these technologies is because our approach intersects these multiple fields and we propose a visualization design that can reduce clutter in geo-spatial connected networks through a mixture of interaction and automatic node layouts. Given the extent of related work we focus on those approaches that are most similar to the present work in some key aspect. Further, each of these existing approaches have their own limitations, and we will discuss them in detail in this chapter.

## 2.1 Visual Clutter Reduction Approaches

Graph networks are a widely used method for demonstrating relationships among data items with nodes and edges. However, the data sets are becoming increasingly large and complex, where excessive edges and nodes often lead to a heavy overlap in at least some regions [21]. We refer to this kind of phenomenon as a visual clutter problem.

Many attempts have been made to reduce the visual clutter problem in recent years [22]. They can usually be divided to 4 categories, namely: content filtering, layout optimization, interaction techniques, and magnification [22]. Some of them may combine several of the above techniques together; however, the fundamental elements of visual clutter reduction approaches has not changed significantly over time.

### 2.1.1 Content Filtering

Content filtering is arguably the simplest approach to solve the clutter problem. This approach removes the information that is currently not important to the participants. The information removed may often include a group of nodes and edges in a graph, or even a complete section of visual elements. Therefore, the data that is "important" can stand out in such an area due to the elimination operation. This approach can deal with small scale clutter problems well and is widely used in a large number of visualization frameworks by combining other techniques at the same time. For example, Figure 2.1 illustrates a filtering operation combined with a bubble chart applied in a network bandwidth visualizing system [37]. The user filters out all bandwidth actions consumed by web browsing.

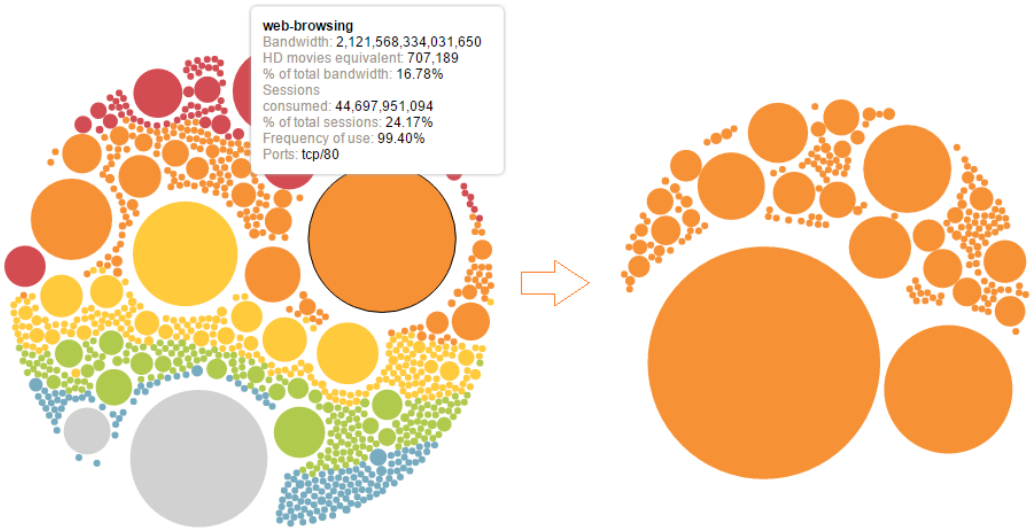



Figure 2.1: Filtering operation combined with a bubble chart.

Researchers have been proposing useful systems that employ content filtering since the early 1990s. Consens et al. discussed visual query operations in scientific visualization from the point of view of data manipulation and database systems. They drew a conclusion that naive approaches do not scale up for graph visualizations, and built a visual query system to map data to visualization results by defining queries directly on the data bases [35]. Another useful tool was built by Mukherjea and Foley [36] to support content and structure analysis for overview diagrams. Filtering is one of the four strategies that are

proposed for visualizing overview diagrams in their work and they demonstrated that filtering is a good technique to be combined with other approaches in visualization systems.

The advantage of filtering is that the visual impact is high due to a simple selection operation, and the operation is relatively easy to understand by inexperienced people. The limitations are also obvious [22, 35, 36]. First, it is difficult to predetermine the importance of a certain aspect of the data, which means we cannot always judge them to be "important" or "not important" very easily. So, the formulation of the filtering thresholds may require extra effort and thought. Second, filtering can potentially cause information loss to some extent because we hide items. Third, filtering operations can interfere with or remove the context. We can lose an understanding of how the items relate with others and this may be misleading [22]. Therefore, content filtering should be used as a supplemental approach to other methods rather than taking the main role itself. We adopt the filtering approach in our designs as well to improve the visual clutter problem for "peapod model".

### 2.1.2 Layout Optimization

The other important approach is layout optimization. Generally, layouts can be optimized based on both node and edge perspectives. Several computational layout adjustment methods were proposed for solving edge congestion problems [39, 40, 41]. However, the computational methods share the same drawback: optimal solutions are difficult to determine [22]. Node layout optimization has been a more active research area than edge layouts in recent years, and several papers regarding node layout optimization appear [38, 42, 43, 44, 45].

Henry et al. put forward a visualization framework for network visualization called "NodeTrix" [38]. They aimed at visualizing local community relationships in social networks, and proposed a set of interaction techniques to generate node matrices for local communities [38]. There are two main contributions in this work, one is a set of interaction techniques, and the other is a hybrid representation for local and global information. They successfully combined node-link graphs with adjacency matrices to analyze both global and local structures. Figure 2.2 illustrates the "InfoVis Co-authorship Network" case study result of their work.

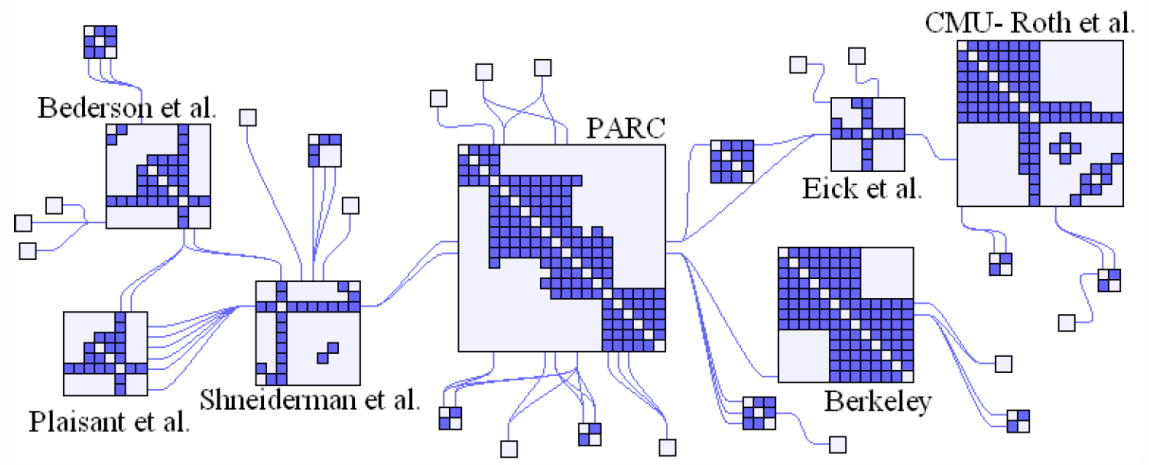



Figure 2.2: NodeTrix Representation of the largest component of the InfoVis Co-authorship Network.

However, there are limitations of this system with respect to visualizing geo-spatial content. Given that this system was not specifically designed for geo-spatial visualization it is understandable that the framework does not fit geo-spatial data sets very well. Given the interaction technique to drag the matrix elsewhere to see the relationship clearly, this approach may cause some information loss on the geo-spatial features. Interactions for geo-spatial applications should retain nodes near to the original place to preserve as much location information as possible. We design an interaction approach that can add and remove nodes automatically while the user is moving the layout, to allow sub-graphs to better retain location information.

Second, the adjacency matrix has its own limitations for showing connected relationships. It may not provide the user with a rapid understanding of how data are connected. Although the matrix is able to show connections in a very detailed way, sometimes the user may simply want to see the trend at a glance. Moreover, the adjacency matrix layout is not flexible and it can only visualize connection data. We propose a more generalized layout that does not limit the visualization to a single technique within it, and we can show multiple sub-layouts together. For example, we can embed more generalized edge bundling techniques inside the circular layout and we can show both the trend and the detail.

Third, the NodeTrix approach may have problems dealing with large data sets, because of the interactive part of the technique. Sometimes there may not be enough space to perform the dragging operation stated in NodeTrix. In large-scale data sets, items can be highly overlapped and special measures may need to be introduced to restore clarity. So in

our model we attempt to utilize the space to the greatest extent by changing the layout of both inner and outer areas, and incorporate edge bending algorithms to help improve the layout optimization overall.

Another common approach of node layout optimization is a hierarchical unfolding of nodes. This approach hides unimportant nodes in the global view, but the user can unfold the nodes whenever with user interactions. This can be regarded as a compromise technique between filtering and layout optimization because it hides items and changes the layout at the same time. The Ask-graphview [42] system was put forward by Abello et al. in 2006 which is a successful framework that applied and improved the general node unfolding approach. They used recursive clustering to build node hierarchies, and improved it with balanced hierarchy reducing technique and user interaction.

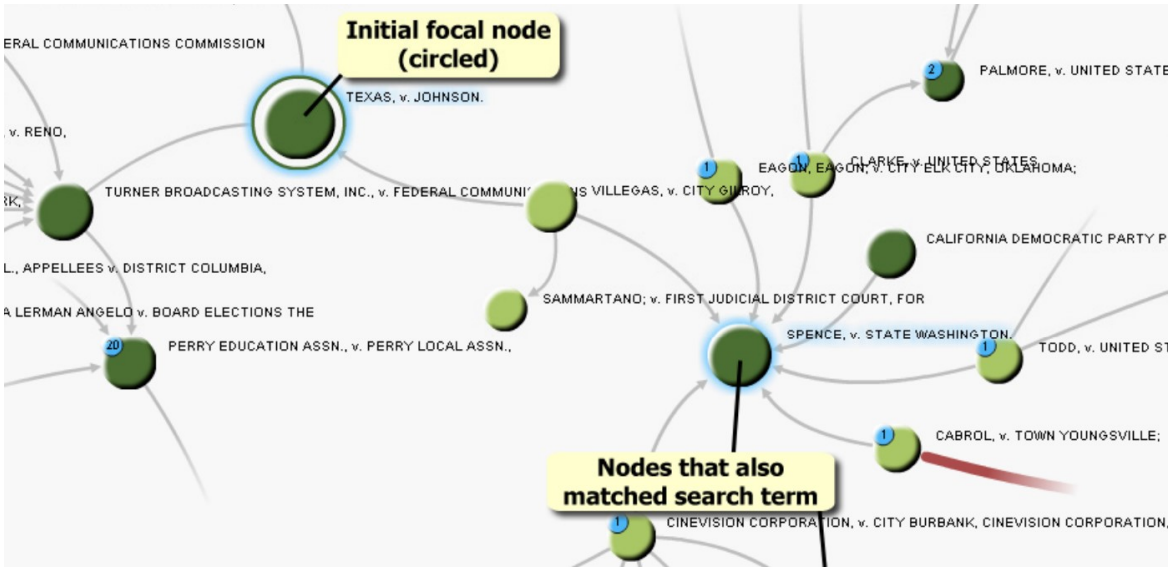



Figure 2.3: Subgraph layout centered on the focus node.

A remarkable approach proposed by Frank et al. in 2009 is an interaction model that supports degree-of-interest operations [62]. They adapt Furnas' original degree of interest function [24] from trees to graphs to extract useful contextual subgraphs, control the complexity of the generated visualization and direct users to interesting data points in the context [62]. The limitation stated in their paper that it is unlikely to find an optimal function to fit to various domains and tasks since they used property metrics to define the interest functions, so there may exist some extensibility issues. Figure 2.3 shows the subgraph centered on the focus node from their paper.

Another interesting paper focused on node organization is Dynamic Graph Animation by Yee et al. [54]. They define a focus node and design animations for interactive exploration in a radial tree. They enforced the ordering and orientation of the tree structure, to achieve a better visual presentation. The author focuses on controlling the animation process by selecting a focus node to transit the graph to a new layout, this may solve the clutter issue to some extent by interaction, however the approach does not consider geographic features of the nodes and it may require a proper focus node selection by the user. Figure 2.4 shows the process for changing the tree layout [54].

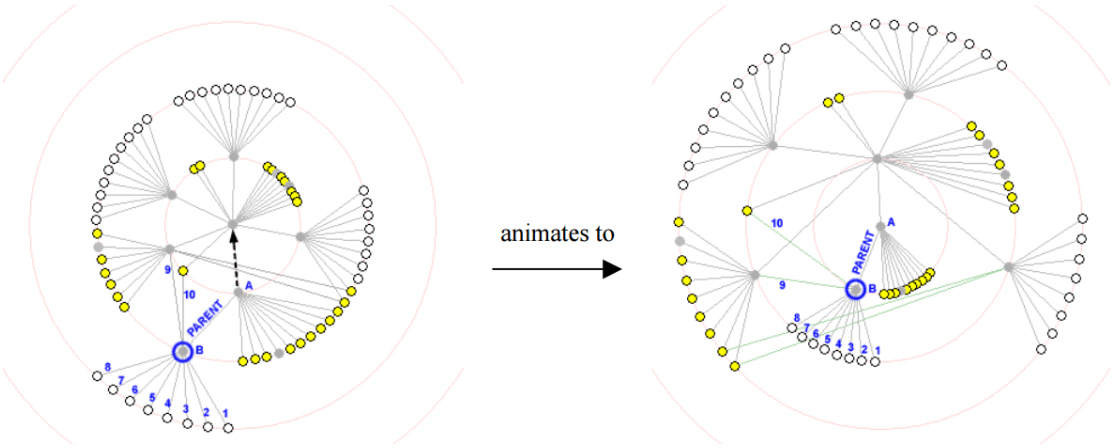



Figure 2.4: Animation process for selecting a new focus.

The shortcomings of hierarchy and degree of interest approaches are similar to filtering approaches. We may suffer information loss if not enough interactions are given, and it is hard to control the interaction when the data set is huge. Therefore more intelligent and automatic algorithms for expanding and folding are being considered by researchers. Besides, this approach may require tree structure or hierarchy level of the data sets themselves, which is a hard restriction for visualizing most network data sets.

### 2.1.3 Bundling Techniques

Edge bundling approaches are special cases of layout optimizations, however, the unique features of these approaches is the reason why we would discuss them in a separate section. Hierarchical edge bundles was initially proposed by Holten [26] and was a milestone for

visual clutter reduction. The approach bends edges as B-spline curves, with the hierarchy structure shown by a standard tree method [26]. The algorithm controls the extent of bundling via a parameter “bundling strength”. The larger the parameter is, the higher level of relations are presented. Moreover, this approach can be used to improve existing tree structure visualization frameworks as well. Figure 2.5 shows a software system and its associated call graph from the original paper [26]. The left picture is a balloon layout and the right picture is a radial layout both with bundling strength 0.85, respectively.

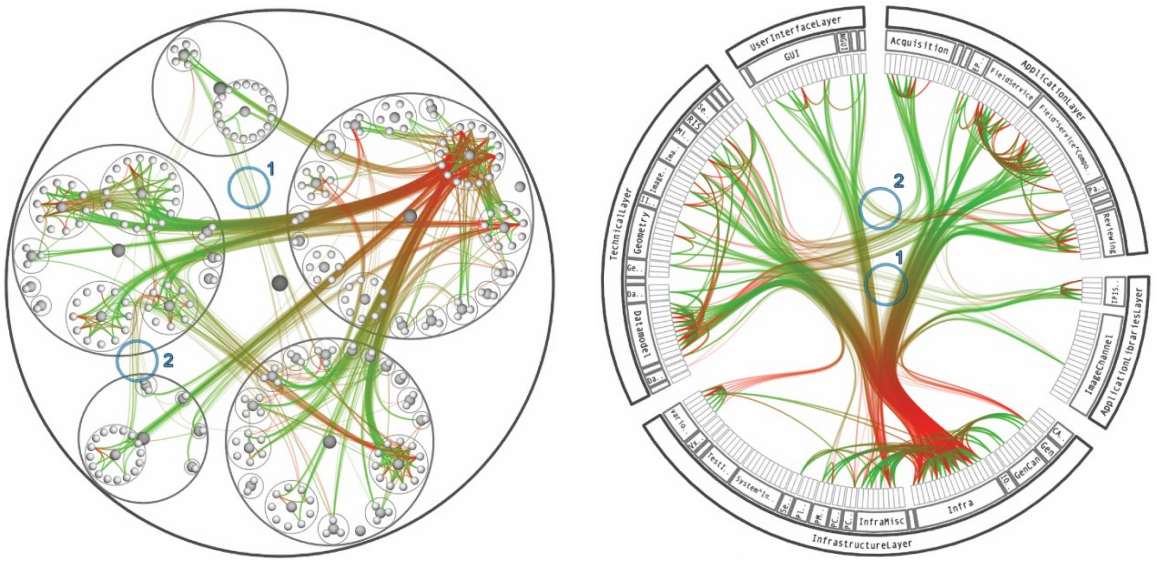
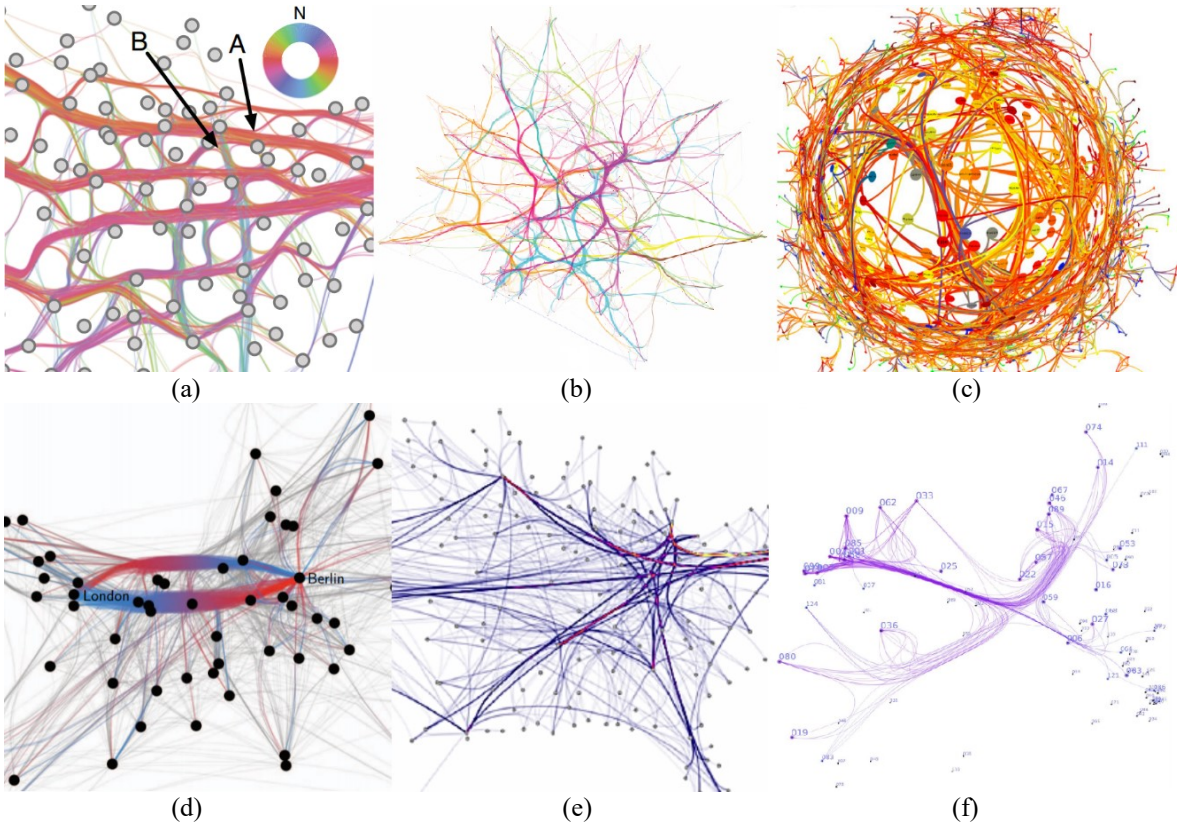



Figure 2.5: A software system and its associated call graph from the original paper.

After hierarchical edge bundling was proposed, a variety of other bundling techniques were put forward by researchers in the recent years [48, 34, 49, 50, 32, 51, 52]. Holten et al. proposed another bundling approach based on his original work, called “Force-directed edge bundling” [19], for node-link graphs. The approach no longer requires a hierarchy level of the data set itself, and performs a self-organizing bundling approach since the edges are modeled as flexible springs that can attract each other [19]. Ersoy et al. introduced “Skeleton-based edge bundling” that combines edge clustering, distance fields, and 2-dimensional skeleton to build bundled layout progressively [10]. The edges are attracted iteratively to the centerlines of the distance fields. The advantage of this approach is the speed and implementation simplicity, and the method allows explicit control of the emphasis on structure of the bundled layout [10].

"Divided edge bundling" was put forward by David et al. in 2011. They dealt with directional edges in their research by modifying forces in physical simulations [33]. "Directional lanes" are modeled and shown to illustrate the edge direction, and they introduced the "bundle weights" concept to enable more accurate visualizations. Hong et al. studied "Energy-based edge bundles" [46] by considering the topology of node-link graphs and optimized the cluster of edges by energy function. They claimed that their method provides good high level abstraction of the graphs. Antoine et al. proposed a system that exploits the full capabilities of GPU's to enhance the usability of edge bundling in real applications [47]. They also provided interactions such as such as zoom & pan, fish-eyes and magnifying lenses.




(a) (b) (c)

(d) (e) (f)

Figure 2.6: Edge bundling techniques. (a): Geometry edge bundling; (b): Skeleton edge bundling; (c): Living flows GPU intensive edge bundling; (d): Divided edge bundling; (e): FDEB; (f): Stream edge bundling.

"Geometry based bundles" [25] uses a control mesh to guide the edge-clustering, and edge bundles can be formed by forcing all edges to pass through some control points on the mesh [25]. The control mesh can be created by different levels of detail, and users can

interact with the bundles through color and opacity enhancement. Another noteworthy approach is “StreamEB” proposed by Quan et al [53]. They proposed temporal compatibility and neighborhood compatibility matrices to define relations between edges. Figure 2.6 shows the collection of bundling techniques mentioned above.

We think the main limitation of existing bundling approaches is that they focus only on the global edge congestion, but do not address node and label clutter problems. In fact, edge bundling presents something of a trade-off. On one hand, it reduces the clutter at the global scale and extracts the trend. But on the other hand, it may cause serious edge ambiguity in local areas. We also take advantage of the bundled approach in our approach to visualize local connections in the circular layout. Our model can be regarded as a progression of the basic model that addresses drawback of losing local structural information when using HEB algorithms.

### 2.1.4 Interaction

Interaction has become a prominent component of many visualization systems. There are a variety of methods for interaction, and they each have their strengths. We will review two typical important approaches in this section.

Edgelens is an interactive technique for reducing edge congestion [22]. The user places a focus of attention and the focus point curve edges around the area interactively. This approach can reveal node and label information while retaining the original node layout. They designed two algorithms to bend the edges, namely: “Spline EdgeLense” and “Bubble EdgeLense” and they conducted usability tests for them separately, then reached the conclusion that “Spline EdgeLense” is preferable to “Bubble EdgeLense”. They then studied transparency and color encoding to strengthen the contrast and developed algorithms for two lenses, achieving good results. Figure 2.7 illustrates the single lens result obtained from the original paper [22]. From this we can see that labels around Toronto can be seen clearly after transparency encoding.

However, limitations exist with regards to scalability. Edgelens focuses only on edge congestion but not on nodes layout. Generally, but in many networks nodes are often overlapped by many edges in large-scale network graphs. We can see from the figure above

that only Toronto and Kingston are shown clearly after transparency encoding, but other nodes are still cluttered in that particular focal configuration. So, there may be an underexplored middle ground that incorporates edge distortion, but also adopts a limited amount of node layout optimization. Therefore, we introduce sub-layouts to achieve a utilization of space that balances the competing importance of local network structure, flow information and the original node placement.

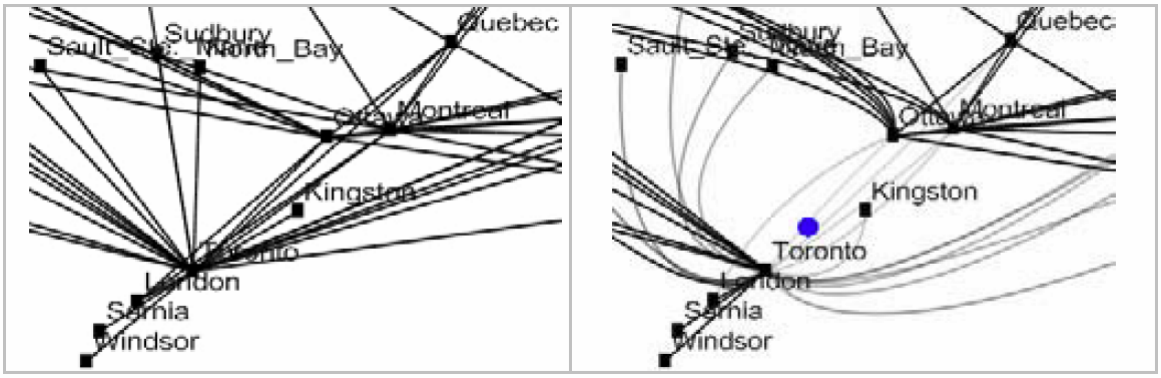



Figure 2.7: EdgeLens plus transparency reveals labels. Left: original graph; Right: result after applying Edgelens transparency encoding.

Michael et al. also proposed a method for interactively manipulating subgraphs in networks [55]. Figure 2.8 demonstrates the sample layout produced by their interface [55]. They focused on the user experience, and developed 3 different selection methods. They also defined commands for translation, rotation, scaling of the nodes via user interaction.

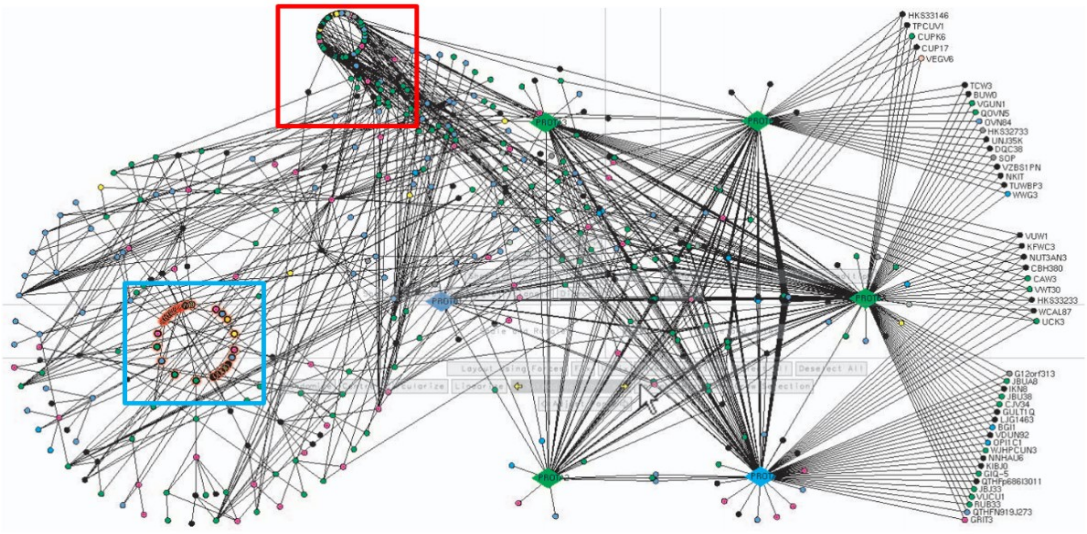


Figure 2.8: Sample layout produced by subgraph manipulation.

The limitation of this subgraph manipulation approach is that the clutter problem still exists after sub layouts are generated. As illustrated by the blue rectangle Figure 2.8, the nodes on the sub layout is overlapped. Moreover, as shown in a red rectangle in Figure 2.8, the edges and the nodes obscure each other in a narrow area. It is claimed in the paper that the layout can be moved to another location, for example, the blank area around the main graph. This movement may solve the clutter problem to a limited extent. However, this is not suitable for geo-spatial visualizations: first, we cannot always find a blank area in a graph that is suitable for the positioning, so the approach may rely on the layout of the graph; second, it is not reasonable to move the nodes to a completely unrelated position in a geo-spatial system.

We propose several novel improvements in contrast to this work and state them as follows. First, we use algorithms to avoid the overlapping of the nodes on the sub layouts by computing a better positioning of the nodes. Second, we use the inner area of the sub layouts to visualize the connection patterns between the nodes, to achieve a higher utilization of the space. Third, we perform edge bending algorithms on edges that are not related to the sub layouts to avoid clutter between nodes and edges, instead of leaving edges directly going through the layouts. Fourth, we propose a novel process to dynamically add and remove nodes from layouts, for preserving the geo-spatial feature of the nodes to the most extent. Finally, we design an array of features to facilitate the exploring of multiple layouts and time series data flows, we will describe these features in detail in Chapter 3.

### 2.1.5 Magnification & Distortion

Magnification and distortion are traditional techniques for browsing cluttered data. They enlarge some areas of a graph by linear or non-linear transforms, so that greater details can be seen after the operation [56, 57, 59, 60].

Furnas proposed the widely known fish eye view in 1986 [24] which defines a degree of interest function that assigns a value to each point in the area. The function defines how interested the user is in seeing that point. Figure 2.9 is a screen shot from the Data Lenses project [58]. Data Lenses is part of an interactive visualization tool that monitors the

activity at bus stops in Singapore, and it uses a fish eye approach to deal with the large volume of geographic data.

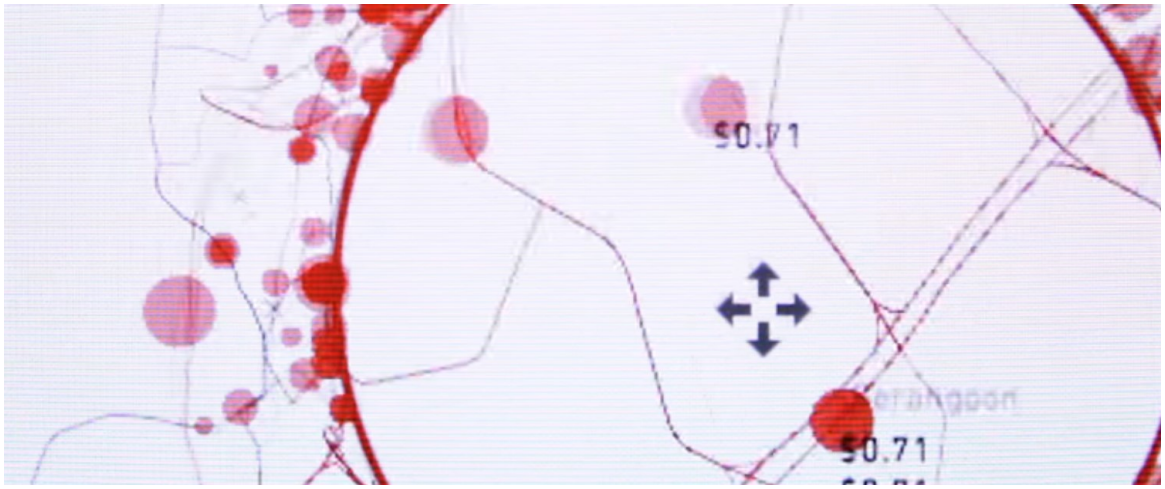



Figure 2.9: Fish eye operation in “Data lenses” system.

There are a few well known limitations of magnification and distortion approaches. First, the approach is not an absolute solution but rather a tradeoff between visual clarity and visual distortion. This may mean losing the original shape of the items in the area of interest to observe certain aspects more clearly. Second, magnification and distortion techniques have an effective upper limit for the number of items, meaning they cannot solve the problem when applied on huge data sets. Specifically, when there are too many relations in a certain area, the edges are still cluttered even after the fisheye operation.

## 2.2 Geo-spatial Network Visualization Frameworks

The last decade witnessed a steep rise in commercial demands for and a revolution of geographic information systems, which led to a series of contemporary geographic visualization technologies that have emerged [6, 7, 31, 32, 33, 34, 61].

Nagel et al. proposed an interactive system for exploring geographic content on a touch screen surface [6]. They dealt with publication co-authorship data and the users were able to explore the relations between their affiliations and other institutions. An interesting 3-dimensional edge bundling technique was described in Lambert et al.’s paper [34]. They extended the traditional 2-dimensional edges bundling method and visualized geo-spatial

information in the context of the globe. They also studied a GPU-based rendering method to emphasize bundle densities while preserving edge color [34].

Figure 2.10 demonstrates sample results in the paper from Tijmen et al [61]. In their work they present spatiotemporal patterns in a visually continuous way. They extended and adapted several image-based visualization techniques, including animation, density maps, and bundled graphs [61]. They also created real-time animations by using GPU-accelerated techniques. They further help reduce clutters using several transfer functions, and then apply bundling technique after the congestion detection process.

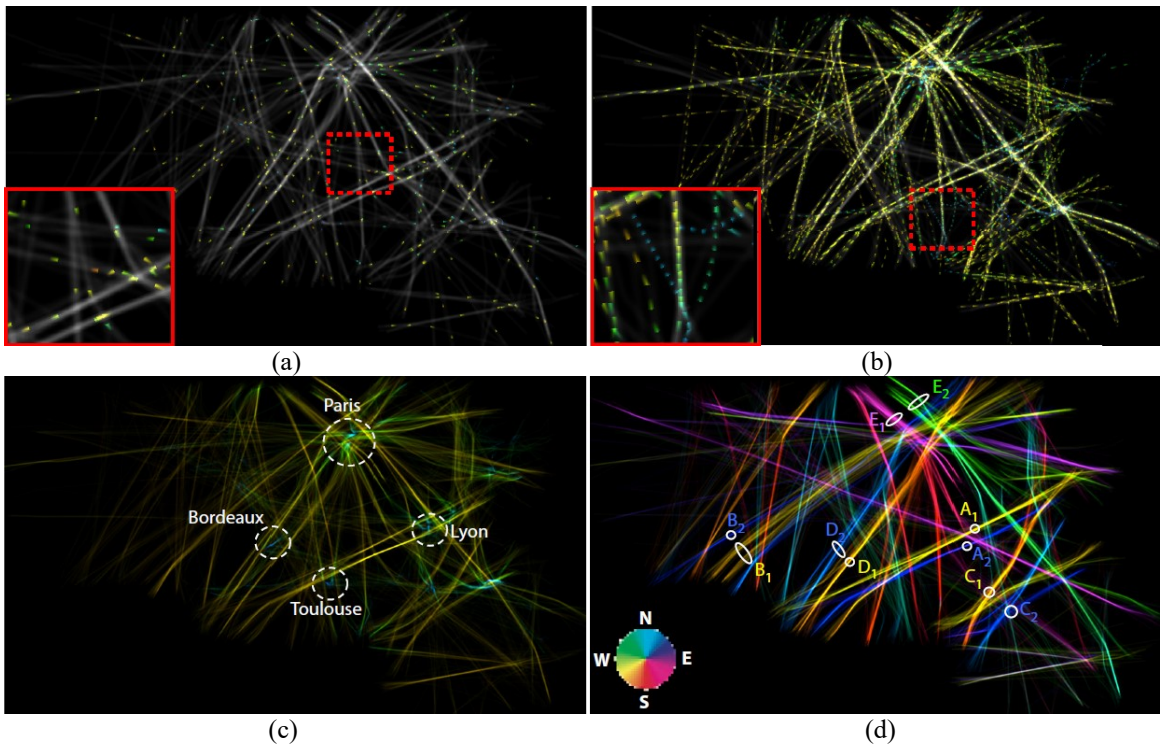
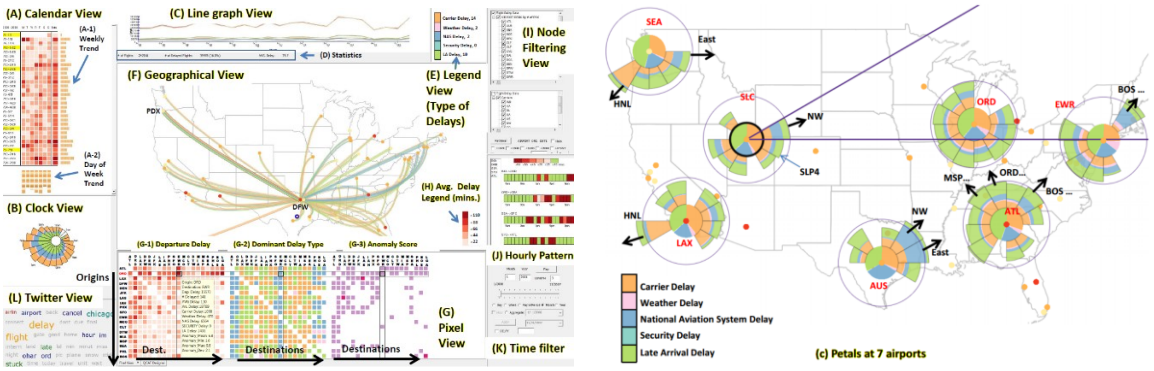



Figure 2.10: Animated multivariate visualization [61] based on French airline data. (a): Instantaneous plane positions, with color-coded height; (b): Trail segments over short time periods, with color-coded height. Trails over entire studied one-week period with color coding height (c) and direction (d).

Sungahnn et al. put forward a framework focusing on the integration of a set of visual analytic techniques for analyzing high-dimensional, multivariate network data that involves spatial and temporal information [31]. They also presented new visual elements, called “Petal and Thread”, to effectively present many-to-many network data including multi-attribute vectors [31]. In addition, they developed a model for anomaly detection, and

displayed highlighted anomalies in a visually consistent manner. Figure 2.11 (a) and Figure 2.11 (b) illustrates the main appearance of their model.




(a) (b)

Figure 2.11: Multivariate network data visualization model [31]. (a): The interface that consists of a set of different views. The main view in the middle is the "geographic view" with "Thread" representations. (b): Visual delay analysis results. "Petals" are located on the map to show the highest delays as well as reduce the visual clutter.

This research provided some inspiration for the current work. First, the concept of integrating multiple views in one system are suitable for modern geo-visualization platforms due to the scale and complexity of the data. As the space is often limited in the map or a terrain to show multi-perspective views, we also introduce additional views in our model to assist the browsing and mining of the detailed information. Second, the "Thread and petal" idea to show multivariate aggregated data only deals with one origin – one destination pair or one origin - multiple destination pairs. We propose a different technique to visualize aggregated information between two different groups both formed by multiple nodes. We will discuss the approach in Chapter 3.

## 2.3 Time Series Visualizing Techniques

Time series data visualization is a major area of information visualization and various technologies have been proposed over many years [67, 68, 69, 70]. However, in this section we again restrict our focus to the following research that are closely related to our work.

"ThemeRiver" was proposed by Havre et al. in 2000 [63, 64], and it was initially proposed for visualizing a large collection of documents. The river shown within the

context of a timeline flows from left to right through time, and the changing width of the river depicts changes in strength of an individual topic or a group of topics [63]. The "currents" in the river stands for themes or topics, and they are colored to describe different item categories. Byron et al. presented the term "stacked graphs" in 2008 and summarized the design decisions and algorithms behind stacked graphs from the perspectives of both geometry and aesthetic [65]. They then discussed techniques for coloring and ordering the layers of such graphs.

In the recent years, stacked graphs are enjoying an increasingly prominent role in subsequent visualization systems. Topic analysis is an emerging field in visualization and Cui et al. proposed "TextFlow" to help us better understand how topics evolve in text data [66]. They firstly extracted three-level features from the data, then designed three new visual components to convey complex relationships between the features. Finally, they allow users to refine the analysis result by interaction. Figure 2.12 shows selected topic flows of "VisWeek" publication data with thread weaving patterns from Cui's paper [66]. Their work is a valuable reference for our own peapod model to be discussed in Chapter 3.

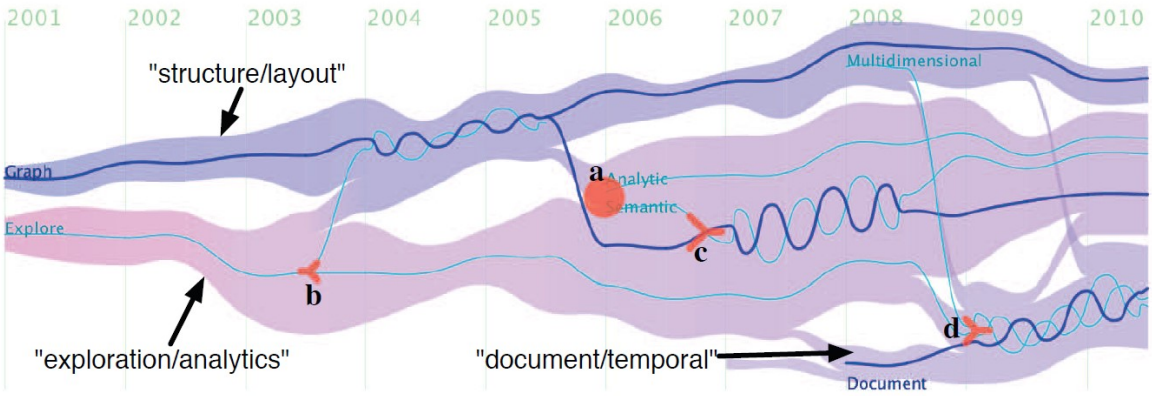



Figure 2.12: Selected topic flows of VisWeek publication data with thread weaving patterns.

Another framework that is worth noting is the "TorusVis$^{ND}$" shown in Figure 2.13 [71]. They use 3 cooperating visualizations to monitor a network and its communication patterns: network display, node selector and time slicer. The interesting point is that they combined radial layout bundling with the theme river technique, and achieved good results. However, they just used "Theme River" without modification as a component interaction technique.

To compare, we also propose a more modified model in Chapter 3 that also combines radial layouts with Theme Rivers and called the “Peapod Model”.

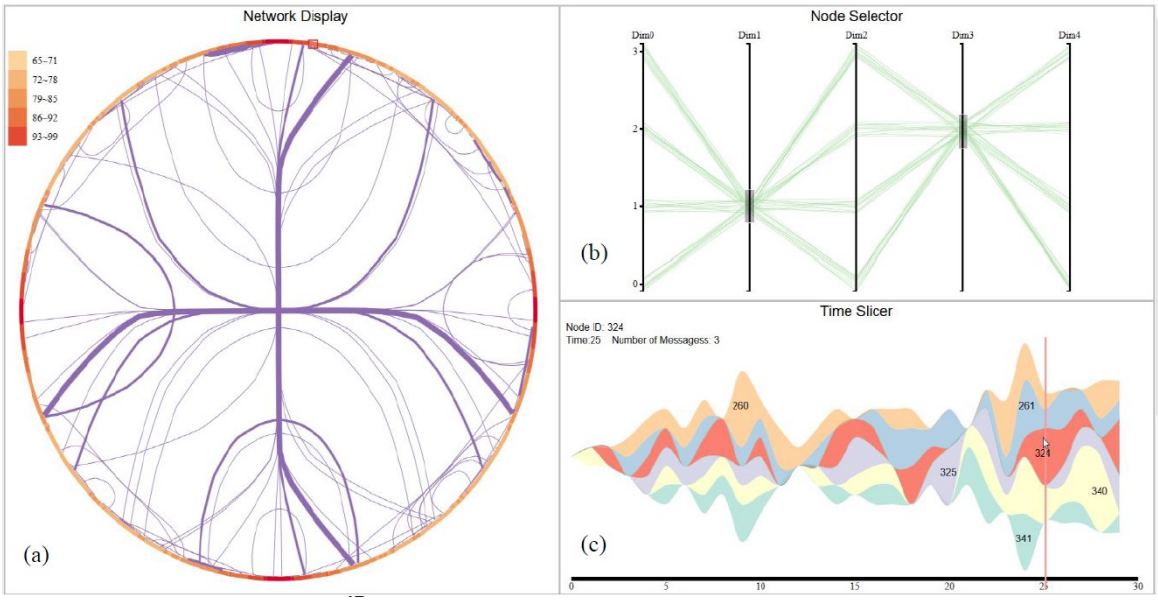



Figure 2.13: Visualization framework TorusVis$^{ND}$ and its three components.

## 2.4 Limitations of Previous Approaches

We have already given descriptions of the limitations for all related works in previous sections; however, for further clarity now offer a brief summary and review the aggregate limitations and the opportunities that remain.

The current visual clutter reduction techniques can be divided to 5 categories: content filtering, layout optimization, bundling techniques, user interaction and magnification and distortion approaches.

Filtering can be used as a complementary approach with other techniques, but the drawback is that the formulation of the filtering threshold is difficult to define, and the information loss problem brought by filtering would cause further misleading for relations between elements.

Layout optimization approaches include node layouts and edge layout. Node layout approaches have problems dealing with geo-spatial contents due to the loss of location information, and suffer interaction difficulty when data grows significantly. Edge layout

approaches, such as bundling techniques, are effectively performing tradeoffs between global congestion and local clutter. They may still leave serious clutter problems in local node and label visualizations.

Interaction usually are combined with other strategies in visualization systems. The limitations of interaction techniques are very specific to individual problems and models. However, all interactive systems face the problem that data volumes are witnessing high growth rates. Therefore, dealing with large-scale data sets should be a key consideration for researchers going forward.

Magnification and distortion enlarge some areas by linear or non-linear transforms. However, shape information are sometimes compromised or even eliminated by the geometric operations. Besides, magnification and distortion techniques exhibit an upper limit of effectiveness for the number of items in the area of interest. They also do not completely solve the clutter problem.

We further introduced several interesting systems for visualizing geographic networks, and they all have their points of innovation worthy of note. However, regarding existing geo-spatial node-link visualization frameworks, some improvements may be made in the following aspects. First, clutter issues in geo-spatial systems should be given more attention, since the data available is becoming increasingly large and the geo-spatial node-link graphs encounter clutter issues frequently. Specifically, reducing visual clutter while maintaining the geo-spatial features as much as possible may be the most challenging topic. Second, various traditional visualization techniques might be improved or combined to better meet the needs of geo-visualization systems. For instance, traditional circular layouts may be further improved by designing a more appropriate node positioning scheme for geo-spatial data. We will introduce several such techniques in Chapter 3.

# Chapter 3

# Approach: The Eye Opening Visualization Model

This chapter describes our approach for visualizing data in connected graphs. First, we provide an overview of the framework and interface design in Section 3.1. Then, we elaborate on several main features proposed in Section 3.2, 3.3 and 3.4, separately. Finally, we introduce a set of complementary features designed to further support the process of data exploration in Section 3.5.

## 3.1 Visualization and System Overview

The Eye Opening visualization framework is an approach for browsing both global and local relations in geographic connected graphs. We follow the standards for geo-visualization network systems we defined in Chapter 1 and refer to the merits and limitations of previous approaches we discussed in Chapter 2 when designing our model to build a more informed design.

From the perspective of system design, we focus on providing dynamic techniques for revealing local information while reducing the local visual clutter, and we develop and incorporate different technologies such as radial layouts, curve bundling techniques, word clouds, and stream graphs in an integrated system to provide multiple views to depict the data sets.

From the perspective of development and implementation, our model is developed by the web-based D3 visualization engine [28] and front-end languages such as JavaScript and HTML. We also incorporate the advantage of modern web development libraries such as JQuery library [72] and Bootstrap CSS library [73] in our system to build a portable and dynamic interface. The system uses object-oriented architecture to model abstract instances to ensure the extensibility and maintainability.

We begin with an overview of the system interface to situate the reader in our discussion. The interface consists of several connected views, namely: the main view (A), the hub

stream view, (B), the ensemble hub view (C), the cloud view (D), the detail view (E), and the control panel (F). Figure 3.1 illustrates the layout of the overall system interface, with corresponding labels identifying different views and we will now provide a brief introduction about the function of each view.

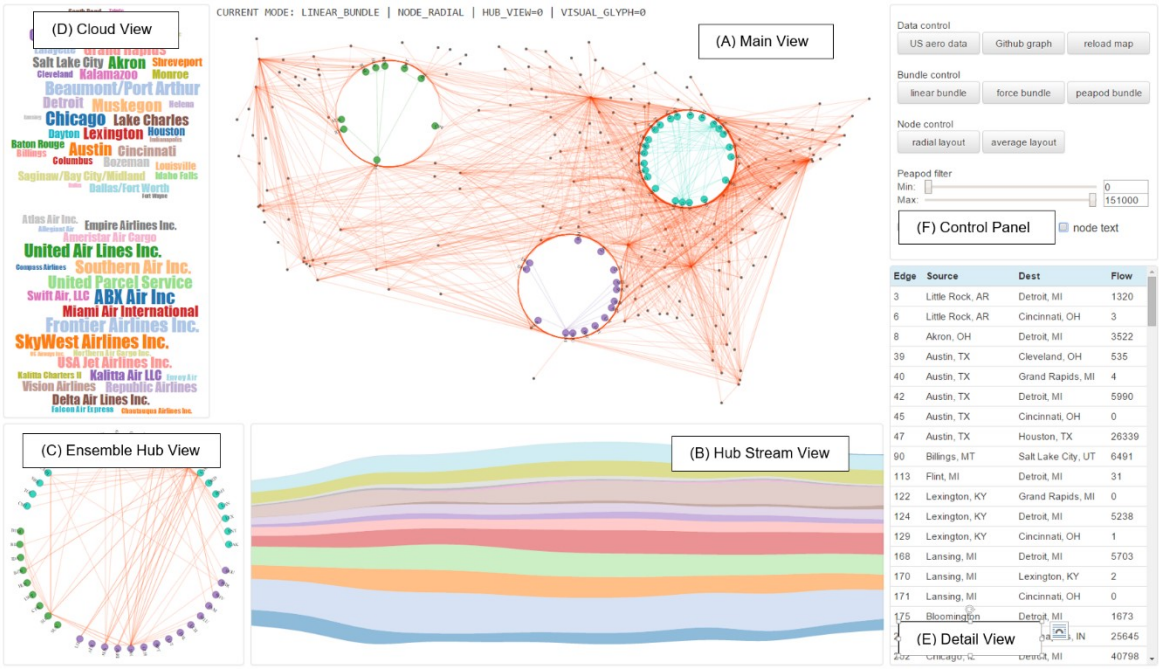



Figure 3.1: System interface design.

*(A) Main view: main display area to illustrate data*

The main view located in the upper middle area of the interface, shows the connected network graph related to the current dataset. One or more areas of interest can be selected in the main view to create sub-layouts and subsequently perform further analysis with the nodes and circular layouts. Two modes are available in the main view: connection mode and hub mode. The purpose of the connection mode is mainly to facilitate the analysis of the connection data within a set of nodes of interest. In the connection mode, we can switch between 3 different edge layouts and 2 different node layouts. The 3 edge layouts are: linear bundle, force-directed bundle and peapod bundle, and the 2 node layouts are: radial layout and uniform layout. Switching among these modes is controlled with simple radio buttons in the control panel. As a concise reference we list and briefly describe all modes introduced above as follows:

***Connection mode**:* visualizing connection information between nodes of interest and temporarily reducing the visual clutter in a local area while maintaining the global edge trends with dynamic circular layouts.

Three edge layouts available under connection mode include:

- *Linear bundle*: visualizing direct links with linear connections. This is the default bundle mode for edge layout.
- *Force-directed bundle*: visualizing direct links with non-linear curve bundling techniques.
- *Peapod bundle*: visualizing time series data flows among selected nodes.

Two node layouts under connection mode:

- *Radial layout*: positioning the nodes by direct radial mapping to the circumference. This is the default mode for node layout.
- *Uniform layout*: positioning the nodes evenly on the circumference.

***Hub mode**:* visualizing time series data flows among different groups of nodes.

As the design of arranging multiple circular layouts in a node-link graph is similar to the layout of the hubs in the distributed computer network, we use a "hub" metaphor, and we define each circular layout as an individual "hub". We will describe the visualization techniques listed above in detail in the following sections.

*(B) Hub stream view: a complementary view to the peapod bundle*

Hub stream views can provide additional aggregated information to individual peapod bundle (described in more detail in section 3.3.1). It is located in the middle bottom area of the interface. The hub stream view visualizes more detailed information about time series data that cannot be shown in the circular area due to the high volume of connections. It employs a stack graph technique to aggregate peapods in the circular layouts together to show more specific information. Taking airline traffic as an example, the hub stream view might show categorized information that represents the business share of different airline

companies among a selection of nodes. Rather than focusing at the individual node level, it is able to focus at the categorical level.

*(C) Ensemble hub view: a supplemental feature to the hub modes*

The ensemble hub view is located in the left bottom of the interface. It incorporates all the connection data among all hub selections (rather than a single hub) and visualizes them as an integrated circular layout. It illustrates the relations between nodes in different hubs that are located far from each other. The ensemble hub view can serve as a complementary view for the hub mode in the main view. It also makes use of consistent color encodings to facilitate comparison across different views.

*(D) Cloud view: showing text clouds about the current selection*

This view is located to the left of the main view, and it presents tag clouds corresponding to the contents in the main view. There are two different clouds representing different levels of aggregation. The upper cloud illustrates information about the nodes, while the lower cloud describes the categorical information for the current selection. The cloud view offers information at a glance about what items are included in the current selection.

*(E) Detail view: listing related data entries for a specific selection*

The detailed view is in the right bottom area of the interface, and it provides a space to show the detailed information of all the edges that are in the current selection. For example, the source and destination city names, and exact values of distances or passengers from several specific airline routes in a transportation network.

*(F) Control panel*

The control panel is located in the upper right corner of the interface, and it provides a set of options which change various aspects of the visualization. For example, there is a

filter option in the control panel to filter out peapod connections outside of a specific range according to the average data flows.

The hub stream view, ensemble hub view, cloud view and detail view update themselves automatically when operation is performed in the main view.

## 3.2 Dynamic Layout Design

This section introduces novel techniques for exploring geographic and network node-link graphs. The motivation for this approach stems from ubiquitous clutter problems that exist in current node-link graph visualization systems as stated in Chapter 1. The dynamic technique can help reduce the visual clutter on demand at the local scale while revealing underlying connection patterns among selections of data entries. We also elaborate the approaches and algorithms for designing node and edge positions in the dynamic circular layouts generated.

### 3.2.1 A Dynamic Design for on Demand Local Clarity

We propose a novel approach to assist in the analysis of local connection information in a highly overlapped context in this section. The main technique of our model involves two types of operations: the initial selection and the dynamic placement of hubs. We discuss the two techniques separately in the following sub-sections.

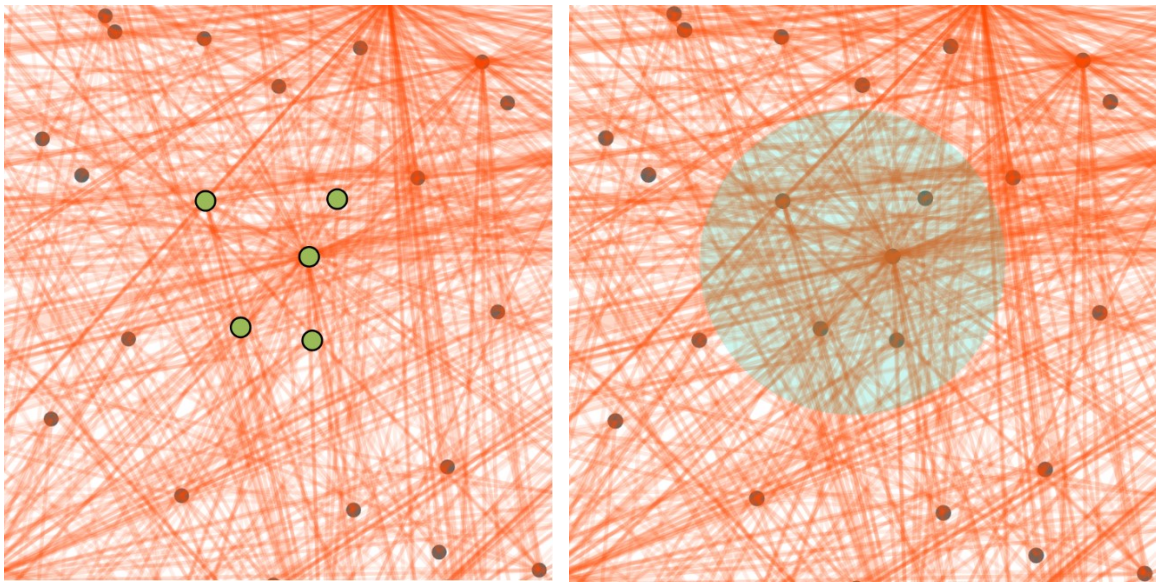
*Initial Selection*

The initial selection process involves defining the area of interest, the selection operation with a circular window, and the generation of the circular sub layouts, called hubs. This process can reveal essential hidden information due to the overlapped or cluttered edges and nodes inside the area of interest.

Our approach extracts all of the related nodes in the circular area of interest through geometry calculations, and then arranges them along a circular sub-layout according to different layout designs. At the same time, the connections between these nodes, which can

be named as “inner edges”, are stored in memory for computing and building inner circular layout in later steps. Moreover, we update any other edges that pass through the circular layouts all over the main graph, which are named as “outer edges”, through an edge bending algorithm. We will discuss the node positioning and the edge bending algorithms in Section 3.2.2 and Section 3.2.3, respectively.

The dynamic process is illustrated in Figure 3.2. First, we find an area of interest in the original graph and then click the mouse to choose a starting point, as illustrated in Figure 3.2 (a). Second, we drag the mouse to the expected ending point. We can adjust the ending point while moving the mouse to confirm the most appropriate area of interest. A circular window is generated and updated to assist the selection, as illustrated in Figure 3.2 (b). Third, the nodes and edges are animated automatically to fit the boundary of the area of interest. The selected area is always a circle in the sub layouts.

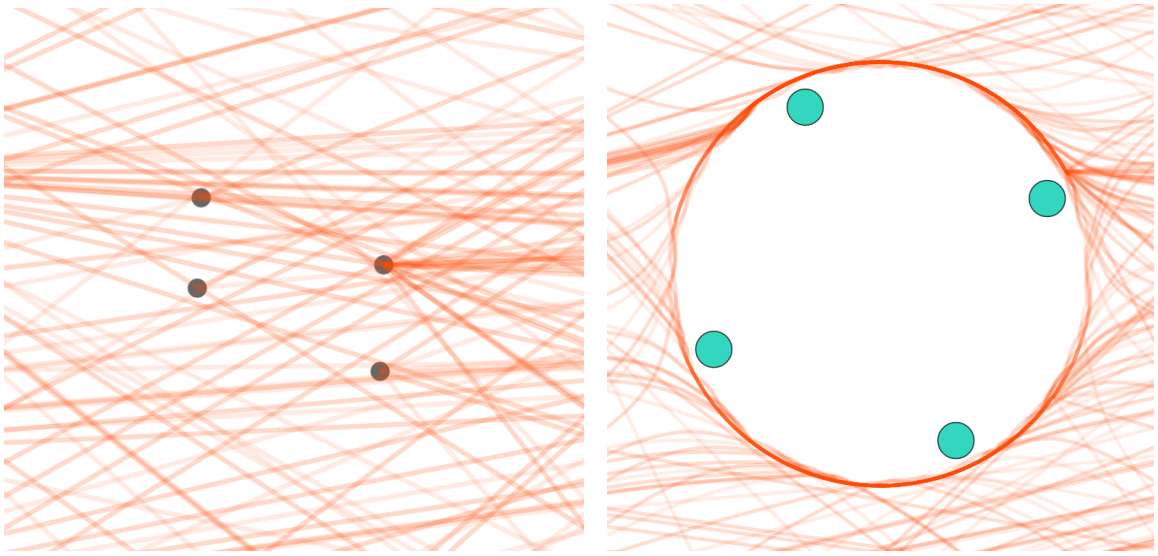


(a) (b)

Figure 3.2: Illustration of the initial selection process – phase 1. (a): The potential nodes of interest including 5 nodes are shown in green. (b): The intermediate state while we are adjusting the final result. A circular window is generated and updated during the process.

Figure 3.3 shows the comparison of the original layout and the layout after the initial selection for 4 local nodes. From the comparison in Figure 3.3 we can see that the 4 nodes in the initial image are arranged with color encodings as a circular layout and a blank area is generated in the center as a result. We note that connections between the nodes in the

original image Figure 3.3 (a) cannot be observed easily due to the overlapping and clutter issue. However, we can draw a conclusion that there are actually no connections among the 4 nodes from the result in Figure 3.3 (b). Therefore, the approach can relieve the clutter problem on demand while a set of nodes and their edges are under scrutiny.



(a) (b)

Figure 3.3: Illustration of the initial selection process – phase 2. (a): The original nodes and edges layout. We cannot see clearly if there are any relations between those nodes. (b): The circular layout after the selection operation. The selected nodes are highlighted by a color encoding and magnification animation. It demonstrates that no connections exist among the selected nodes.

Our approach allows the selection of multiple circular layouts simultaneously as well, and the circular layouts can be likened to the “hubs” in distributional computer networks in this case. Figure 3.4 illustrates multiple circular layouts existing in one large connected graph after multiple initial selections.

*Dynamic Placement of Hubs*

Another important issue for manipulating sub layouts is to allow the flexible control of the layouts themselves. The system allows dynamic placement of the hubs after the initial selections. The key benefit of providing the dynamic placement is to perform fine tuning on the location of the layout after the initial selection. Moreover, it is a dynamic way to depict the update of internal connections in a sub layout by adding and removing nodes.

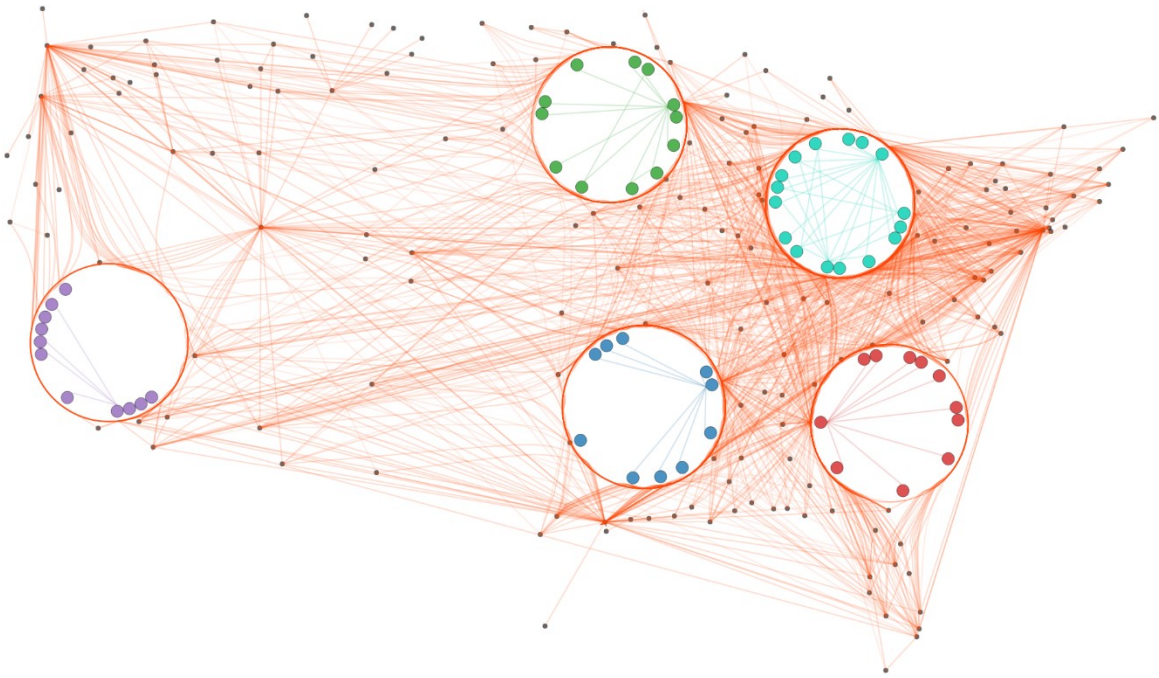


Figure 3.4: Multiple circular layouts in one connected graph. The data set is US Airline Data [74, 77].

To facilitate this process, we build a “circular window” for selecting and moving the area of interest. The circular window is defined by the actual selected circular area in the initial selection step. Once selected, the generated circular layout and all of the nodes on it will update themselves while the “circular window” is repositioned. When the movement operation is completed, the circular selection area disappears.

The system keeps an array of the selected nodes in memory and adds or deletes nodes dynamically from the current selection while moving. Specifically, when the “circular window” intersects with a new node, the system adds the new node to the array and updates the layout immediately. Moreover, during each movement process, the system recalculates the shapes of all related “outer edges” by spline interpolation and update them synchronously with animation. After the movements are finished, the final state is then computed by the system. Figure 3.5 demonstrates a typical dynamic placement process for a circular layout. The “circular window” moves along the diagonal line of the screen area from upper left to the right bottom. The dynamic layout design is enabled by a series of corresponding algorithms and features, and the complexity analysis of the related algorithms are stated in Chapter 4.

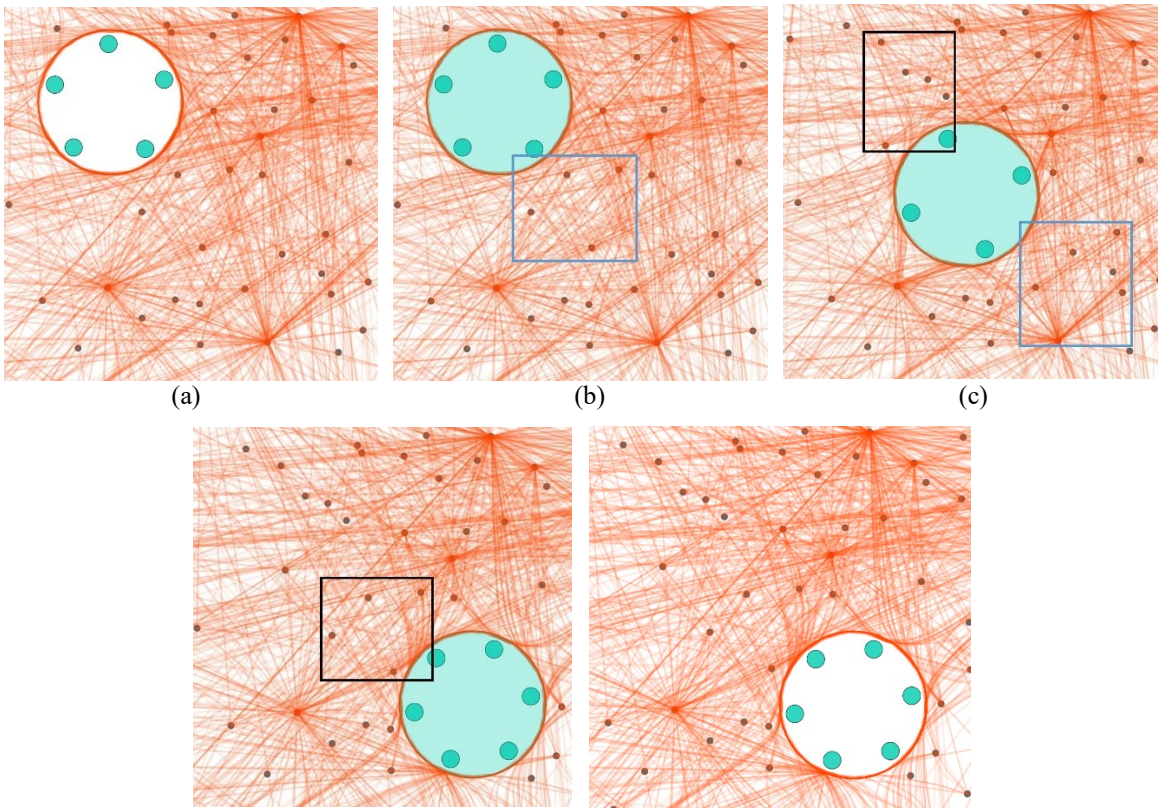


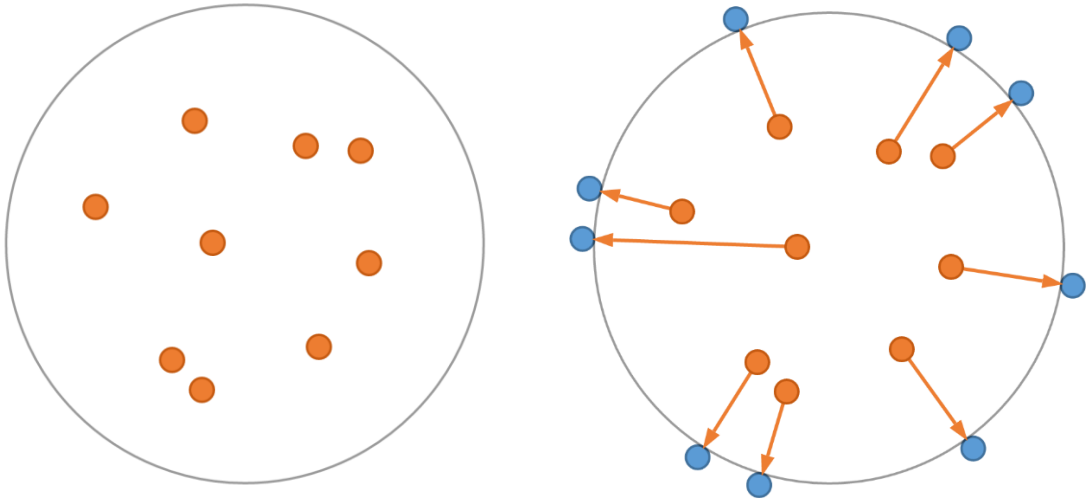

Figure 3.5: A typical hub repositioning. (a): Initial state after the selection operation. 5 data items are selected as a circular layout. (b): A circular window is generated to represent the area of interest. The items in the blue box are going to be added to the layout. (c): The circular window are dragged along the diagonal direction, and the layout is updated. The items in the black box are excluded from the layout, and the items in the blue box are to be added. (d): At this stage, items in the black box are excluded. (e): Final state of the movement, the circular window is hidden.

### 3.2.2 Node Layout Design

In this section we discuss the algorithms that determine node positioning in the initial selection and the hub dynamic placement process. We design two node layouts to adapt to different situations and requirements, namely: radial layout and uniform layout. These two layouts represent trade-offs between retaining some information about the original location of the nodes and overall clarity.

*Radial layout*

The radial layout is the default way to calculate new node positions and places more of an emphasis on the original node locations. It moves nodes along the radius of the circular layout by geometry computation. Figure 3.6 demonstrates how 9 nodes in the area of interest can be transformed to the circumference by a radial mapping.

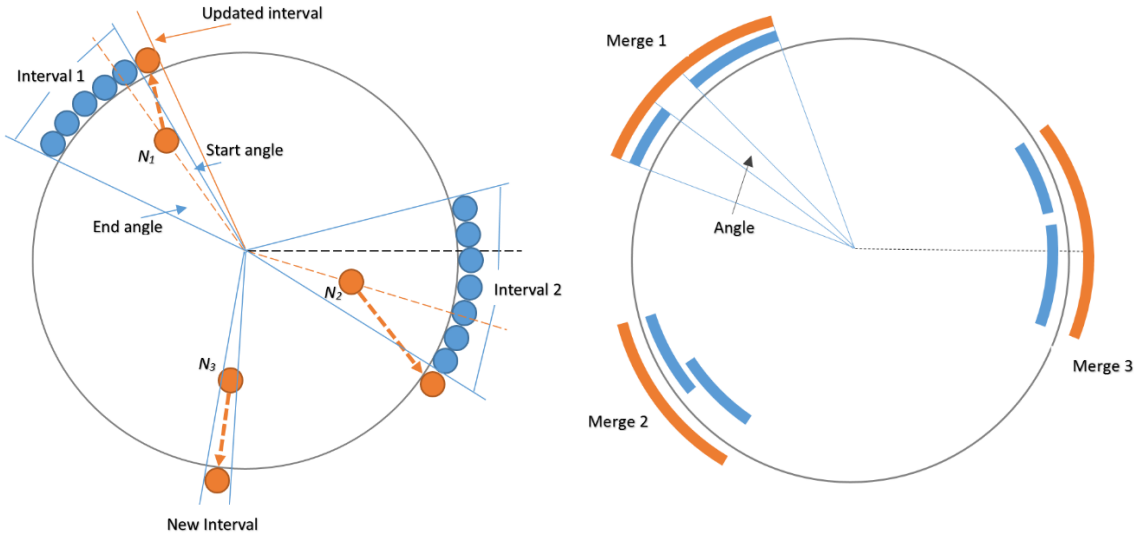


(a) (b)

Figure 3.6: Radial layout illustration. (a): The initial state: 9 nodes are selected and located in their initial places. (b): The nodes are transformed to the circumference by a radial mapping.

However, there is a drawback of this approach. Sometimes two nodes can be mapped to very close positions or even the same position can cause an overlap problem if they sit near the same radius in the circular layout. Although we can still distinguish between nodes in most cases even if they are overlapped, it is reasonable to design an approach to reduce the overlapping when this issue arises.

We propose an iterative node positioning algorithm to tackle the issue. We iterate all selected nodes and insert them into the circular layout one by one and we ensure that the current node can only be mapped to available positions that do not intersect with previous nodes. To achieve this, in each iteration step, we perform a two stage algorithm: *inserting* and *merging*. The two stages are illustrated in Figure 3.7.




(a) (b)

Figure 3.7: Two staged radial mapping optimization. (a): Inserting stage: nodes are inserted to the circumference according to relative positions with the intervals. Intervals are either created or updated after insertion. (b): Merging stage: intervals are merged if intersections exist to facilitate further insertions.

The inserting process is illustrated in Figure 3.7 (a), and we provided the pseudocode for the inserting process as Algorithm 3.1. Here we maintain a global array of intervals that stores different angle ranges to record all spaces that are already occupied on the circumference, we also track the angle along the x-axis for every node to be inserted. In the insertion stage, if the current node intersects with any interval, we then test on which side of the interval the node should be inserted. The node is inserted to the nearest side on the interval, a typical case can be illustrated by inserting node $N_1$ to into *Interval 1* in Figure 3.7 (a). When the current angle is close to the start angle, we then add an increment to the current angle, and insert the node according to the new angle.

When the intervals go across the positive x-axis, as illustrated by node $N_2$ and Interval 2 pair in Figure 3.7 (a), we have to add additional boundary tests and adjustments before performing the insertion. For example, if the intervals go across the range $[0, 2\pi]$, we then constraint them back to $[0, 2\pi]$ by a periodic function. However the principle is all the same with previously discussed. Once we insert a new node to an interval, we update the range of that interval. Or, when no intersection occurs, we directly map the node to the circumference by the current angle then add a new interval to the interval array, as

illustrated by node $N_3$ in Figure 3.7 (a). By performing the insertion process, we ensure that all nodes are arranged at least half a diameter away from each other.

**Algorithm 3.1: InsertNode (O, R, Location, Intervals)**
Insert a new node into the circumference according to the existing node layout.

**Input:** O is the hub center, R is the hub radius, location is the coordinate of the current node to be inserted, and Intervals is the interval array for the current circular layout.
**Output:** Intervals is the interval array for the current circular layout, and NewLocation is the new position for the current node.

**Start algorithm**

```
NewInterval = true;
for Interval in Intervals:
   if Location intersects with interval:
       Calculate NewLocation for the current node according to
       O, R, and the closer end of the intersecting interval;
       Update intervals due to the new node insertion
       operation;
       NewInterval = false;
       break;
   else:
       continue;
   endif
endfor

if NewInterval == true:
   Calculate NewLocation for the current node according to O,
   R, and Location;
   Create a new interval, and add it to Intervals;
endif
```

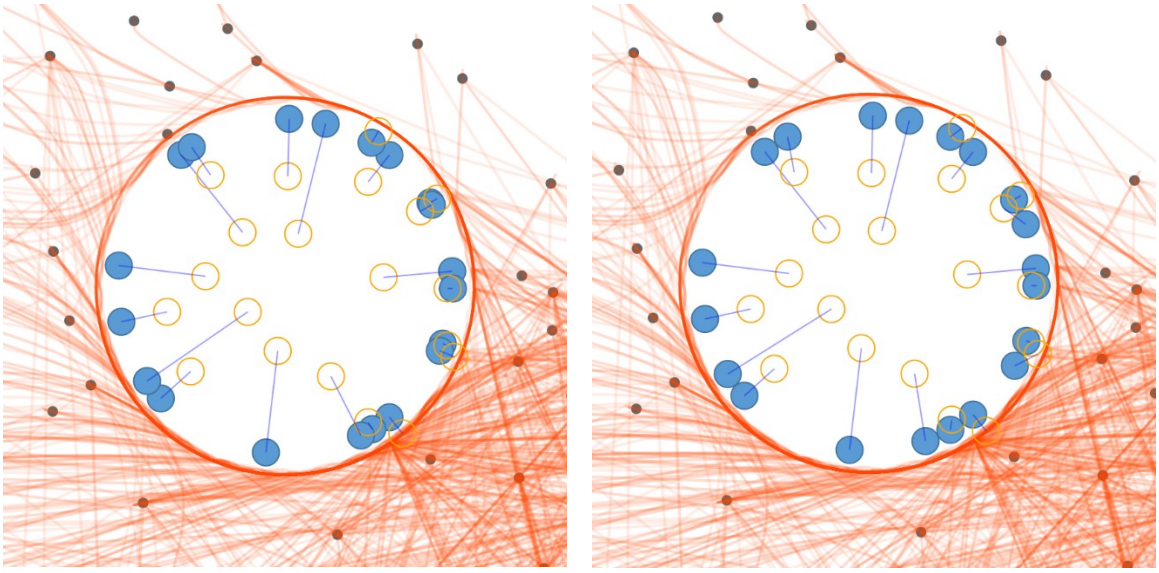
**End algorithm**

The merging process is described by Figure 3.7 (b). It merges and updates the interval list once we insert a new node. The reason why we have this step is that we have updated the interval list during the inserting process, so the intervals may be very close to each other after the insertion. Several typical cases under which the intervals should be merged are illustrated in Figure 3.7 (b). In Figure 3.7 (b) the blue arcs represent the existing intervals, and the orange arcs denote the intervals after merging operation. The pseudo code for merging the intervals is stated in Algorithm 3.2.

```
Algorithm 3.2: MergeIntervals (Intervals)
  Update and merge the intervals after a node insertion
  operation.

Input: Intervals is the interval array for the current hub.
Output: Intervals.

Start algorithm
   for IntervalA in Intervals:
      for IntervalB in Intervals:
          if IntervalA != IntervalB && IntervalA intersects with
          IntervalB:
              NewInterval = IntervalA ∩ IntervalB;
              Remove IntervalA and IntervalB from Intervals, and
               add NewInterval to Intervals;
           endif
      endfor
    endfor

End algorithm
```

We iteratively test if an intersection exists between each pair of intervals by their start and end angles. The intersection is defined as: the angle between the end angle of the first interval and the start of the second interval is smaller than the corresponding central angle that a node should occupy. Taking the *Merge 1* in Figure 3.7 (b) as an example, the orange merged interval sets its start angle as the longer blue arc's start angle and end angle as the shorter blue arc's end angle. Once the new interval is generated, we remove both old intervals from the list and insert the new interval to the list. Intervals may also overlap with each other, as illustrated in *Merge 2* in Figure 3.7 (b), the principle is the same. Similar to the node insertion process, when the intervals go across the positive x-axis, as illustrated by *Merge 3* in Figure 3.7 (b), we have to add some more boundary tests and adjustments before performing the merging operation.

Figure 3.8 shows the comparison for the previous direct mapping result and the mapping result after the two staged optimization, with visual glyphs to indicate the original positions of the corresponding nodes. From Figure 3.8 we can see that the two staged optimization reduces the overlap between individual nodes by an incremental angle while helping to preserve the original radial mapping structure.



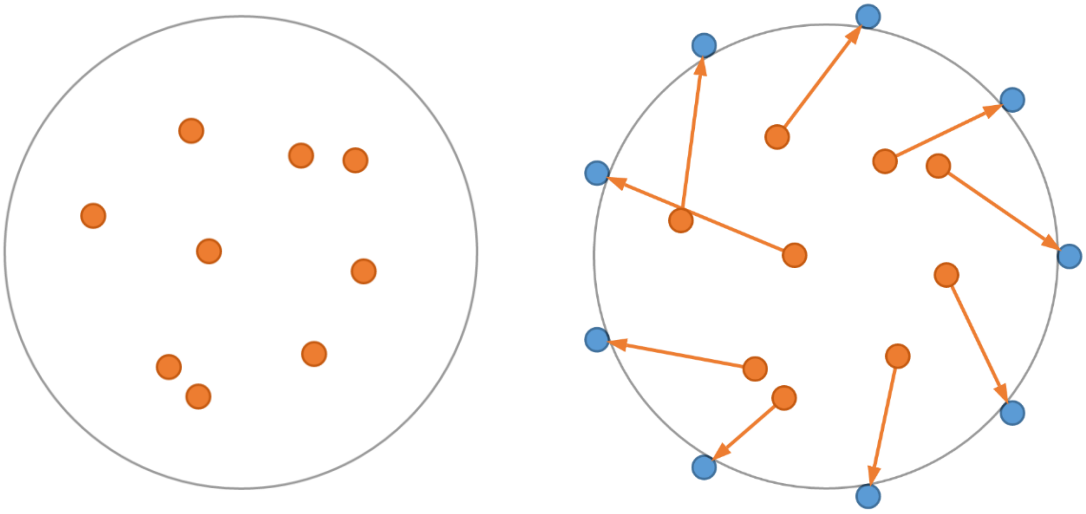
(a) (b)

Figure 3.8: Comparison of direct mapping and two staged mapping optimization. (a): Result of Direct mapping. (b): Result of two staged mapping optimization.

The radial layout tries to maintain the spatial information around the circular layout to by mapping the nodes directly along the radius to the nearest point on the circumference. This may help to trace back where the original location of one particular node was. However, the nodes are not arranged at equal distance along the circle so this approach still favors a radial placement which retains some of the original location information.

*Uniform layout*

As implied by the name, the uniform layout arranges the nodes evenly on the circumference. It trades the retention of some positional information for greater clarity when needed. The naive calculation process of the new node locations would be as follows. First, we determine the angle interval according to number of nodes $n$ to be mapped. Then we use trigonometric functions to compute $n$ fixed coordinates on the circle layout. Finally, we match each node to their new location according to their order in the original data set. Figure 3.9 is the illustration for a uniform layout.



(a) (b)

Figure 3.9: Uniform layout illustration. (a): The initial state: 9 nodes are selected and at their initial places. (b): The corresponding nodes are transformed to the circle by a uniform mapping.

However, Figure 3.9 (b) is not the appropriate solution of the node arrangement in this context because the algorithm orders the nodes according to their arbitrary item IDs in the original data set. Therefore, this approach may sometimes place the nodes in opposite directions to their nearest slots depending on which node was repositioned first. A better strategy would reduce such positional inconsistency after the mapping. That means, on average we would like the new locations to be as close as possible to the original locations, with preference to the first nodes that are repositioned.

We propose a force repulsion solution for this issue based on the force directed layout system found in the D3 Engine [28]. The force layout in D3 employs Verlet integration [78] to allow the specification of simple constraints and a Barnes–Hut simulation [79] that uses quad trees to accelerate charge interactions between elements [28]. The force layout adds repulsion or attraction forces to node elements to allow the physical simulation among them. Our solution adds custom forces to the original force layout model in order to constrain the nodes on the circular layout as well as let them repulse each other.

The simulation starts with the result of the radial node layout described in the previous section. On each iteration of the physical simulation, the algorithm performs a fine tuning of the nodes by letting them repulse from each other. When complete, the nodes are arranged on the circumference almost evenly. Generally, every node in the force simulation

layout process in the D3 Engine is assumed to be an infinitesimal point with equal charge force, which defines the strength of interaction with other nodes in the current selection, computed by Barnes–Hut simulation [22, 79]. The pseudo code for calculating the uniform layout by Barnes-Hub simulation is described as Algorithm 3.3 as follows.

```
Algorithm 3.3: CalcUniformLayout(O, R, Locations, Condition)
  Calculate the uniform node positions by force repulsion.

Input: O is the hub center, R is the hub radius, Locations is the
  node array to process for one hub, and Condition is the stop
  criteria for the physical simulation.
Output: Locations.

Start algorithm

 Calculate Variance for the current node layout;
 while Variance >= Condition:
     do: Simulate physical repulsion for one iteration and
     update Locations;
     Calculate current Variance;
 endwhile

End algorithm
```

For reproducibility, we note that the default charge force value in D3 is set to -30, which means the nodes will stop at positions 30 pixels away from each other. However, we have to apply an adaptive charge force according to the number of nodes in the current selection to determine the space between the nodes in a specific hub. We define this distance as $Dis_{charge}$, computed by the formula below.

$$Dis_{charge} = 2 * R * sin(\pi / n) \tag{3.1}$$

The meaning of Equation (3.1) can be simply explained as follows. Every node on the circumference should be pushed away from each other with a distance, and the distance should be the chord length between two neighborhood nodes, given the requirement that the nodes are uniformly arranged on the circumference. Equation (3.1) therefore defines the $Dis_{charge}$ value by the chord length between hub nodes.

Figure 3.10 illustrates the principle of the force based node positioning, which focuses on the forces applied on an individual orange node. The repulsion force are imposed to the nodes by setting adaptive charge distance parameter in D3 Engine, as illustrated in Figure 3.10 (a). In Figure 3.10 (a) the orange node receives combined repulsion force from all other blue nodes, as illustrated as a blue arrow.

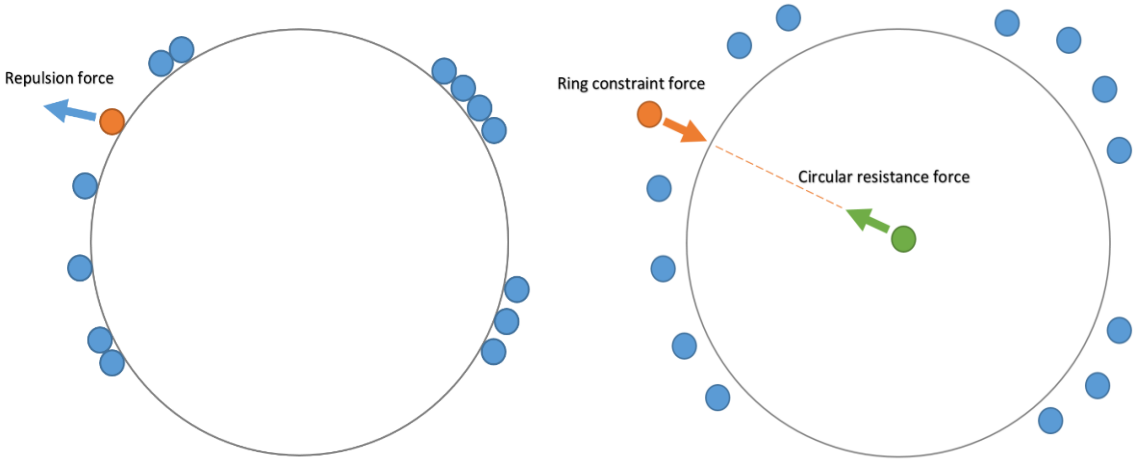



Figure 3.10: Force based uniform layout illustration. (a): State before applying repulsion force: the orange node bears combined repulsion force from blue nodes. (b): Intermediate status: circular resistance force and ring constraint force are applying on the orange node.

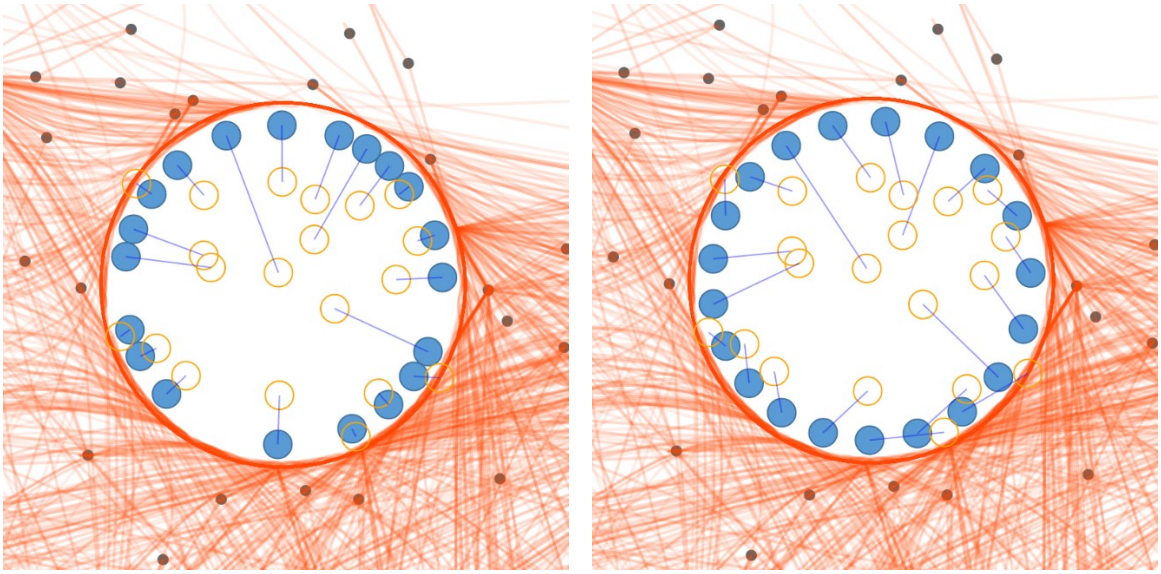
We then need to impose another two kinds of forces to all of the hub nodes, as shown in Figure 3.10 (b). One is the circle resistance force, which prevents the nodes going inside the circular layout, as represented by a green arrow; the other is the ring constraint force, which constrains the nodes and direct them towards the circle center once they deviate from the circular layout, as illustrated by an orange arrow.

To simulate the circle resistance force, we add another fixed node at the center of the circular layout, and add it to the node array for simulation, shown as a green node in Figure 3.10 (b). The position of the fixed node cannot be altered by any forces in the drawing area, and the charge force imposed on this node is specially set to negative infinity, to force other nodes away from the internal circle constantly.

To simulate the ring constraint force in each simulation step, we provide a new position that is located along the expected ring force direction for each node in the tick event provided by D3 Engine, to drag the node back to the circumference. The expected ring

force direction should be towards the circular layout along the radius, as illustrated by the orange arrow in Figure 3.10 (b). Typically we set this position on the circumference.

The simulation result using real data is shown in Figure 3.11. Visual glyphs are provided to facilitate comparison. Figure 3.11 (a) shows the initial state before force layout simulation while Figure 3.11 (b) illustrates the optimization result of the uniform layout. We can see that the nodes are moved by an incremental angle according to their relative positions. However, the order of the nodes are not altered, and the extent of movement is relatively mild, which helps to ensure that we retain the visual consistency between the initial and final states.



(a) (b)

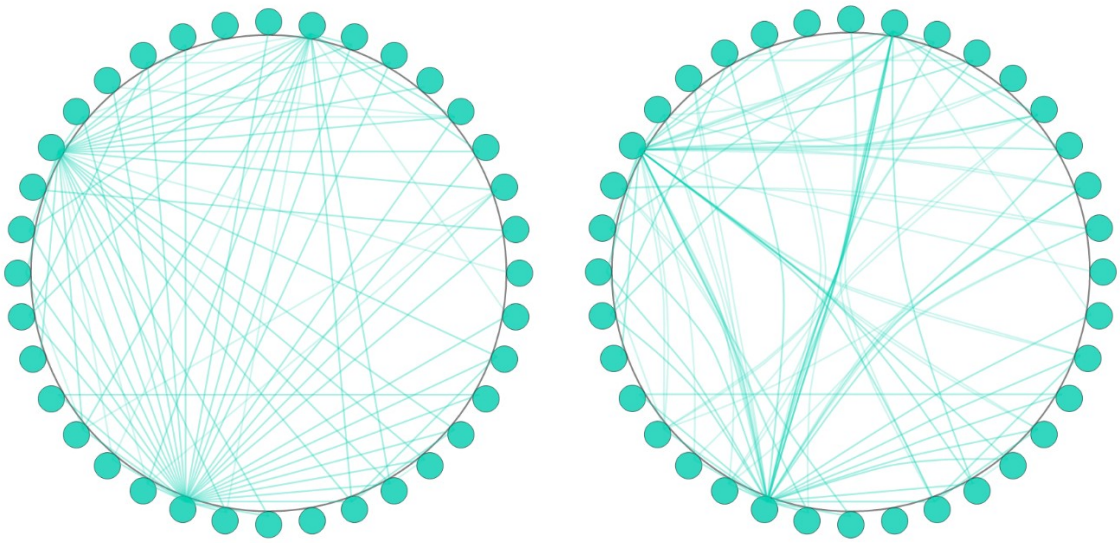
Figure 3.11: Comparison between radial layout and uniform layout. (a): Initial state before applying uniform optimization. (b): Final state of the uniform optimization.

### 3.2.3 Edge Layout Design

The edge layout is another crucial issue in our dynamic technique. When the sub layouts are generated and the node layouts are altered, the corresponding edges should also be taken into account simultaneously. We design edge layouts for both inside and outside of the circular layout. We define two different groups of edges here and we perform different layout algorithms separately. The inner edges are edges that connect the nodes on the

circular layout, while the outer edges involve other edges that would normally cross through the circular layout but have no relations with the hub itself.

Three modes can be selected for the inner edges, namely: linear bundle, force bundle, and peapod bundle. We will discuss the former two modes in this section and introduce the peapod model in Section 3.3 separately. To facilitate comparison, we firstly show a demonstration for linear and force bundles with 36 nodes in Figure 3.12 and then describe them respectively. Figure 3.12 (a) shows the linear bundle result, and Figure 3.12 (b) illustrates the force directed bundle result.



(a) (b)

Figure 3.12: Demonstration of linear bundle and force bundle with 36 nodes. (a): Linear bundle. (b): Force directed bundle.

*Inner edges: linear connections*

The most straightforward variant for node connections around a circle is the circular graph layout. In a circular graph layout the nodes are constrained to lie on a circle. The nodes are often evenly spaced so that they form the vertices of a regular polygon. In our model, the first strategy simply connects related nodes by a direct line segment with each other. The linear technique is illustrated as Figure 3.12 (a).

The linear layout can be sufficient in situations that involve less nodes and edges very well, since straight lines show the individual connection information distinctly. Therefore,

it may be suitable for the situation that requires accurate observation for a few individual links.

*Inner edges: force-directed edge bundling*

Compared to the linear connections, non-linear edge bundling is more appropriate to be applied on data sets that involve many nodes and connections. This bundling helps to reduce overall clutter while retaining high level trends of the edge directions. We integrated the force-directed edge bundling technique [19] in our model to utilize the benefit of non-linear bundling techniques. We choose force-directed edge bundling for several reasons.

First, no hierarchy level of the nodes are required by the FDEB algorithm and it is a self-organizing approach [19], which can adapt to most node-link graphs without further processing of the raw data. Second, the method achieves significant clutter reduction and retains clearly visible high-level edge patterns. For convenience we describe the main principle of the force-directed edge bundling technique here briefly as shown in Figure 3.13 from [19].

In Figure 3.13, two interacting edges $P$ and $Q$ are subdivided into segments. The position of edge end-points $P_0$, $P_1$, $Q_0$, and $Q_1$ remains fixed. A local spring constant $k_P$ is calculated for each segment of edge $P$ as $k_P = K / |P|$ to control the amount of edge bundling, where $K$ is a global spring constant, and $P$ is the initial length of edge $P$. Furthermore, an attracting force $F_e$ is used between each pair of corresponding subdivision points.

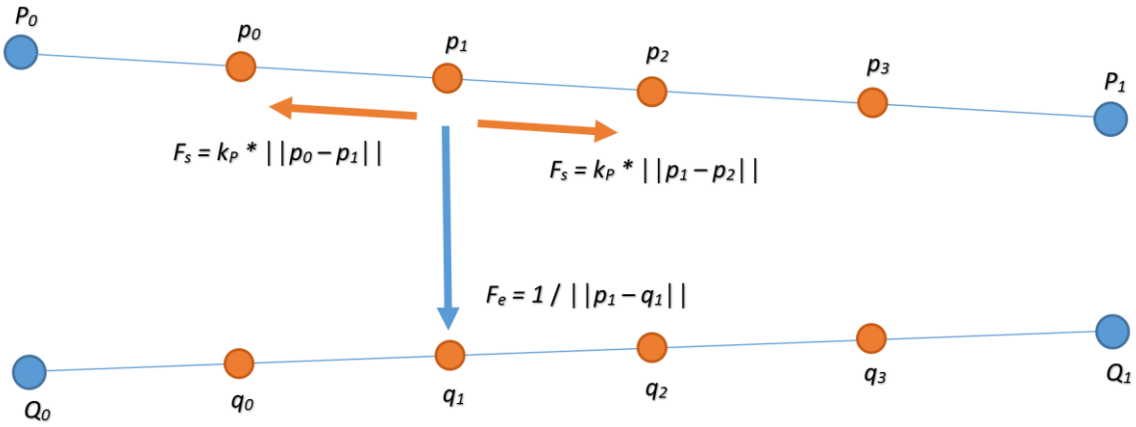



Figure 3.13: Principle of FDEB algorithm [19].

During each calculation iteration, the combined force on each subdivision point of each of the edges is calculated. The position of each subdivision point is updated by moving it a small distance in the direction of the combined force. For a subdivision point $p_i$ on edge $P$, the combined force $F_{pi}$ is a combination of the two forces $F_s$ and the sum of all $F_e$ defined as Equation (3.2), where $E$ is a set of all interacting edges except edge $P$.

$$F_{p_i} = k_P * (\|p_{i-1} - p_i\| + \|p_i - p_{i+1}\|) + \sum_{Q \in E} \frac{1}{\|p_i - q_i\|} \tag{3.2}$$

Figure 3.12 (b) demonstrates the result of applying the FDEB technique to a circular layout. The bundling result reduces clutter in favor of retaining high-level edge patterns. We can see from Figure 3.12 (b) that edges are bundled together to reduce visual clutter around some major nodes as well as revealing the hidden global pattern between edges.

*Outer edges: curving edges around hubs*

The term outer edges are defined relative to the inner edges. Specifically, outer edges are a set of edges that are not inner, a subset of which may pass through a circular layout. Our model aims at extracting information and reducing clutter in local areas while preserving the global structure of the main graph. Therefore, one of the key aspects for preserving the global view after generating the circular hub layout is the treatment of the outer edges. In particular, we want to prevent outer edges from crossing through hubs. We take advantage of spline interpolation algorithms to tackle the issue.

In numerical analysis, a cubic spline is a spline where each piece is a third-degree polynomial [75]. One established approach for interpolating a set of control points is cardinal interpolation [76], which is a special form of cubic hermit splines [76]. The cardinal interpolation principle can be described as follows, as illustrated in Figure 3.14.

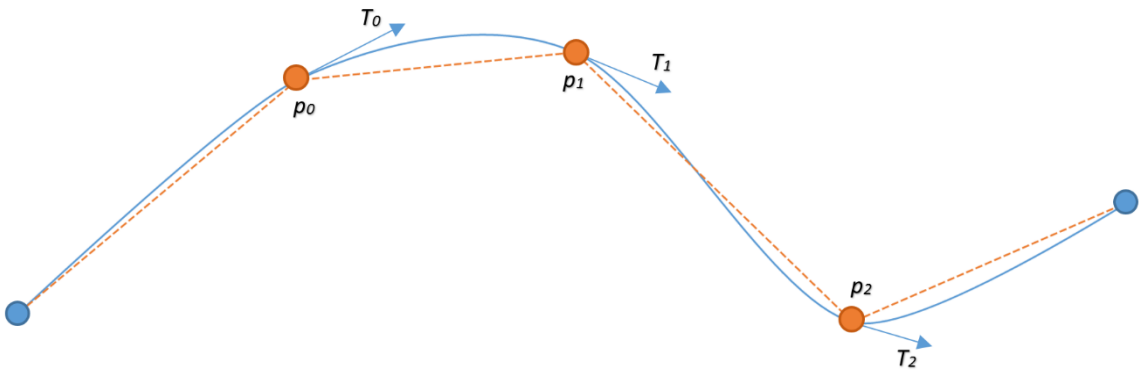



Figure 3.14: Cardinal interpolation principle.

The algorithm calculates two tangent values for the spline according to the control points, and uses them together with two end points to substitute into the 4 polynomials in Hermit spline functions. The formula for calculating the tangents is as Equation (3.3). In Equation (3.3), *tension* is a constant which affects the tightness of the curve. $T_i$ is the tangent value for control point $p_i$. The tension value defaults to 0.5 to smooth the spline. Taking the control point $p_1$ in Figure 3.14 as an example, the tangent value $T_1$ is calculated by $p_0$ and $p_2$ according to Equation (3.3).

$$T_i = tension * (p_{i+1} - p_{i-1}) \tag{3.3}$$

The number of control points used will determine how controllable our curves will be. This is a trade-off of precision vs computational efficiency. We define 15 control points to interpolate for an individual outer edge. Though experimentation, we determined that 15 control points is sufficient to provide enough control of the curves to bend them neatly around the hubs. Additional control points could be used, but the visual result becomes increasingly subject to diminishing returns and would only serve to increase the computational cost. Figure 3.15 shows an example of placing 15 control points for a single edge which are provided as the input of the cardinal interpolation.

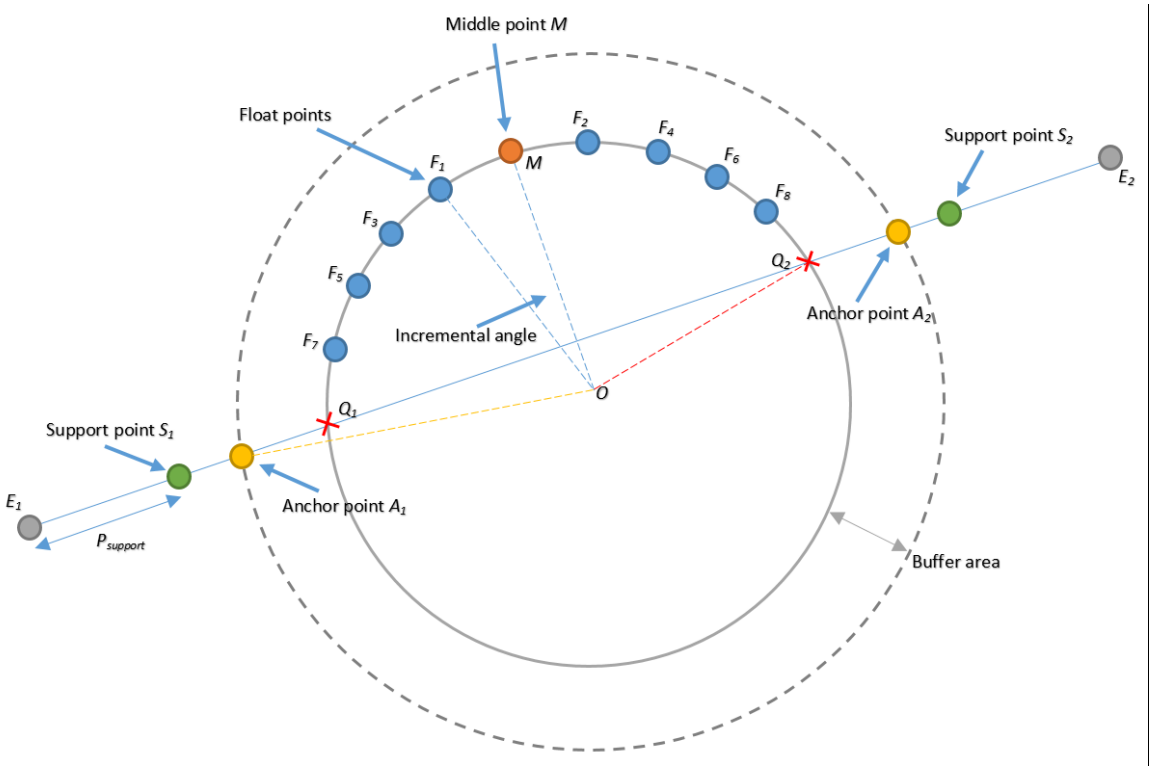
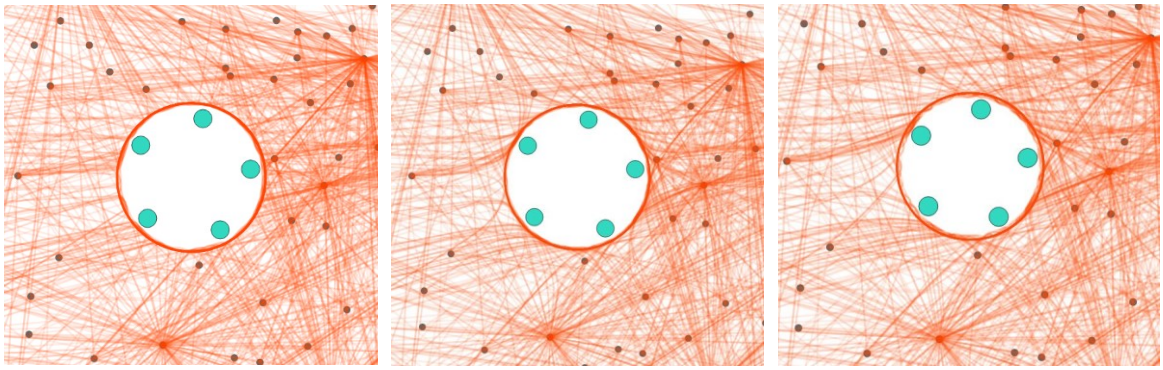



Figure 3.15: Computing 15 control points for an outer edge curve.

In Figure 3.15, the inner circle represents the circular layout of a defined hub, while the outer dashed circle is a "buffer area" for preserving the global edge trends around the circular layout. The buffer area is a ring around the original selected circular layout, it is defined by multiplying a constant factor $D_{buffer}$ to the radius of the circular layout. The buffer area is necessary for preserving the edge trends around the circular layout by smoothly connecting control points located on the circumference and on the edge. The selection of $D_{buffer}$ determines the size of the buffer area around the circular layout that can convey the edge trends. Figure 3.16 shows how increasing and decreasing $D_{buffer}$ can affect the edge bending result.

As illustrated in Figure 3.16, a smaller $D_{buffer}$ will allow less edge bending area and tighter bending effects, while a large $D_{buffer}$ will generate more mild bending effect within a broader area around the circular layout. For example, $D_{buffer}$ = 1.0 leads to a very tight edge bending effect in a narrow area around the layout, which gives us the illusion that the edges go directly through and behind the circle, as illustrated in Figure 3.16 (a). Conversely, Figure 3.16 (c) shows the result with $D_{buffer}$ = 2.0 and the edges are gently bended in a larger area. Figure 3.16 (b) demonstrates the more balanced result of selecting $D_{buffer}$ = 1.5.



(a) (b) (c)

Figure 3.16: Edge bending results with different $D_{buffer}$ values. (a): $D_{buffer}$ = 1.0. (b): $D_{buffer}$ = 1.5. (c): $D_{buffer}$ = 2.0.

Note that the nodes in Figure 3.15 represent control points instead of the data items on the circular layout. The blue line going through the circular layout to be bended is denoted by *E*, with both ends represented by grey nodes. The high level procedure for building the control point array is described in the pseudo code in Algorithm 3.4.

**Algorithm 3.4: BuildControlPointArray (O, R, $E_1$, $E_2$, $D_{buffer}$)**
Build a control point array for one intersecting edge E going through one hub.

**Input:** O is the hub center, R is the hub radius, $E_1$ is the start point of edge E, $E_2$ is the end point of edge E, and $D_{buffer}$ is the buffer area radius for the current hub.
**Output:** control point array C.

**Start algorithm**
```
Add E1, E2 to A at corresponding index: C[0], C[14];
Calculate point M, then add M to C, at index: C[7];
Calculate 8 float points F1 – F8 to corresponding indexes in C;
Calculate 2 anchor points A1, A2, add A1, A2 to C[2], C[12];
Calculate 2 support points S1, S2, add S1, S2 to C[1], C[13];
Return C
```
**End algorithm**

For clarity, we explain each step in the pseudo code as follows. Firstly, we add both ends $E_1$ and $E_2$ of edge *E*, as illustrated by grey nodes in Figure 3.15, into the control point array *C*. Then, we calculate the middle point *M*, as illustrated by an orange node on the circle, according to the circular layout radius *R* and slope of *E*.

Next, we calculate 8 float points $F_1 – F_8$. The blue nodes on the circular layout represent 8 float control points to smooth the spline near the circular layout to generate a more controlled visual effect around the circle. The more fixed points we have, the better fit to the circular shape can be achieved. We iteratively calculate 8 float control points by adding either a positive or a negative incremental angle to the middle point *M*. In each iteration, we calculate two points on both side of *M*, then add them to the corresponding indexes of the control point array *C*.

We locate the float points on the circumference evenly between middle point *M* and point $Q_1$ and $Q_2$. $Q_1$ and $Q_2$ are the intersections of edge *E* and the circular layout, as represented by red crosses in Figure 3.15. Taking the calculation for $F_2$, $F_4$, $F_6$ and $F_8$ as an example: the incremental angle $I_i$ for each float point $F_i$ is determined by angle $MOQ_2$, according to Equation (3.4), where $Angle_{middle}$ denotes the angle between line segment *MO* and positive x-axis, and *n* is the number of float control points on each side of *M*.

$$I_i = Angle_{middle} - (i \,/\, 2) * (MOQ_2 \,/\, (n \,+\, 1)) \tag{3.4}$$

Then, we calculate anchor points, $A_1$ and $A_2$, as represented by yellow nodes in Figure 3.15. These are the most important points to preserve the global edge trend around a circular layout. They are located on the intersections of the edge *E* and the buffer area. We calculate the anchor points using the cosine value determined by the distance from center *O* to edge *E*, and the buffer area radius $R_{buffer}$.

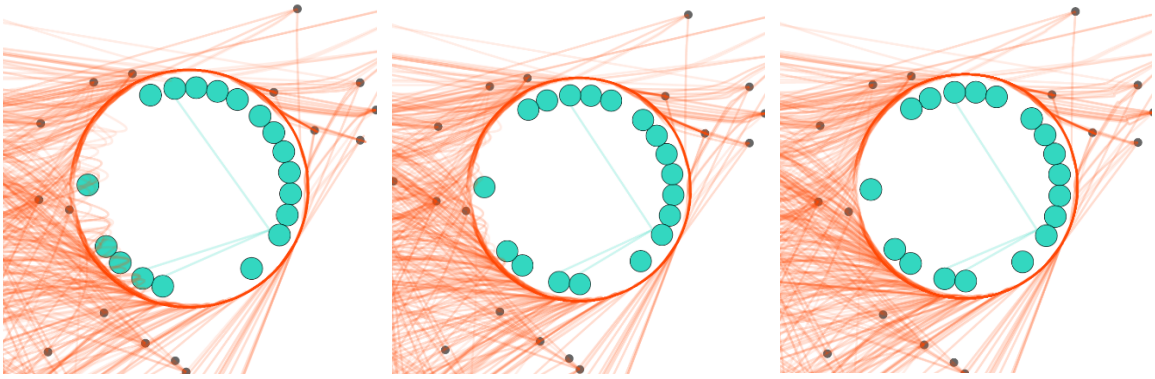
Finally, "support points" $S_1$ and $S_2$ are calculated. A support point is used for reinforcing the cardinal interpolation for edges that have ends locates far away from the circular layout. The reason why this is necessary is because cardinal interpolation interpolates a control point using its neighbors on both sides. The derivative of the current control point is determined by the vector determined by the points before and after it. If one of the neighbors is located at a far distance, the derivative may be a considerably large value. As a consequence, it causes the spline shape to become sharp enough to intersect with the circular layout.

We set the support points on edge *E* between one anchor point and its corresponding edge end point, as illustrated in Figure 3.15 by green nodes. The support point $S_1$ is located

at a percentage distance from $E_1$ to $A_1$, and the percentage is defined by $P_{support}$. Generally, the situation differs for different end points and edges, and we should choose different $P_{support}$ for different situation, so we calculate the adaptive $P_{support}$ value according to Equation (3.5), where *dis* denotes the distance from one edge end point to the corresponding anchor point, and $dis_{max}$ denotes the maximum distance allowed in the graph.

$$P_{support} = dis \,/\, dis_{max} \tag{3.5}$$

To clarify, we demonstrate how the selection of support point positions effect the result in Figure 3.17. Figure 3.17 (a) illustrated the result of setting $P_{support} = 0.25$, and we can see that there are several strange curves going through the circular area, which exhibit "kinks". We then set $P_{support}$ to 0.5 under same condition, as shown in Figure 3.17 (b). The amount and extent of the edges that going through the circle are reduced. In Figure 3.17 (c), we see no "kink" issues by setting $P_{support}$ to 0.75. From Figure 3.17 we can state that the support points can alleviate the "kinks" and contribute to a smoother shape around the anchor points.



(a) (b) (c)

Figure 3.17: Edge bending results with different $P_{support}$ values. (a): $P_{support} = 0.25$. (b): $P_{support} = 0.5$. (c): $P_{support} = 0.75$.

Finally, all the control points in *C* are interpolated with cardinal spline functions. We test the algorithm with a variety of cases using real data sets. Some of the typical results for a single circular layout are demonstrated as Figure 3.18.

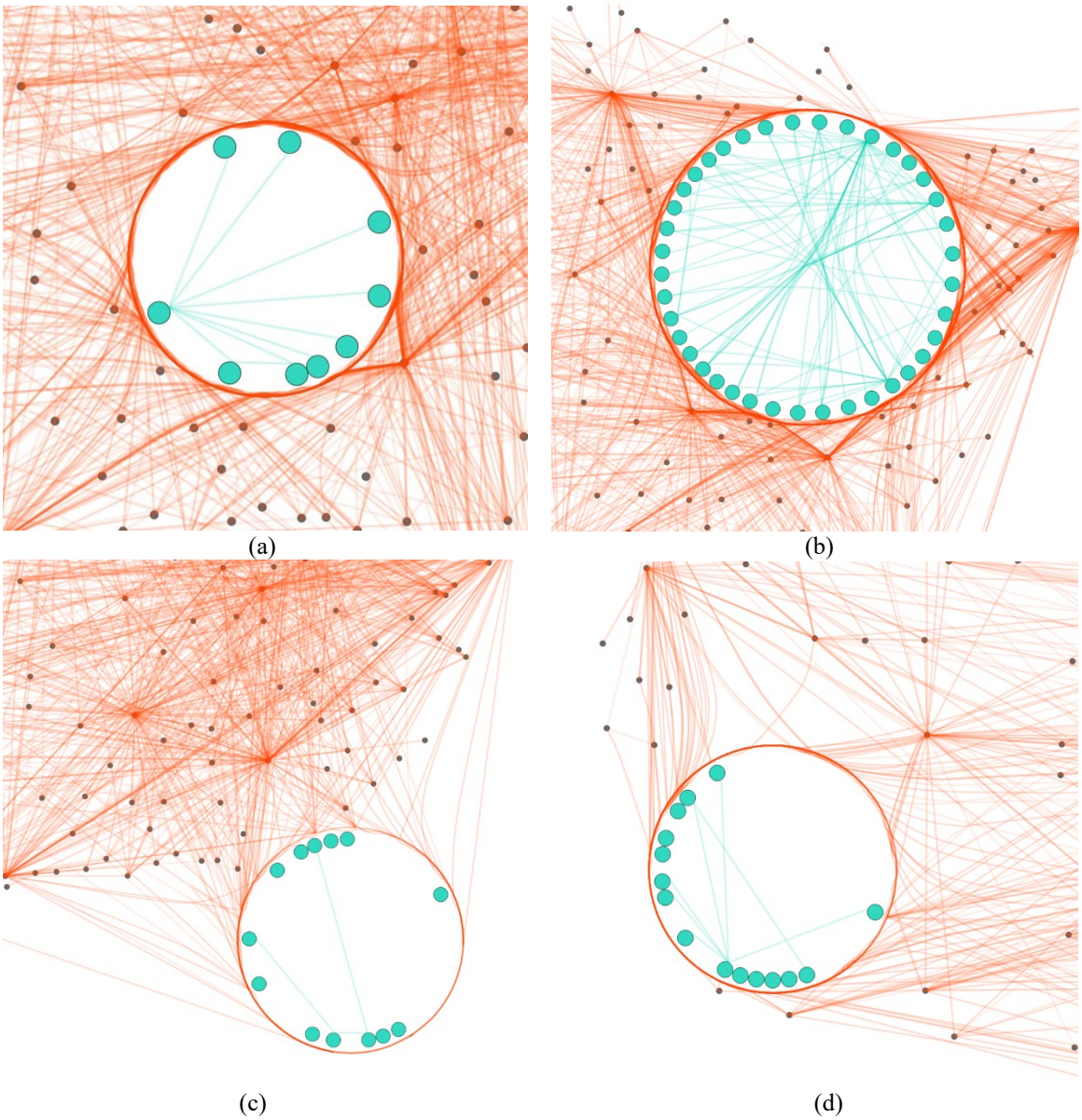


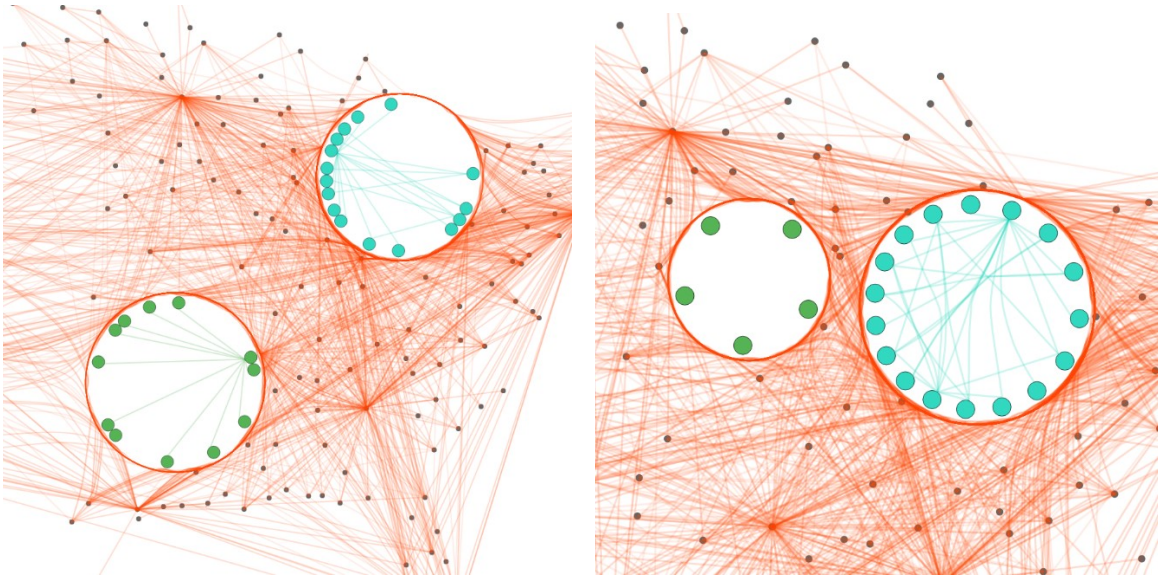

Figure 3.18: A variety of test cases for single circular layout. (a): A small circular layout with linear bundle, $D_{buffer}$ = 1.5. (b): A large circular layout with a force bundle, $D_{buffer}$ = 1.5. (c): A circular layout locates at the boundary of the main graph with linear bundle, $D_{buffer}$ = 1.5. (d): Another boundary test with $D_{buffer}$ = 1.5.

In terms of manipulating multiple circular layouts in one graph, there is an issue that can arise if the buffer areas from different circular layouts conflict when they move too close to each other. We offer a solution to this issue by introducing adaptive buffer areas for dynamic circular layouts, to allow the updating of $D_{buffer}$ automatically. We set the maximum $D_{buffer}$ value to 1.5, which means that in most cases the circular layout has a

buffer area 1.5 times larger than the original selection. However, when two circular layouts interact with each other at a closer distance less than 1.5 times of the sum of their radiuses, we calculate $D_{buffer}$ for both layouts according to Equation (3.6), where $D$ denotes the distance between two circle centers, and $R_1$, $R_2$ represents the radius for the circular layouts.

$$D_{buffer} = D \,/\, (R_1 + R_2) \tag{3.6}$$

For multiple circular layouts, we iteratively calculate the $D_{buffer}$ value array for the current circular layout with all other layouts, and set the current $D_{buffer}$ to the minimum value in that array. Some representative results for multiple layouts are shown in Figure 3.19, Figure 3.20, Figure 3.21, Figure 3.22, Figure 3.23 and Figure 3.24.



(a) (b)

Figure 3.19: Test cases for two circular layouts. (a): Two layouts with radial layout and linear bundle, $D_{buffer}$ = 1.5. (b): Two layouts close to each other with uniform layout and force bundle, using adapted $D_{buffer}$ = 1.15.

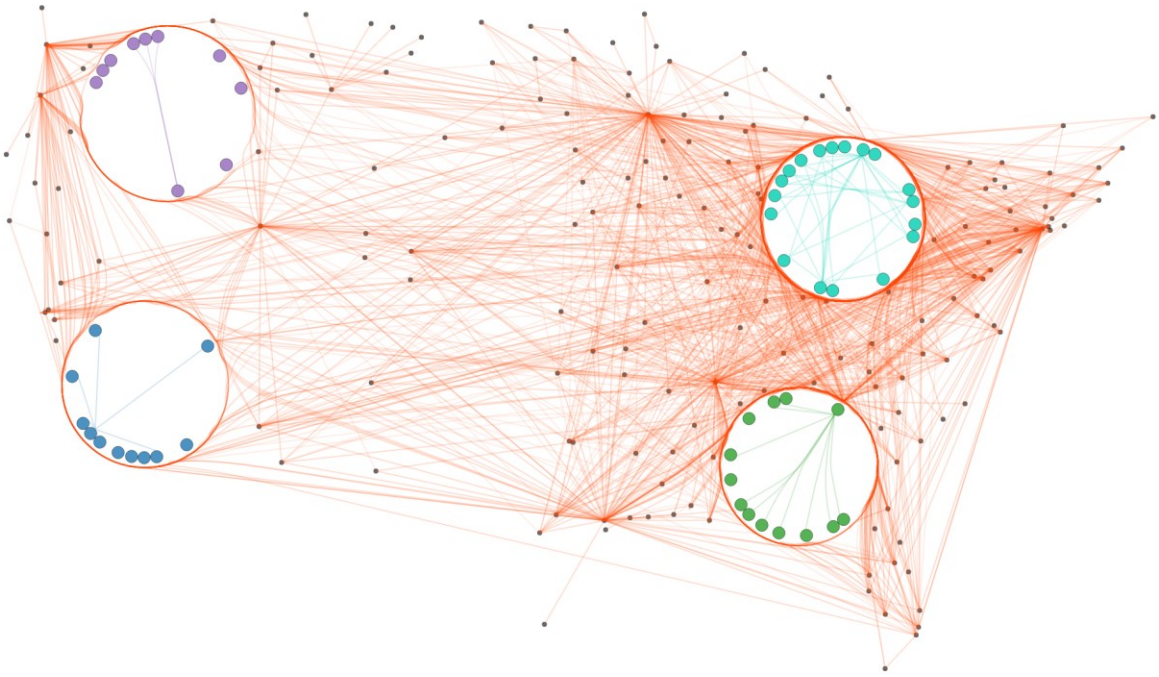


Figure 3.20: Test case for two groups of circular layouts located away from each other in the left and right area of the main graph, with radial layout, force bundle, and $D_{buffer}$ = 1.5.

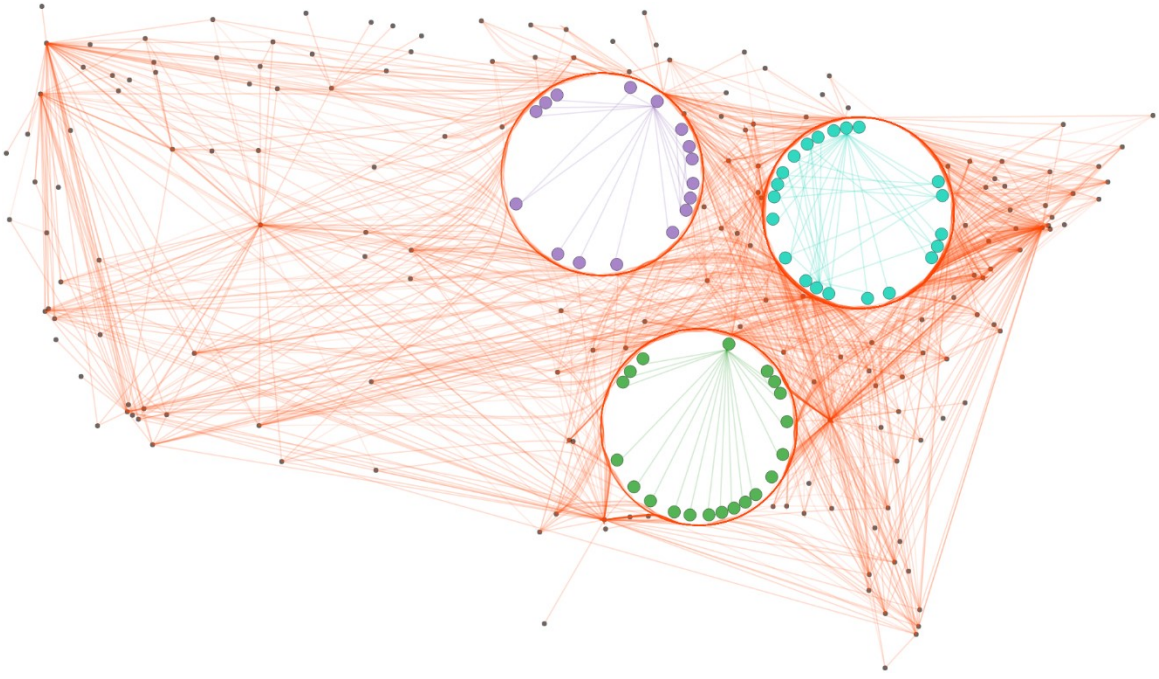


Figure 3.21: Test case for 3 circular layouts that are close to each other, with radial layout, linear bundle. Purple layout is with $D_{buffer}$ = 1.32, green layout is with $D_{buffer}$ = 1.38 and blue layout is with $D_{buffer}$ = 1.32.

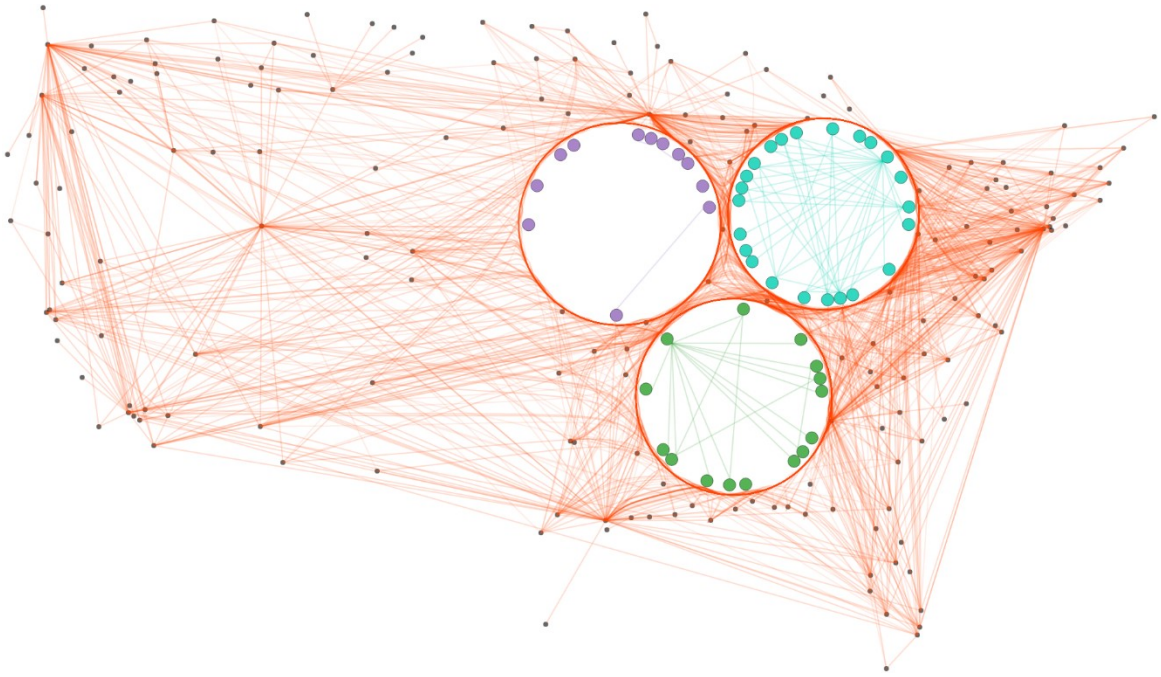


Figure 3.22: Test case for 3 circular layouts very close to each other, with radial layout and linear bundle. Purple layout is with $D_{buffer}$ = 1.04, green layout is with $D_{buffer}$ = 1.05, and blue layout is with $D_{buffer}$ = 1.04.

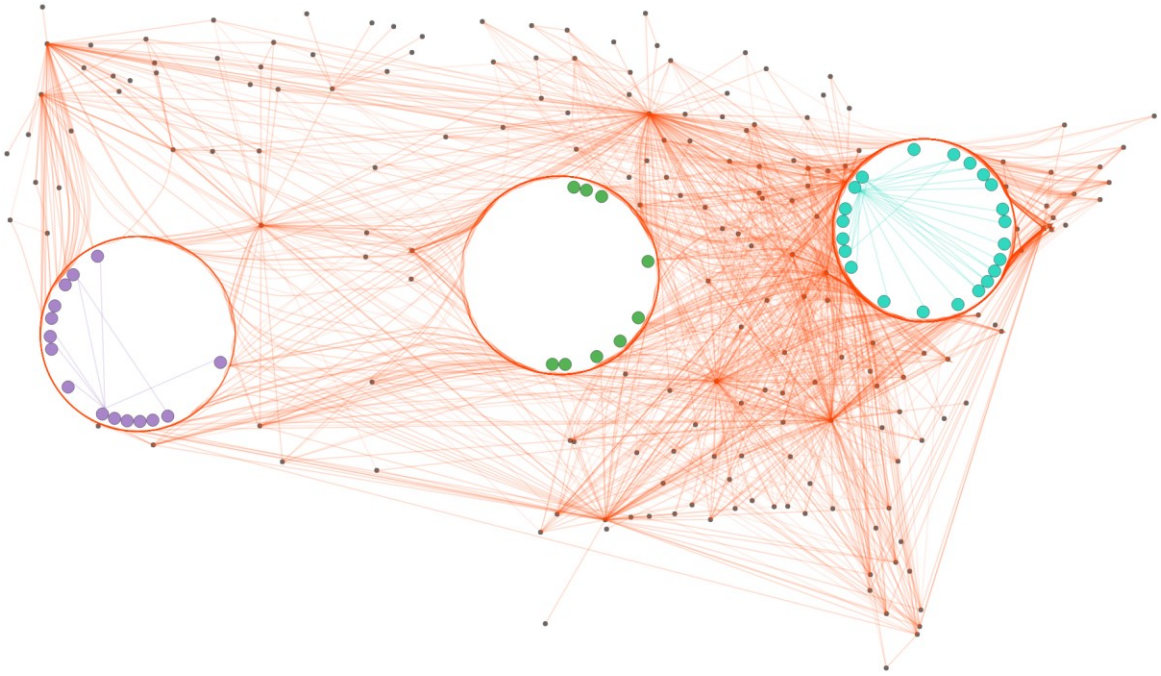


Figure 3.23: Test cases for three circular layouts that form a straight line, with radial layout, linear bundle, and $D_{buffer}$ = 1.5.

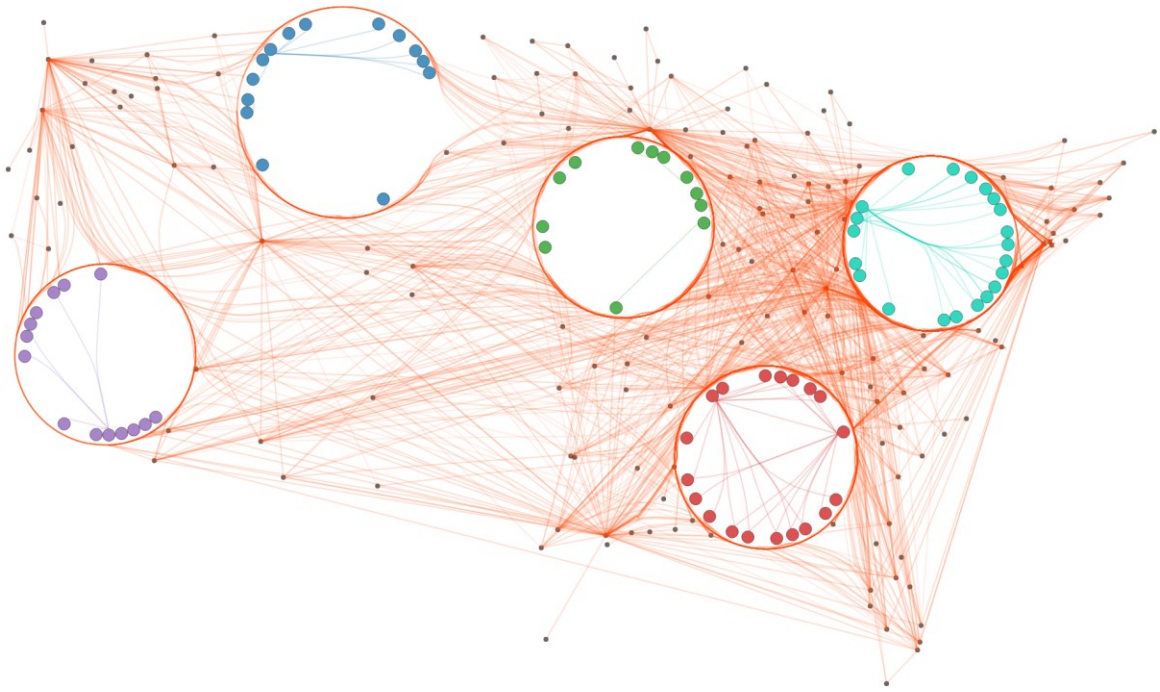


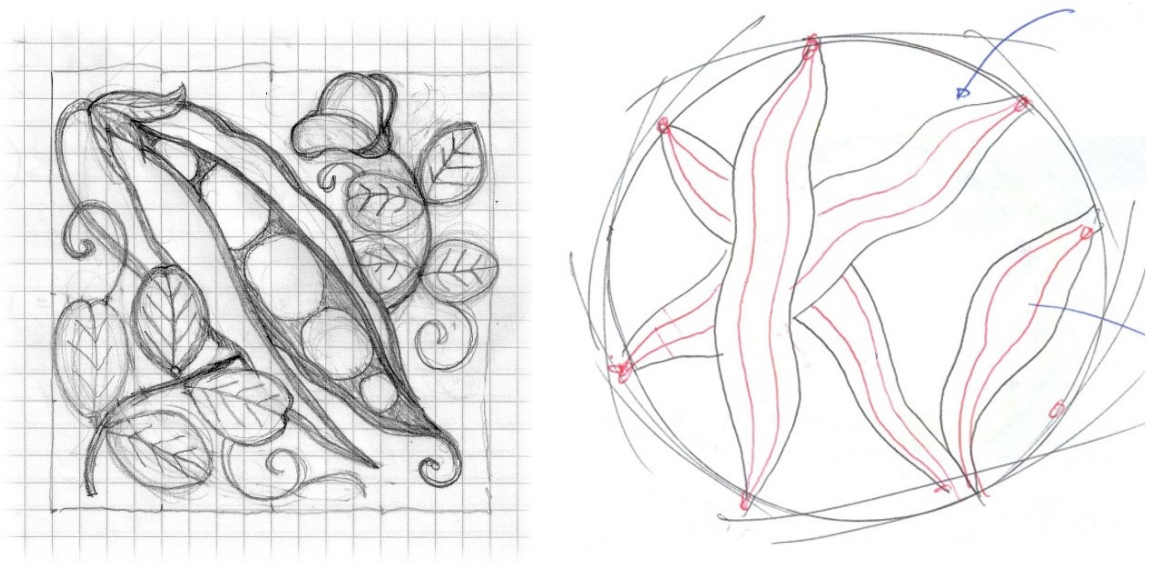
Figure 3.24: Test case for five circular layouts with radial layout and force bundle. Purple layout is with $D_{buffer}$ = 1.5, dark blue layout is with $D_{buffer}$ = 1.5, green layout is with $D_{buffer}$ = 1.49, red layout is with $D_{buffer}$ = 1.48, sky blue layout is with $D_{buffer}$ = 1.48.

From the previous results we can see that this geometry interpolation approach is suitable for generating outer edge layouts for a wide variety of cases. It is worth mentioning that the approach for outer edge layout is a tradeoff between the clarity offered within the circular layout and the use of the narrow space around it, because it pushes any edges that are currently unrelated to the selected nodes together. However, the use of this technique operates under the assumption that this dynamic temporary tradeoff is desirable in order to more closely examine a set of nodes that are of particular interest. So it is not unreasonable to impose a temporary deformation on the remainder.

In this section, we stated how our approach can assist the exploration of cluttered areas in pervasive node-link graphs with dynamic circular layout designs. This represent a conscious, yet temporary spatial distortion. Spatial position is a very important aspect in geo-visualization systems, and the distortion of those positions should not be done casually but with purpose. But it is necessary to understand the local area in highly cluttered graphs. Moreover, we will introduce an additional feature in Section 3.5 that can serve to recall the original positions with visual glyphs.

## 3.3 Peapod Model

We further propose a novel approach to depict time series data within our dynamic circular layout by introducing a design based on the metaphor of peapods. Figure 3.25 shows the inspiration and initial design of the peapod model.



(a) (b)

Figure 3.25: Peapod model inspiration. (a): A traditional art drawing of peapod sketch from [86]. (b) An early sketch of the peapod model design.

### 3.3.1 Peapod Layout Design

We study the peapod geometry and formulate the design of the peapod model, as illustrated in Figure 3.26. We use a two-way stream to represent the directional flows of the data, and cut the peapod body into subdivisions according to time period intervals. Both ends of the peapods connect to the starting and ending nodes $N_1$ end $N_2$ in the circular layout. We then define the baseline of our pod as the line segment between $N_1$ and $N_2$, where both ends represent the nodes on the circular layout. The line segment $N_1N_2$ are divided evenly into a number of time series segments, to match each data value to the coordinates on the baseline, as illustrated in Figure 3.26. Start and end in Figure 3.26 represent where the data flow

starts and ends in the peapod. The shapes at the end do not convey data values but simply provide the geometric connection.

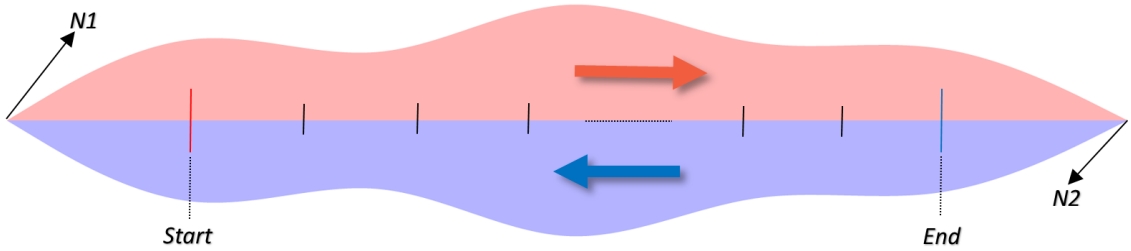



Figure 3.26: A two-way peapod prototype design.

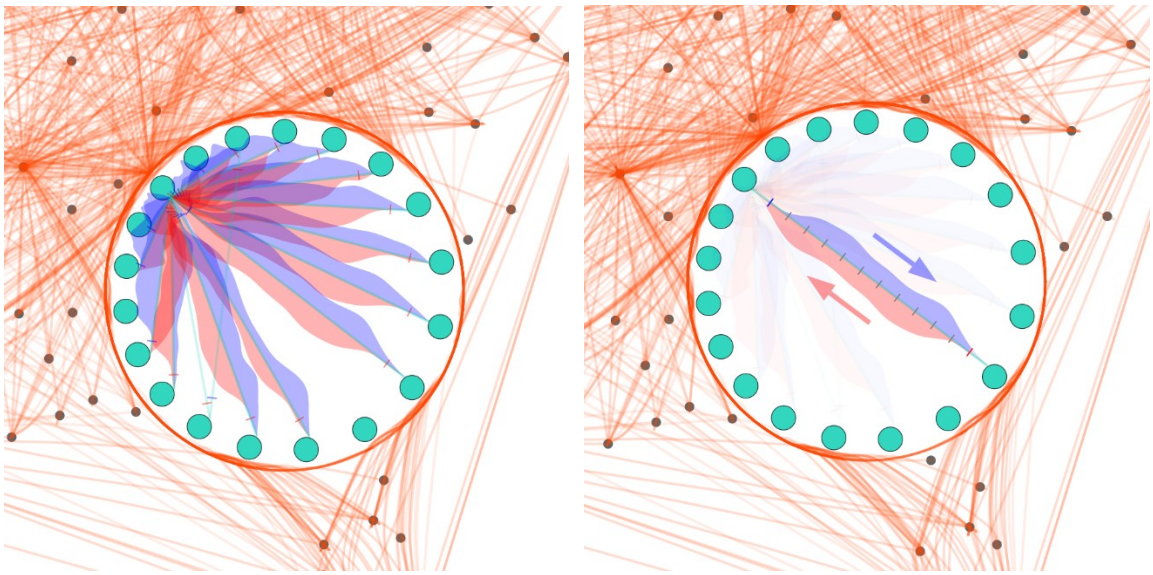
With the peapod model, the system matches the data values to both sides of the baseline and renders a new horizontal peapod. Then, a geometric rotation transform is performed on the generated peapod to adapt to the circular layout. To preserve the consistency of the pod shapes and limit the width of the pod bodies, we also perform a scaling operation on the data items. The data is scaled up by the percentage from their minimum value. For a time series data pod $P$ with $n$ data entries, the algorithm for scaling the data is described as follows. First, the maximum and minimum of the series data is calculated and stored, formulated as $p_{max}$ and $p_{min}$. For each data item $p_i$ in $P$, we calculate the scaled match of the item according to Equation (3.7), where $P_{match(i)}$ is the geometry matched value for the $i_{th}$ data item in the data set, and a *tensor* is a constant to control the width of the peapod.

$$P_{match(i)} = tensor * (p_i - p_{min}) / (p_{max} - p_{min}) \quad (3.7)$$

The benefit of the scaling process is that the variation between data items can be interpreted more readily, since there exist prevalent time series arrays that consist of very large data values. For example, the traffic flows between two airports may include some large numbers over the year, say 120,000. Therefore it is difficult to reflect variations along the time period by absolute values. This approach may generate similar widths among different peapods, but it is a tradeoff to convey the most essential information.

In terms of smoothing and shape control, we employ the B-spline interpolation [76] process on the peapod data set after previous steps. Figure 3.27 (a) shows the generated

peapod circular layout result for 19 nodes with 12 time series data entries in each pod. We can see how the peapods can convey the time series data in a generated circular layout. It is worth mentioning that the straight line connections between two individual nodes in Figure 3.27 (a) represent the path that conveys no data flows in the current time period. The short red and blue dashes in the peapods remind the direction of the data flow of the corresponding half pod.



(a) (b)

Figure 3.27: Peapod examples for 19 nodes in a circular layout. (a): The peapod layout. (b): Selection of a single peapod using mouse hover function. Axis and directional cues are displayed to facilitate identifying data intervals.

It is worth mentioning that there exist some cases when we have a many peapods in one circular layout concurrently. We take these situations into account and include a simple hover technique to tackle the issue. The remaining peapods fade out automatically when a single peapod is selected as the focus pod by hovering the mouse on it, as illustrated in Figure 3.27 (b). So we temporarily remove the issue of peapod overlap with a high volume of data. When hovered over a peapod, the triangle areas on both end of the select peapod are dimed to highlight the main pod body that actually conveys data items. Moreover, two corresponding directional cues are provided to illustrate the flow direction when hovering the mouse.

To enhance the flexibility of the model, a filtering slider is provided in the control panel area as shown in Figure 3.1. Any range between the maximum and the minimum of the data can be filtered out by the slider operation according to their average values during the specific time period, and the peapod model in the main view will conduct an update correspondingly. The above approaches can not only alleviate the peapod overlap but also provide a flexible way to analyze the data.

To complement this fine grained information we also designed a higher level hub stream view based on stacked graphs to show additional categorical information for a group of peapods.

### 3.3.2 Hub Stream View

Since the space is limited in the circular layout, we only integrate the peapod shape into the layout to illustrate the flow trend between individual nodes. Therefore, some broader data patterns cannot be explored in the peapod circular layout. We introduce a styled stacked graph solution called “hub stream view” to visualize more aggregate data. The hub stream view is located at the middle bottom area of the interface and it serves as a complementary feature to the previous peapod model.

Similar to the peapod model, we compute the aggregated data of all the current edge selections for each category across the timeline from the original data set and then perform a scaling operation on the results. The data are then stacked together by an offset function with color generator defined by D3 Engine [28].

The hub stream view provides more detailed information about the time series data. It models aggregated peapods as a stacked stream to show categorized information that represents the proportion of different data categories during the time period. These categories might include, for example, the business share of various airlines on the selected airline routes. Figure 3.28 shows a hub stream demo for a random selection of the US airline dataset. Each stack in the graph represents the passenger flow conveyed by an individual carrier among 25 different carriers in the United States.

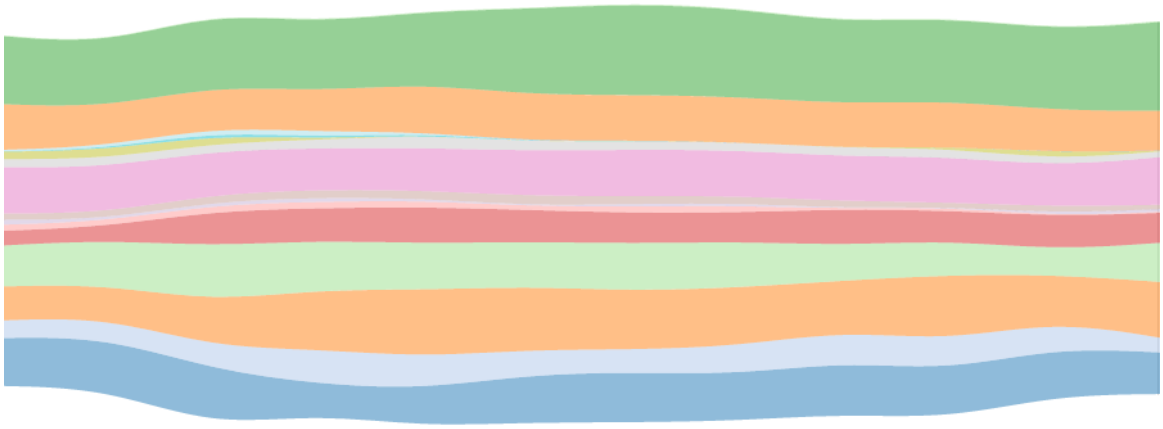
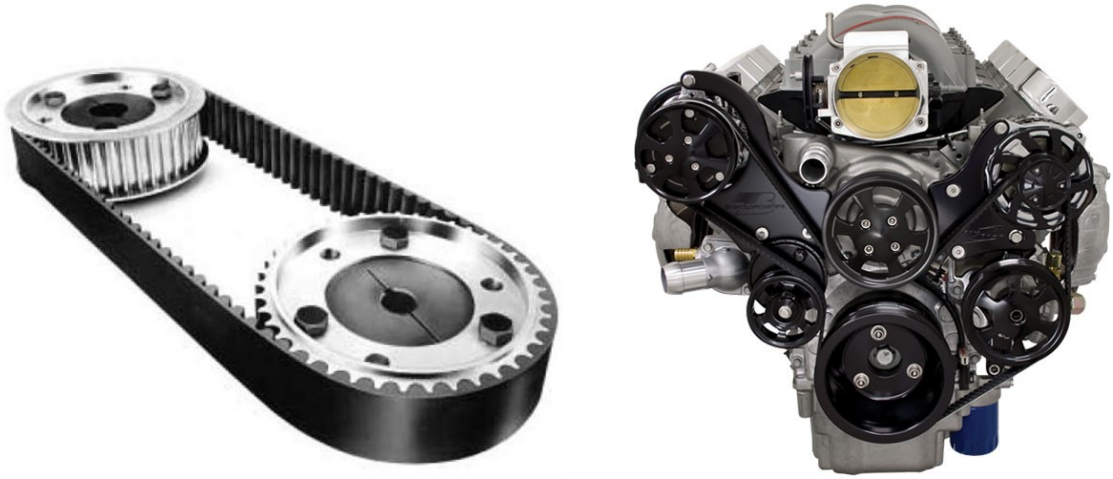


Figure 3.28: A demonstration of the hub stream view.

## 3.4 Hub Model

We propose a novel approach to visualize relations between different groups of nodes in traditional node-link graphs. The inspiration stems from the pulley systems found in mechanical manufacturing industries, as illustrated by Figure 3.29. We connect the circular hub layouts in a large graph by hub level “peapods” similar to those discussed in Section 3.3 to reveal the aggregated patterns between them.



(a) (b)

Figure 3.29: Hub model inspiration: In mechanical manufacturing industry, pulleys are often connected by belts around them to compose a pulley system. (a): A simple timing pulley system in a transmission model. (b): pulley system in BLK13450 Top Mount Automobile Engine.

### 3.4.1 Hub Layout Design

We design a model that combines the peapod concept with the circular layout to show aggregated information. We abstract a circular layout as a single hub and visualize the data flows among different hubs while maintaining the relative positions of multiple hubs in the main graph. We design hub level "peapods" around the idea of "timing belts" that connect the pulleys in a pulley system, the hubs are connected by data flows that demonstrate aggregated time series information.

The geometric computation of "hub pods" are similar to the "peapods" in previous sections, however we cut the peapods in half to imitate two-way "timing belts". The blue belts represent the aggregated data flow from the source hub to the target hub, while the red belts depict the data flows in opposite direction. We firstly compute 4 reference points for each pair of hubs to locate the belts between them. Then we set the direct line segments formed by the 4 reference points as base lines for generating the belts. We then employ the calculations we already described in Section 3.3.1 for building peapods. The process for calculating the 4 reference points $U_1$, $U_2$, $L_1$ and $L_2$ is described as following, as illustrated in Figure 3.30.

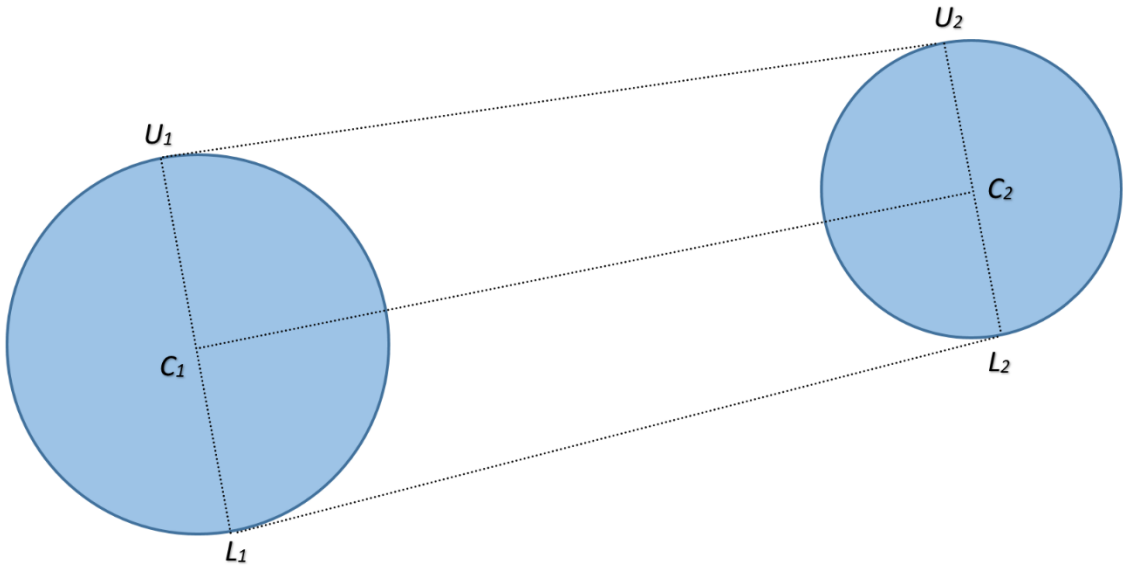



Figure 3.30: Geometry scheme for calculation of 4 reference points.

First, the *slope* of line segment $C_1C_2$ between two circle centers $C_1$ and $C_2$ are calculated. Second, the angles between the perpendicular diameters $U_1L_1$, $U_2L_2$ and the *x-axis* are computed by the *slope*. Finally, the control points $U_1$, $U_2$, $L_1$ and $L_2$ are generated by standard trigonometric functions.

Figure 3.31 illustrates the result of our hub model that shows aggregated time series data between different hubs on the US aero data set. When the hub view is selected in the control panel, other edges in the main graph will fade out automatically to highlight the relationships between hubs. Three different hubs are generated on the right side of the main graph. The 3 hubs represent 3 different zones of heavy air traffic around several main passenger terminals in eastern United States respectively, and we can see from Figure 3.31 that our hub model can present with the higher level relations among different groups of nodes. Figure 3.32 shows the dynamic placement result of moving the purple hub to the left area. The layout in Figure 3.32 now shows the data flows across the main graph.

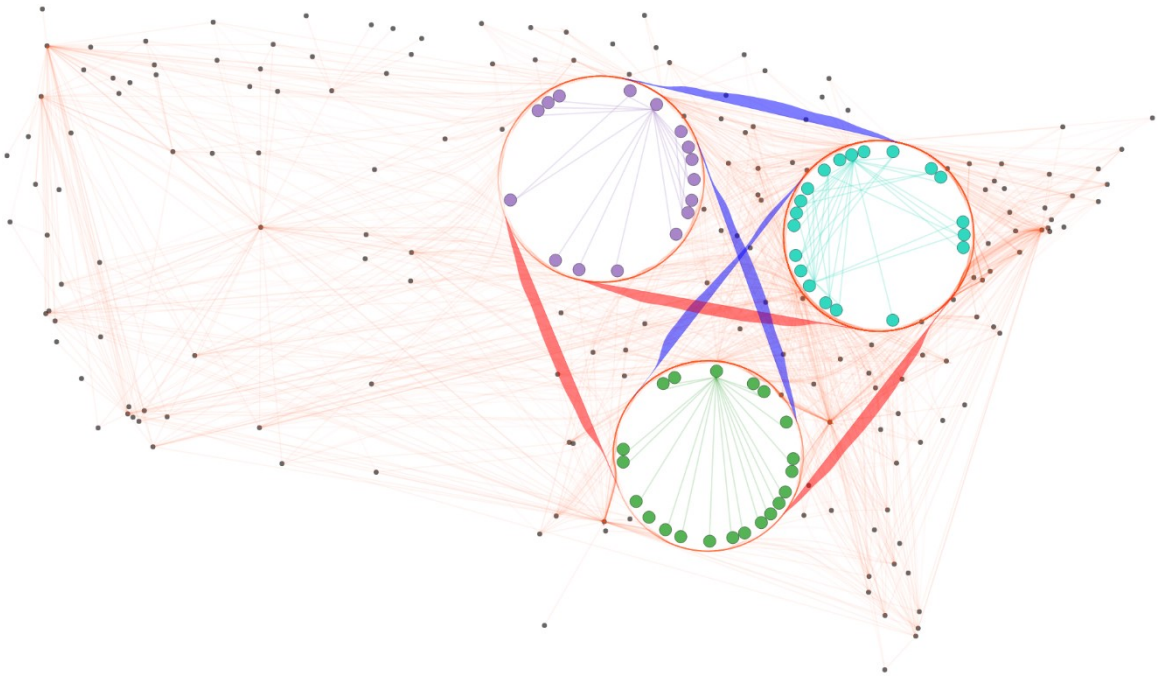


Figure 3.31: Hub layout demonstration showing aggregated time series data among 3 different hubs.

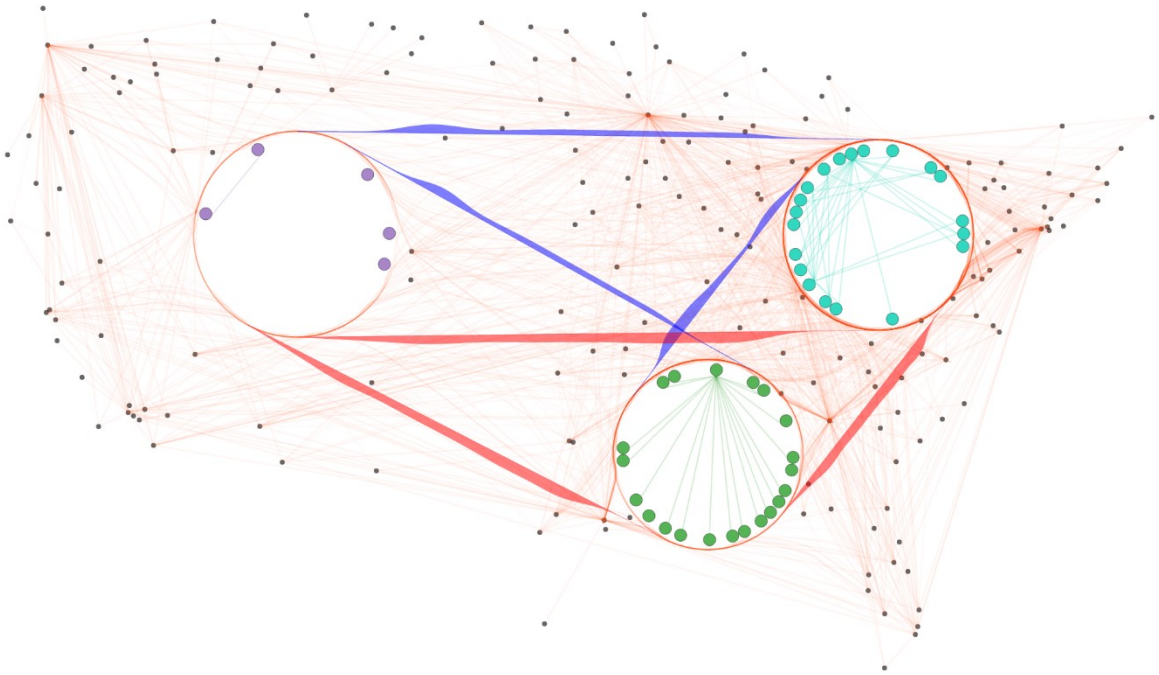


Figure 3.32: The layout after performing a dynamic placement of the purple hub from Figure 3.31.

Similar to the peapod model, the hub model also incorporates mouse hover functions to facilitate identifying directions and data intervals by axis. The hub model can show aggregated data flows among the current nodes of interest while preserving the relative spatial information and inner visualizations for every group of nodes in the main graph.

However, the hub view does not visualize the individual connections across the hubs, because the individual connections across hubs are not always suitable for displayed in the main view. The reason is described as follows. First, the connections may go through internal areas of the hubs or the nodes around the hub, which may lead to some information loss of the inner connections inside the hubs. Second, the connections between hubs are of a complex structure and may even clump together, therefore it is reasonable to have an alternate organization of the nodes and edges. We propose a complementary ensemble hub view to address the limitations of the aggregated hub model in the next section.

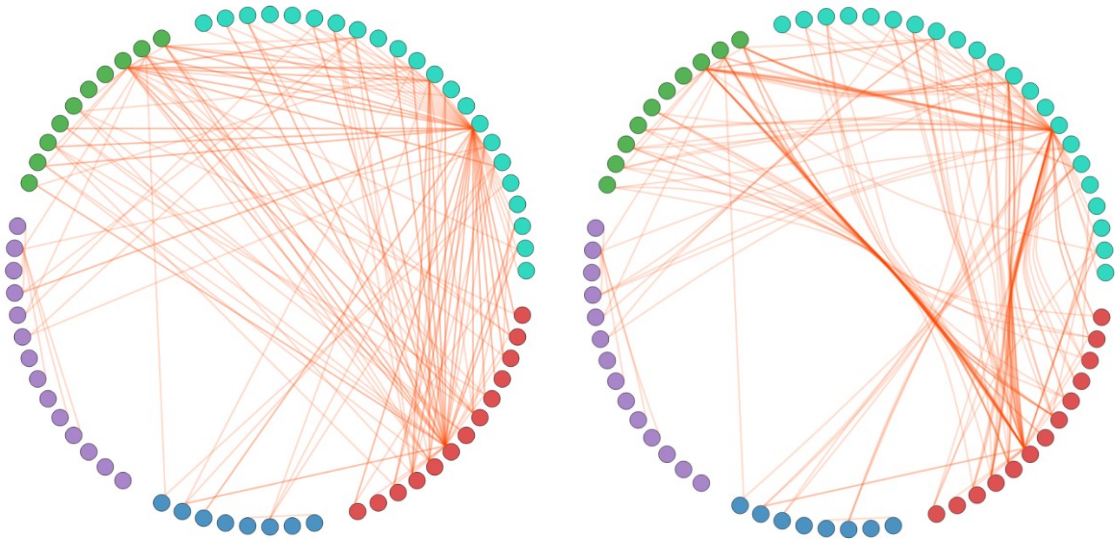
### 3.4.2 Ensemble Hub View

The individual connections between different hubs often have complicated structure and interfere with the visualization of the hub itself. The ensemble hub view integrates all

selected hub connections together by generating an additional circular layout at the left bottom of the interface. It utilizes the FDEB bundling technique [19] within a super-hub circular layout for all hub nodes in the main view. It also represents the ensemble nodes with matching color schemes corresponding to the hubs in the main view. The order of the node groups in the ensemble hub view is determined by the generating order of the corresponding hubs in the main view, and the nodes inside each group are ordered by their IDs in the original dataset.

Similar to the hub stream view, it can update itself automatically without any further direction. Figure 3.33 shows the demonstration of an ensemble hub view corresponding to 5 different hubs in the main view. We can see from Figure 3.33 (b) that some higher level edge flow trends between different hubs are highlighted by the force-directed edge bundling technique.

The ensemble hub view serves as a complementary visualization for the hub model in the main view. It addresses the limitation of not clearly visualizing all hub connections on the main graph by providing a more focused layout of the node and edges in another separate view. It also employs both the advantages of linear and force-directed edge bundling techniques for visualizing connections in a circular layout. These complementary views of the same data are intended to operate together.



(a) (b)

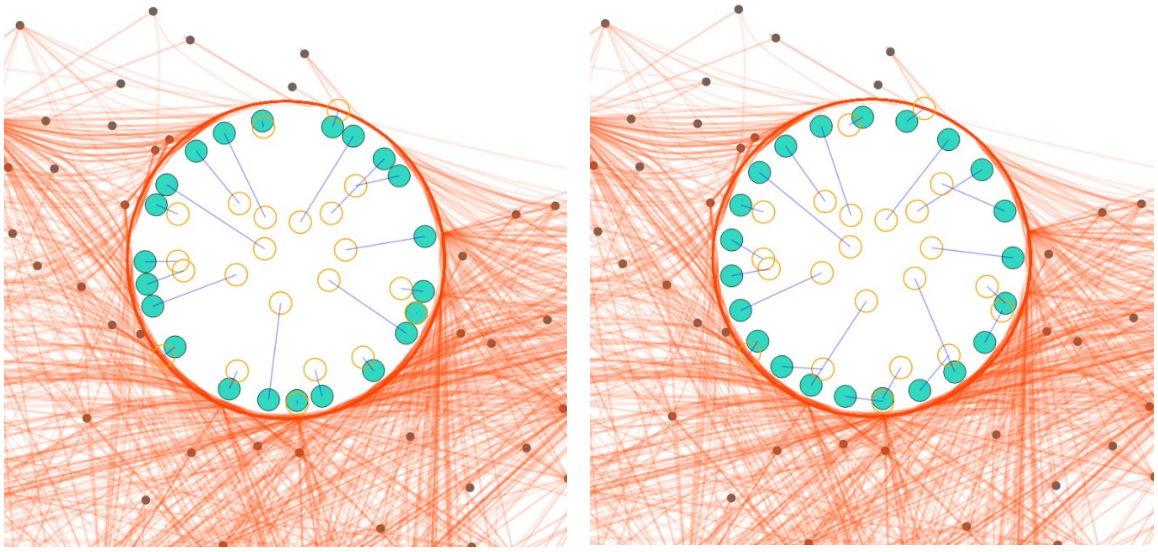
Figure 3.33: A demonstration of the ensemble hub view. (a): linear bundle. (b) force-directed bundle.

## 3.5 Other Features

Our model provides several other features to assist the exploration of the data. We summarize these minor features in this section.

### 3.5.1 Visual Glyphs

We proposed our technique for reducing visual clutter while generating sub layouts in geo-spatial connected graphs in Section 3.2.1. However, the trade-off of the approach is that we may lose the precise location values of the nodes temporarily by generating the circular layouts and repositioning the nodes. To mitigate this issue, we introduce an optional feature for recalling the original spatial information of the related nodes by visual glyphs. The visual glyphs can be switched on and off in the control panel, and the inner visualizations are hidden while the visual glyphs are displayed. Figure 3.34 shows the demonstration of this visual glyph feature for both radial layout and uniform layout.



(a) (b)

Figure 3.34: Visual glyph demonstration. (a): radial layout glyphs. (b): uniform layout glyphs.

By optionally presenting visual glyphs after generating the sub layouts, we can provide clue about exactly where the nodes come from in the original graph. As illustrated in Section 3.2.1, we introduce an animation process for how nodes are mapped to the circular

layout. The visual glyphs also coordinate with the node moving animation in the dynamic process to strengthen the geo-spatial features. Specifically, the animation process is a dynamic approach while the visual glyph technique is a static alternative. Both approaches can alleviate the geo-spatial information loss brought about by the dynamic hub technique.

### 3.5.2 Detail View

The detail view can be found at the right bottom area of the interface as illustrated in Figure 3.1. It provides a list of raw data entries from the original data set to show the detailed information of all the edges that are selected in current hubs. For example, the source and destination airport names and exact value of passengers from a specific airline route in an aero network. By introducing the detailed view, the model is more precise for specific queries at a very detailed level.

### 3.5.3 Cloud View

The cloud view is located to the left of the main view, as illustrated by Figure 3.1. It maintains two tag clouds for different data focuses that reflect the contents shown in the main view. Taking aero traffic network as an example, one cloud visualizes tags of the nodes, namely airport or city names; the other cloud describes categorical information for the current selection. The cloud view gives a fast impact about what items are being visualized in the main view.

# Chapter 4

# Evaluation: Complexity and Quantitative Analysis

Due to the complexity and flexibility of modern visualization approaches, performance issues in geo-spatial network visualization systems need to be closely considered. In this chapter, we present performance discussions about several key algorithms incorporated in our approach.

We first provide the analysis of several main algorithms in our model regarding complexity theory. Second, we present a series of accurate quantitative measurements in a controlled experimental environment for different algorithms using three real data sets with different amount of data, respectively. By controlled experiment we mean that we test a single algorithm's actual run time under the same conditions, except for the ground dataset. All of our experiments are conducted on the same Windows PC with Intel Core i7-4770k CPU at clock rate 3.5GHz, 12 GB memory, using Google Chrome as the test environment platform.

## 4.1 Computational Complexity Analysis

Our approach incorporates several main algorithms that may involve iterative access of the nodes and edges, such as node positioning, edge bending and bundling, and another algorithms that require concentrated geometry or mathematical computations, such as the generation of a control point array and the subsequent interpolation process. We summarize all of the key algorithms and the corresponding computational complexity analysis results in Table 4.1. We need to point out that we use the worst-case complexity for all algorithms listed in Table 4.1 for analysis purposes, however the real cases may be much better in practice than the worst-cases analyzed in Table 4.1. The analysis results for related algorithms in Table 4.1 are discussed as follows.

The radial node positioning algorithm positions a selection of nodes as a circular layout by radial mapping, and is introduced in Section 3.2.2 in Chapter 3. It iteratively accesses the nodes in the selection and maps them to the circumference according to the conflict

status with previous nodes. Since there is a double iteration of the nodes, and in the outer iteration we merge the intervals by double iteration as well, the computational complexity is $O(n^3)$, where $n$ is the number of nodes in the selection.

Table 4.1: Complexity analysis for main algorithms.

| **Algorithms** | **Computational Complexity** |
|---|---|
| Radial node positioning | $O(n^3)$, $n$ is number of nodes in the circle |
| Uniform node positioning | $O(knlog(n))$, $n$ is number of nodes, $m$ is number of simulation times [79] |
| Force-directed edge bundling | $O(n^2mk)$, $n$ is number of edges, $m$ is number of subdivision points, $k$ is iterations [19] |
| Control point generation | $O(n)$, $n$ is number of related edges |
| Control point interpolation | $O(nm)$, $n$ is number of control points, $m$ is number of edges [91] |

The uniform node positioning is described in detail in Section 3.2.2, and it locates a selection of nodes which are previously mapped by the radial node positioning algorithm evenly on a circumference. The D3 Engine uses Barnes–Hut approximation [81] to simulate the physical repulsion process of related nodes, and Barnes–Hut approximation is applied on each simulation step with a time complexity of $O(nlog(n))$ [81], where $n$ is the number of nodes to be calculated.

The combined node positioning algorithms discussed have significant complexity, but we need to point out that this complexity is applied on a sub selection of nodes in the main graph, therefore the data volume is not expected to scale excessively in this context. Further quantitative analysis also demonstrates that the algorithms achieves acceptable performance on real datasets.

The force-directed edge bundling technique is used in our model to facilitate the visualization of edge trend patterns, as introduced in Section 3.2.3. The original paper [19] states that the complexity of the algorithm is $O(n^2mk)$, where $n$ is the number of edges, $m$ is the number of subdivision points for each edge, and $k$ is the number of iterations executed. Force-directed edge bundling algorithm has a considerably high complexity, but that complexity is already optimized due to the complexity of the bundling task [19], and it is demonstrated as acceptable in large scale edge bundling situations in the original paper. Similar to the node positioning algorithms, we clarify that we apply this algorithm to the

inner edges of the selection, which means that the data volume shall also not scale excessively in this context.

The control point generation algorithm utilizes 15 control points around the circle to smooth the edges going through the circular layout as well as maintaining the edge trends around it, as discussed in Section 3.2.3. We identify the computational complexity of generating control points for one single edge that passes through a circular layout as $O(1)$, since we use a fixed number of control points. For a single circular layout, the time complexity should be $O(n)$, where $n$ is the number of edges that interact with the circular layout.

Interpolation of the control points is another important algorithm to consider. Although we leverage the build-in implementation in the D3 Engine, we still analyze the complexity of interpolating a cardinal spline as $O(n)$, where $n$ is the number of control points used [91]. For a single circular layout, the time complexity should be $O(nm)$, where $n$ is the number of control points, and $m$ is the number of edges that interact with the circular layout. However, the interpolating calculation for each control point may consume significant time by performing mathematical calculations such as the computing of second derivatives for control points. Therefore, we also provide the quantitative analysis results for the built-in function for generating cardinal splines in D3 Engine in the following section.

We discussed the computational complexity for several main algorithms involved in our approach in this section. However, the result discussed above only considers the complexity and scalability of our model from the perspective of algorithm analysis in theory. In fact, a considerable amount of geometry computations and numerical calculations is also executed in real time, which may also contribute to the degradation of performance. We therefore also conduct the quantitative evaluation on three real datasets to demonstrate the practical performance of our approach.

## 4.2 Quantitative Performance Evaluation

In this section we use real data sets with different data volumes to test our model's performance and scalability in practical applications. We designed controlled experiments

to accurately record the execution times for the algorithms listed in Table 4.1 on different datasets respectively. The information about the 3 data sets are described in Table 4.2.

Table 4.2: Summary statistics of 3 data sets.

| **Datasets** | **Number of nodes** | **Number of edges** |
|---|---|---|
| GitHub collaboration graph | 164 | 639 |
| US aero network | 235 | 2101 |
| US migration graph | 6517 | 9780 |

We provide a brief description of the data sets as follows. The GitHub collaboration graph is a node-link graph, and it is a subset of the database we use for a case study in Chapter 6, and it includes 35 hours of collaboration event data among open source developers in 164 cities and 629 communication links. This dataset is generated by extracting a part from the real dataset for experimental purposes. The US aero network data includes 2,101 airline routes between 235 airports in United States, and it is also demonstrated as another case study in our research. The US migration graph involves migration patterns of 6,516 individuals in United States. Compared to the small and medium sized data sets discussed previously, the data set is often used by researchers for scalability analysis purposes due to the large scale of nodes and links.

The design of experiments is described as follows. We generate 5,000 random test cases for each algorithm on each dataset, and compute average running times based on a statistical approach. The test case simulates the process of generating a random circular layout in a fixed visualization area. We define the visualization area as a rectangle with width 1,000px and height 600px, and the data are scaled to fit to the visualization area. We then randomly select circle centers within the visualization area, for each circle center, we generate a random radius in the open interval between 0px and 150px, which is a typical range for the generated circular layouts in our approach. We then execute the process for each algorithm and record the elapsed time for each test case, and compute the average value of the elapsed times. The simulation results are provided in Table 4.3, and the measurement unit is milliseconds. Note that all of the results are averaged values for generating a random circular layout 5,000 times.

Table 4.3: Quantitative analysis results for main algorithms on different datasets.

| | **GitHub** | **US Aero** | **US Migration** |
|---|---|---|---|
| Radial node positioning | 0.016ms | 0.035ms | 0.183ms |
| Uniformed node positioning | 0.565ms | 0.728ms | 0.959ms |
| Force-directed edge bundling | 2.649ms | 3.081ms | 181.468ms |
| Control point generation | 0.131ms | 0.313ms | 0.704ms |
| Control point interpolation | 1.298ms | 3.027ms | 6.735ms |

*Node Positioning Analysis*

We can see that the radial node positioning is within an acceptable performance range costing the least time among the main algorithms, although the time complexity of the positioning algorithm was previously analyzed as $O(n^3)$. That is because that the algorithm involves less numerical calculation than other algorithms and only a limited number of nodes in practice, and the process is straightforward with several trigonometry calculations.

In terms of the uniform node positioning algorithm, the default force simulation algorithm provided by D3 Engine employs Barnes–Hut approximation [81] and uses a cooling parameter *alpha*, which controls the layout temperature. The cooling parameter is updated in each iteration step of the physical simulation to imitate the natural cooling process of the atmosphere. As the simulation converges on a stable layout, the temperature drops, causing nodes to move more slowly. Eventually, *alpha* drops below a threshold, namely 0.05, and the simulation stops completely [92]. However, the limitation of this implementation is that the algorithm converges too slowly, and this is because D3 Engine considers the force based simulation as an asynchronous process and keeps updating the layout even if the nodes are almost in position. This approach has the advantage of utilizing the asynchronous feature of the scripting language in the browser to achieve the most accurate simulation results, but it is not suitable for our case, as our visualization approach requires immediate results in a synchronous way. We state the testing results simulated by controlling *alpha*, which is the automatic cooling parameter defined in the D3 Engine as Table 4.4.

Table 4.4: Convergence times for original cooling parameter control on different datasets.

| | **GitHub** | **US Aero** | **US Migration** |
|---|---|---|---|
| Cooling parameter control | 7149.197ms | 9434.939ms | 30937.418ms |

From the results in Table 4.4 we note that the original algorithm is unsuitable in a real time visualization system, as it takes at least 7 seconds to generate the uniformed layout for a circular selection. We could improve the algorithm by adjusting the parameters, but the parameters are not specially designed for our case and it is difficult to define the relevance between the cooling parameter and real cases. We therefore design a custom convergence criteria for specific use in our approach. The convergence condition should be triggered when nodes are almost positioned at equal distance from each other. We then define the mathematical expectation of the distances between every node in the simulation process as *E*, which is defined by the formula below, where n is the number of nodes on the circular layout.

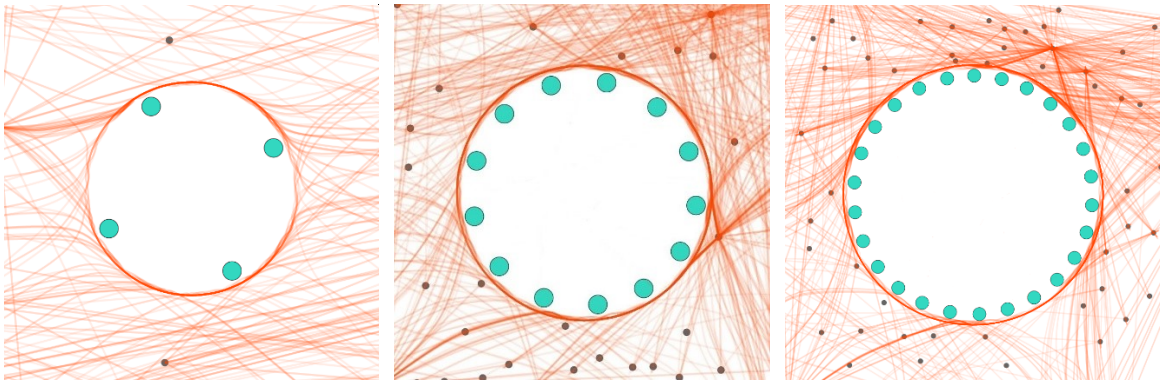
$$E = 2\pi / n \tag{4.1}$$

The actual meaning of the expectation *E* is the expected center angle between each pair of two neighboring nodes on the circular layout. We then perform condition tests in each iteration of the simulation process, and stop the simulation process if the condition in the formula below is satisfied, where *variance* is the stop condition, $angle_i$ is the angle between the node with ID *i* and the node with ID *i-1*, *E* is the expectation defined in Equation (4.1), and *n* is the number of nodes on the circular layout.

$$variance = \sum_i |angle_i - E| / n < 0.05 * E \tag{4.2}$$

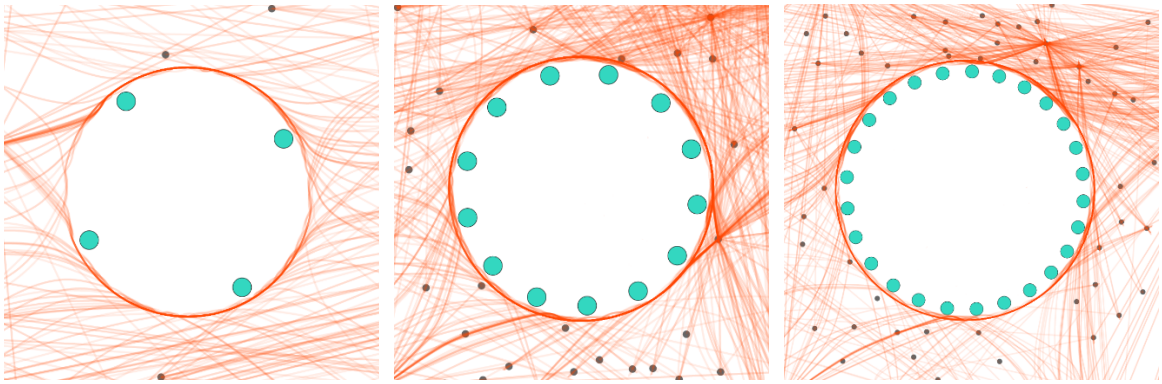
The condition test ensures that all nodes in the simulation process are eventually positioned no further than 5% of the expected distance between the neighboring nodes. The quantitative analysis results for different datasets are demonstrated in Table 4.3, and we draw the conclusion according to Table 4.3 that the uniformed node positioning algorithm in our model scales well when applied to large datasets, and we can obtain the layout within

1ms on average on the migration dataset. We also compare our simulation results under the same condition with the convergence criteria defined in D3 Engine, and the results are shown as Figure 4.1 and Figure 4.2.



(a) (b) (c)

Figure 4.1: The uniformed node layout simulation results with original cooling parameter in D3 Engine. (a): Small number of nodes. (b): Medium number of nodes. (c): Large number of nodes.



(a) (b) (c)

Figure 4.2: The uniformed node layout simulation results with improved convergence condition in our approach. (a): Small number of nodes. (b): Medium number of nodes. (c): Large number of nodes.

From the comparison between Figure 4.1 (a) and Figure 4.2 (a) we can see that there are slightly difference between the two approaches, and the 4 nodes simulated by our result, as illustrated in Figure 4.2 (a), are not as evenly positioned as the result simulated by the original cooling parameter in D3 Engine, as shown in Figure 4.1 (a). The other two comparison groups are simulated with different number of nodes, as illustrated by Figure 4.1 (b) - Figure 4.2 (b), and Figure 4.1 (c) - Figure 4.2 (c), respectively. We note that no obvious differences can be identified by the naked eye from the comparison of the two groups of image results.

By comparison between two different convergence approaches, we admit that our improved convergence condition generates slightly worse visual effects than the approach with automatic cooling parameter control in D3 Engine. However, our improved approach significantly reduces the actual execution time of the algorithm, and enables the force based approach to be utilized in large scale dynamic visualization systems, by trading off a subtle degree of visual effect.

*Force-directed Bundling Analysis*

The result of force-directed edge bundling in Table 4.3 accords with the computational complexity provide in Table 4.1. We need to clarify that the parameters have been fixed to specific optimized values in our experiments. The optimized values are found through an iterative refinement scheme by the authors of the original paper to improve performance [19]. For example, the spring constant $K$ is set to 0.1, which controls the amount of bundling, and the number of iterations $I=60$ for simulating force interactions. The parameters are demonstrated by the authors as sufficient to guarantee smooth edges at an acceptable performance [19].

According to the simulation results in Table 4.3, the average running time for edge bundling job in one hub is 181.468ms when applied to a large data set with more than 9,000 edges. We can therefore confirm that the algorithm is suitable for edge bundling in our model. However, the complexity of this bundling technique is much higher compared to other algorithms in our approach possibly causing a bottleneck of the performance, and in fact it may take enough time to cause a perceivable delay when applied to a large circular layout for the migration dataset. We therefore only utilize the force-directed bundling to the inner edges of the circular layouts, and we do not ensure the real-time updating of the edge bundles during the dynamic placement process as well.

*Control Point Generation and Interpolation Analysis*

The control point generation algorithm is tested for multiple edges related to one circular layout. From Table 4.3 we can see that it costs 0.131ms, 0.313ms, and 0.704ms to compute control points for all edges relate to a circular layout. The algorithm is thus acceptable

regarding performance. The complexity of control point interpolation process is the same as the generation process, but it involves more complicated numerical computations in each iteration. From the simulation result in Table 4.3 we can draw the conclusion that the interpolation runs in an acceptable range of running times on a graph with very high density.

# Chapter 5

# Case Study: US Aero Network

In this chapter we provide a detailed investigation into the visualization of transportation related data facilitated by our model. We firstly introduce the motivation for choosing US aero data set. Then, a detailed description of the raw data and the integration and processing of the data are given. Finally, we show how our model can assist the geo-spatial data analysis when applying the data set to our visualization model.

## 5.1 Motivation

The transportation data visualization is a large and important category of geo-visualization systems. It describes terminals or path information from various types of vehicles, such as airplanes, vessels, and automobiles. The importance of visualizing such information is increasing since there are many real and potential applications that can be demonstrated by these kind of data [32, 61]. Therefore, we believe that the ability to interpret transportation related data is the most important consideration in the evaluation of our visualization system in order to ensure a broad applicability.

Within this context, we focus on the analysis of airline route connections in the United States, because the US domestic airline network is one of the busiest transportation systems around the world. The air traffic data is therefore appropriate for testing visual clutter reduction approaches, since there are many cluttered areas with a high density level of nodes and edges. This matches our initial motivation very well as mentioned in Chapter 1. We integrated information from two real-world data sets: OpenFlights.org [74] and the RITA Databank of US government [77] to collect enough raw data. These two data sources provide quite complete, accurate and credible data entries regarding the airline routes in United States.

## 5.2 Data Preparation and Processing

The raw data is obtained directly from the RITA Databank [77]. The RITA Databank records available transportation data regarding various aspects of scheduled airline traffic, and it maintains a set of databases for each category. Among all databases the T-100 database describes the US domestic and international airline market and segment data, and certified US air carriers have been reporting monthly air carrier traffic information since 1990. The database can be widely used in various data mining and GIS systems due to the very low data missing rate and continuous record during the time.

We use Python for all of the data integration and processing tasks throughout our study, and the data processing results are stored in JSON format to achieve maximum compatibility and performance with the JavaScript libraries we employ.

The raw data for defining geo-spatial features is in XML format and it includes location information of 235 primary airports in the United States and 2,101 air route connections between them. Specifically, the airports are represented by their 3 letters IATA codes and their node identifiers in the data set, and the edges are represented by source and target node identifiers.

The time series data about passenger flows, and detailed information about the airports, such as city names, airline route distances, and category information such as carrier names and their business share information, are extracted from the T-100 database. We extract around 256,000 data entries related to passengers, time, and carriers in the period between January and December in the year of 2014. The raw data is stored in comma separated values format and each data entry in the database represent a particular directional airline route. Some sample data from the T-100 database is listed in Table 5.1.

The raw directional data are then matched to the previous undirected database extracted from the XML data source to build a Json target. We extract and integrate the raw data into a tree structure with each single airline as the root node. The first level represents the direction of the edges. In the second level, the carrier share node and data node both includes a list of 12 time series data items, where the data represents passenger flows over the time period and the carrier share represents the corresponding business share of all related domestic carriers on this route. The child nodes of the carrier_share node consists

of a set of dictionary items, and each dictionary item maintains a list of carrier and value pairs that keep track of the business shares for that particular directional airline.

Table 5.1: Sample data entries from RITA dataset showing passengers, distances, unique carrier codes, carrier names, origins, origin names, destinations, destination names and current months.

| PASSENGERS | DISTANCE | CARRIER_CODE | CARRIER_NAME | ORIGIN | ORIGIN_NAME | DEST | DEST _NAME | MONTH |
|---|---|---|---|---|---|---|---|---|
| 79778 | 1235 | AA | American Airlines Inc. | DFW | Dallas/Fort Worth, TX | LAX | Los Angeles, CA | 5 |
| 79923 | 1235 | AA | American Airlines Inc. | LAX | Los Angeles, CA | DFW | Dallas/Fort Worth, TX | 6 |
| 80051 | 404 | DL | Delta Air Lines Inc. | ATL | Atlanta, GA | MCO | Orlando, FL | 11 |
| 80151 | 1235 | AA | American Airlines Inc. | DFW | Dallas/Fort Worth, TX | LAX | Los Angeles, CA | 6 |
| 80457 | 404 | DL | Delta Air Lines Inc. | MCO | Orlando, FL | ATL | Atlanta, GA | 7 |
| 80482 | 404 | DL | Delta Air Lines Inc. | MCO | Orlando, FL | ATL | Atlanta, GA | 6 |
| 80570 | 1235 | AA | American Airlines Inc. | LAX | Los Angeles, CA | DFW | Dallas/Fort Worth, TX | 3 |
| 81289 | 1448 | AS | Alaska Airlines Inc. | SEA | Seattle, WA | ANC | Anchorage, AK | 7 |
| 81299 | 404 | DL | Delta Air Lines Inc. | ATL | Atlanta, GA | MCO | Orlando, FL | 5 |
| 81457 | 101 | HA | Hawaiian Airlines Inc. | OGG | Kahului, HI | HNL | Honolulu, HI | 1 |
| 81699 | 1448 | AS | Alaska Airlines Inc. | ANC | Anchorage, AK | SEA | Seattle, WA | 8 |

We also calculate and record the average passenger flow value during the full time period for each airline route to support the peapod filter feature. A sample output data entry is described as the format below.

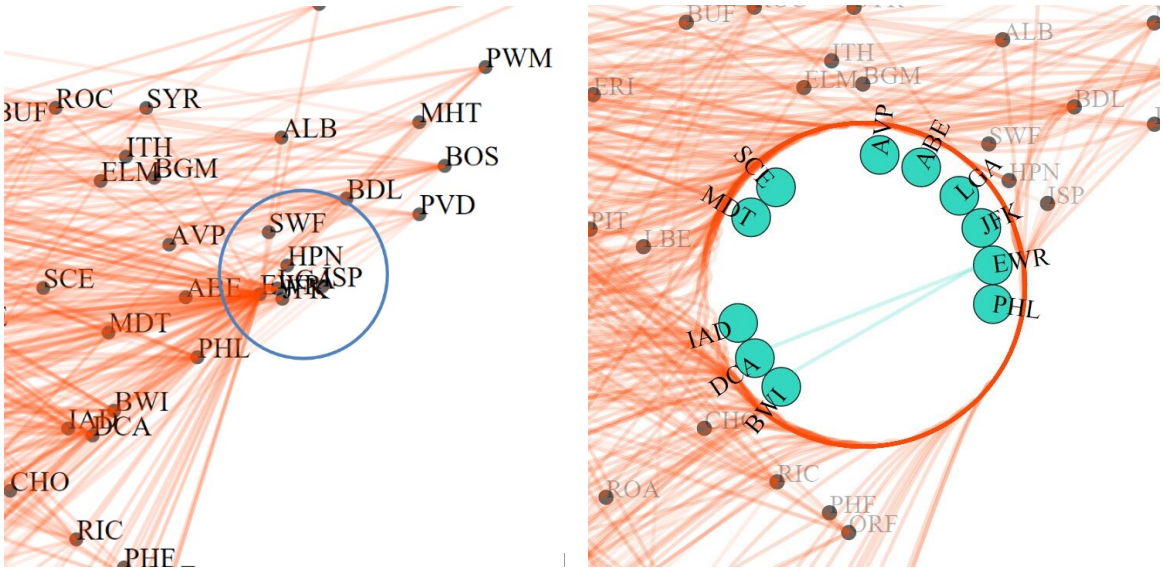
```
{
 "go":
       {
        "origin": "DTW",
        "dist": "705.00",
        "origin_name": "Detroit, MI",
        "dest": "LIT",
        "carrier_share": [{"WN": 45, "EV": 1037} … {"EV": 1236, "9E": 67}],
        "dest_name": "Little Rock, AR",
        "data": [1082, 910 … 1303]
       },
 "come":
       {
        "origin": "LIT",
        "dist": "705.00",
        "origin_name": "Little Rock, AR",
        "dest": "DTW",
        "carrier_share": [{"EV": 1028} … {"EV": 1307, "9E": 73, "OH": 1}],
        "dest_name": "Detroit, MI",
        "data": [1028, 1118 … 1381]
       }
}
```

## 5.3 Results and Discussion

Given the varying needs and knowledge level requirements in geo-spatial transportation data visualization, our solution provides various ways to explore and analyze the data in an integrated interface. We describe several representative scenarios that deal with many of the activities that may be performed in a typical related analysis process in the following sections.

### 5.3.1 Scenario 1 – Presenting Local Routes by Circular Layouts

We first demonstrate our model's ability to reduce the visual clutter while revealing the internal connections in local areas in a large graph. The local area, as illustrated in Figure 5.1 (a), is the New York Metropolitan Area surrounded by several main air traffic terminals that have a large volume of passenger flows in the eastern part of United States. We label each city in this area with their IATA codes provided by US government.




(a) (b)

Figure 5.1: Local airline routes in New York Metropolitan Area. (a): Several main airports in the metropolitan area that are overlapped seriously. (b): Initial selection result.

From Figure 5.1 (a) we can find some airport codes in main cities easily such as BOS (Boston Logan International Airport), PHL (Philadelphia International Airport) and DCA

(Reagan National Airport). However, the two large and crucial airports in the New York Metropolitan Area, namely: JFK (Kennedy International Airport) and LGA (La Guardia Airport) are highly overlapped by the surrounding edge cities, as illustrated by a blue circle. The issue has arisen that we cannot distinguish among the surrounding cities by their text labels, let alone the connections between them.

Figure 5.1 (b) shows the initial selection result of the same area with Figure 5.1 (a). We note that the airport nodes are arranged radially to match a circular layout, and every airport node in that area can be observed easily. The circular layout preserves a considerable extent of outer edge patterns going around the area as well.

Moreover, from Figure 5.1 (a) we can only tell that there are numerous airline routes going out of or coming into the New York Metropolitan Area, however we gain no knowledge about the internal airline routes in that area due to the narrow space. We note that our initial selection result shows clearly that there are connections between EWR (Newark International Airport), DCA and EWR, IAD (Dulles International Airport) node pairs. Therefore, we can also draw a conclusion according to the initial selection result that there are few airline routes among the selected airports in the current data set, and JFK and LGA in New York has no short-haul routes with other airports in the metropolitan area.

Another typical case is shown in Figure 5.2 with uniform node layout and dynamic placement. LAX (Los Angeles World Airports) is a key terminal in southwestern United States, and the surrounding airport structure is illustrated as Figure 5.2 (a). The situation is similar to the New York Metropolitan in Los Angeles, since we have a cluster that includes 8 satellite cities around it. Figure 5.2 (b) shows the result with glyph cues after the initial selection. We can see that the 8 satellite cities are positioned evenly on the generated circular layout, and we can identify the original locations of the cities according to the visual glyphs easily because the visual glyphs are much less cluttered than the original layout with labels.

Figure 5.2 (c) shows the linear bundle result of the selection. We recognize from the linear bundle that LAX has two airline routes to SBA (Santa Barbara Municipal Airport) and SAN (San Diego International Airport). We then perform a dynamic placement operation to the current selection by moving it to the northwest slightly to include SBP (San Luis Obispo County Regional Airport) into the selection, as illustrated by a blue

rectangle. As a consequence, as shown in Figure 5.2 (d), SBP has only one connection to LAX and has no other relations with the rest of the current selection.

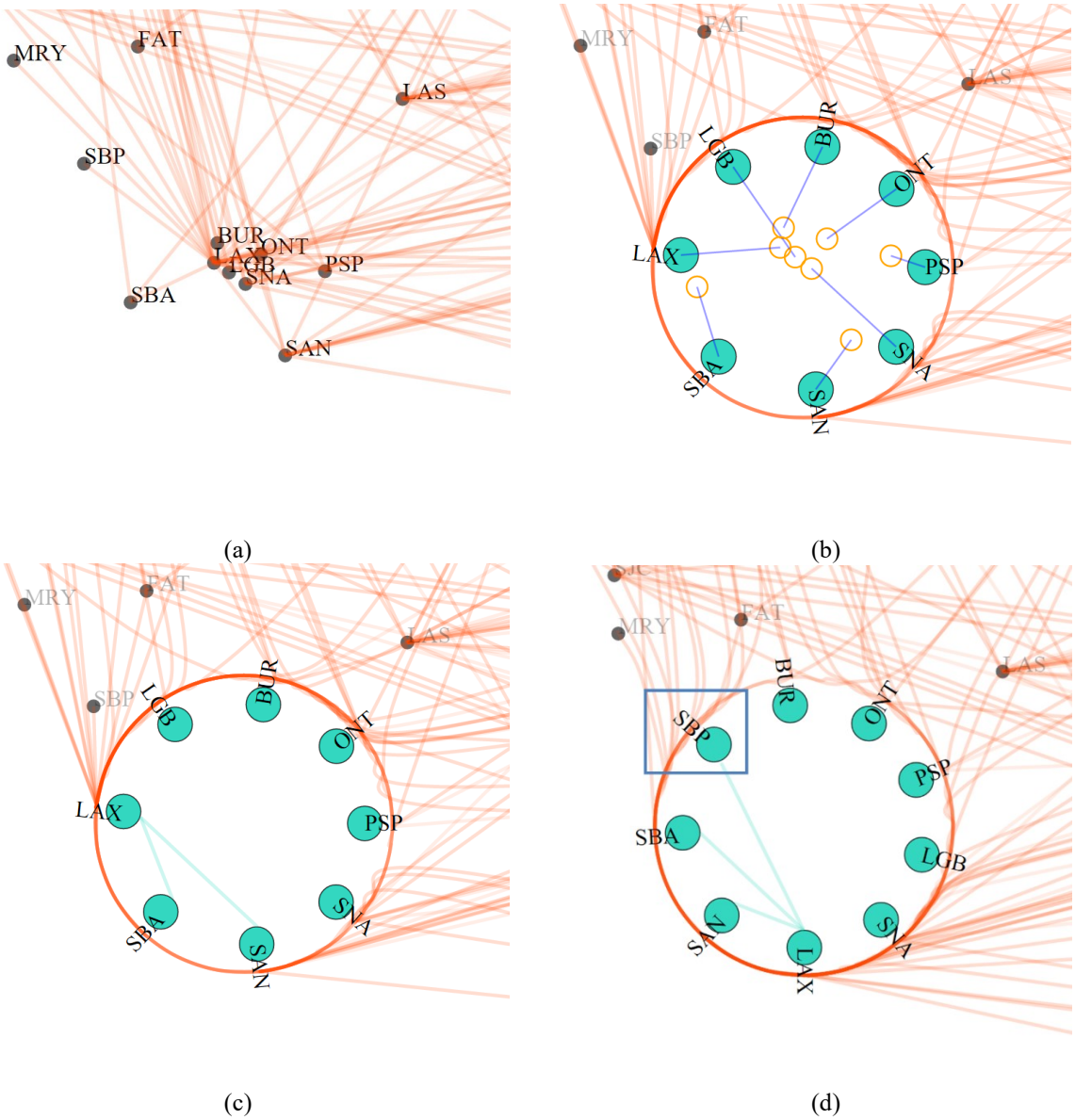
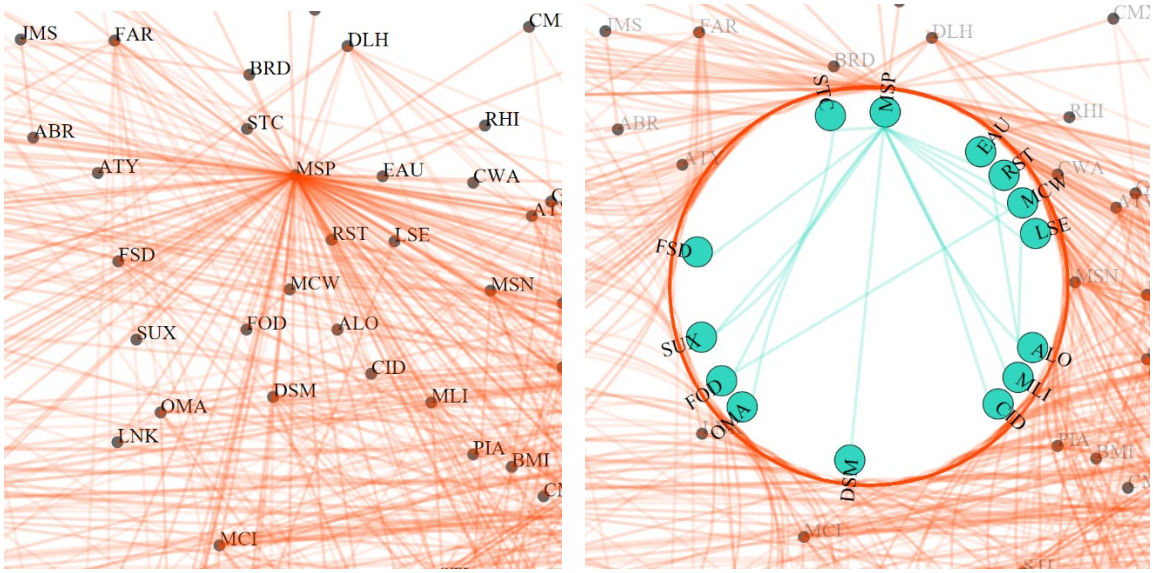



Figure 5.2: Local airline routes in Los Angeles Metropolitan Area. (a): Initial airport structure of the area. (b): Visual glyphs after the initial selection. (c): Linear bundle result of the current selection. (d): Performing dynamic placement operation to include node SBP by adjusting the position of the circular layout.

We also examine our model's ability to present local area edge trends in a selection with a larger number of edges by the force-directed edge bundling technique. Figure 5.3 shows a typical force edge bundling result applied to the Minneapolis surrounding area. MSP (Saint Paul International Airport) is a regional center in Minnesota State, as shown in Figure

5.3 (a). We do a selection of 14 nodes in the local area and apply the force-directed edge bundling to the circular layout, the result is illustrated as Figure 5.3 (b).




(a) (b)

Figure 5.3: Local airline routes around Minneapolis. (a): Initial airport structure of the area. (b): Force-directed edge bundle result applied on Figure 5.3 (a).

From the bundling result in Figure 5.3 (b) we can see clearly two edge trend patterns sourced from MSP node, namely: the southwest city group including OMA, FOD and DSM, and the southeast city group including ALO, MLI and CID. The edge trend pattern in Figure 5.3 (b) demonstrates that MSP has a number of airline connections to both southwest and southeast directions.

### 5.3.2 Scenario 2 – Presenting Passenger Flows by Peapod Model

In this scenario, we will investigate the ability of our approach to visualize time series flow data. Houston is a chief city in southern United States, and IAH (George Bush Intercontinental Airport) is the regional terminal for air traffic. Figure 5.4 (a) shows the result for choosing peapod bundle in the generated circular layout. We can see that 9 peapods are generated in the circular layout with 13 major cities in the region, which conveys an overall feeling of the passenger flows in this area.

We then perform a filter operation by sliding the range controls in the control panel, to filter out the peapods that transport the average passenger flows between 3,500 and 8,500 during the whole year, as illustrated by Figure 5.4 (b). From Figure 5.4 (b) we can see that 3 airline routes are filtered out, namely: IAH-DAL, IAH-SHV and IAH-LCH. The red dash in the peapod body indicates the source city of the data flow, for example, the red flow in path IAH-DAL starts from node IAH, and vice versa for the blue flow.

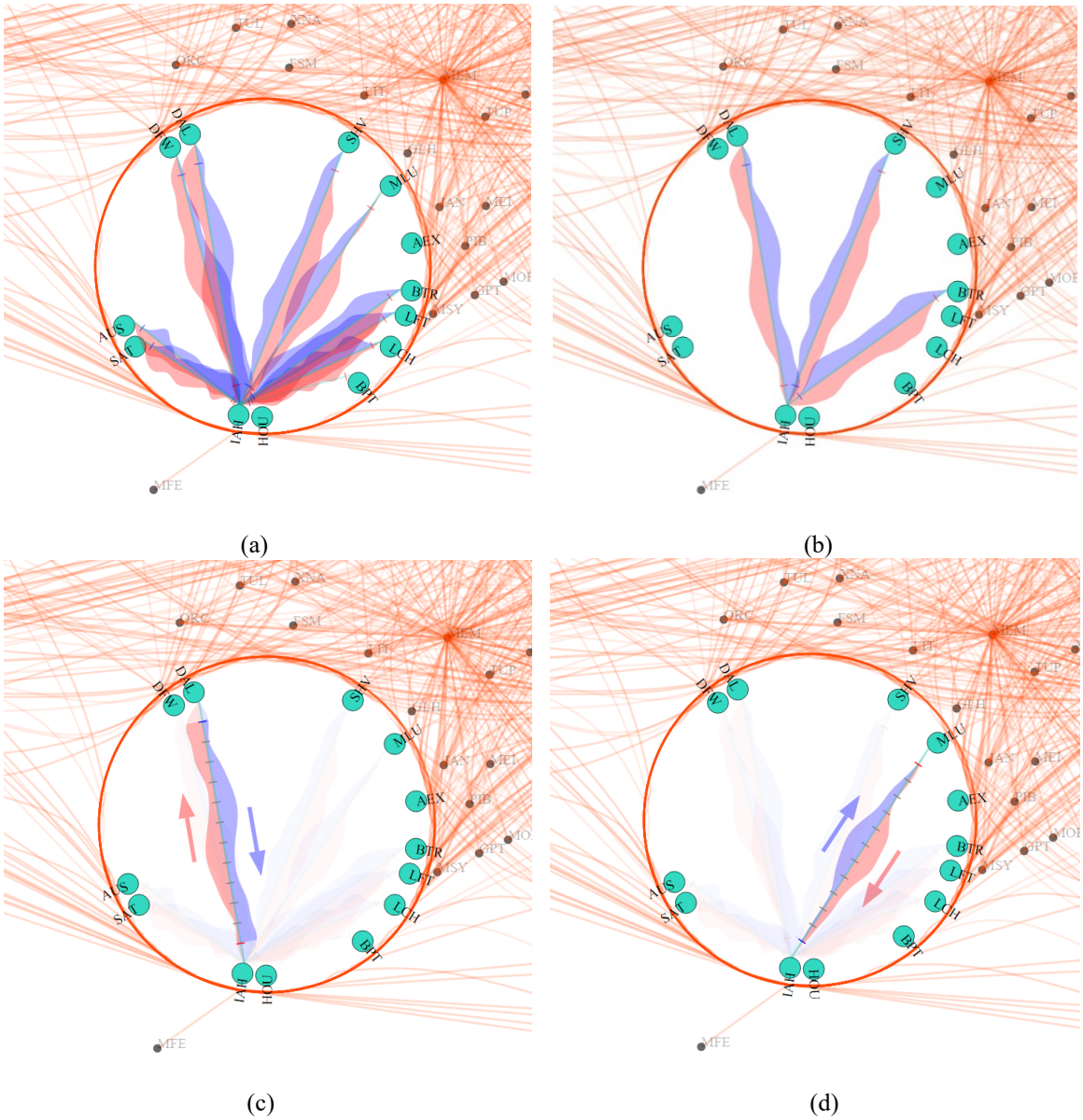
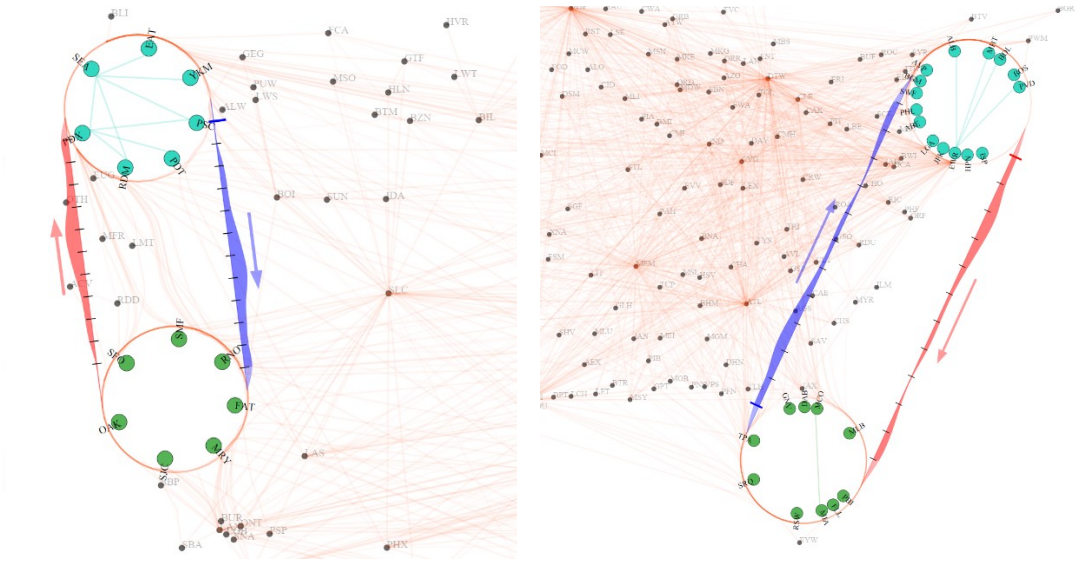



Figure 5.4: Passenger flow visualization in the neighborhood of Houston. (a): 9 peapod connections in the generated layout. (b): Peapods that are filtered out between the range 3000 and 7500 by the peapod slider. (c): A peapod between IAH and DAL is selected by mouse hover function. (d): A peapod between IAH and MLU is selected by the mouse hover function.

Figure 5.4 (c) and Figure 5.4 (d) are two representative peapod flows selected by the mouse hover function. When the mouse hovers on one peapod, the axis and directional cues are shown to facilitate identifying the intervals. There are 12 time series in each peapod to denote 12 months in the year 2014. Figure 5.4 (c) shows the time series flow from IAH to DAL (Dallas Love Field), and we can see that there are higher passenger traffic records during the time intervals from January to March, and from July to September in both directions, and this route has less passenger traffic during April, May, and the end of the year. In comparison, the airline route between IAH and MLU (Monroe Regional Airport) shown in Figure 5.4 (d) has a quite large number of passenger flows in summer, while a small number of passenger flows in winter.

### 5.3.3 Scenario 3 – Presenting Terminal Groups with Multiple Hubs

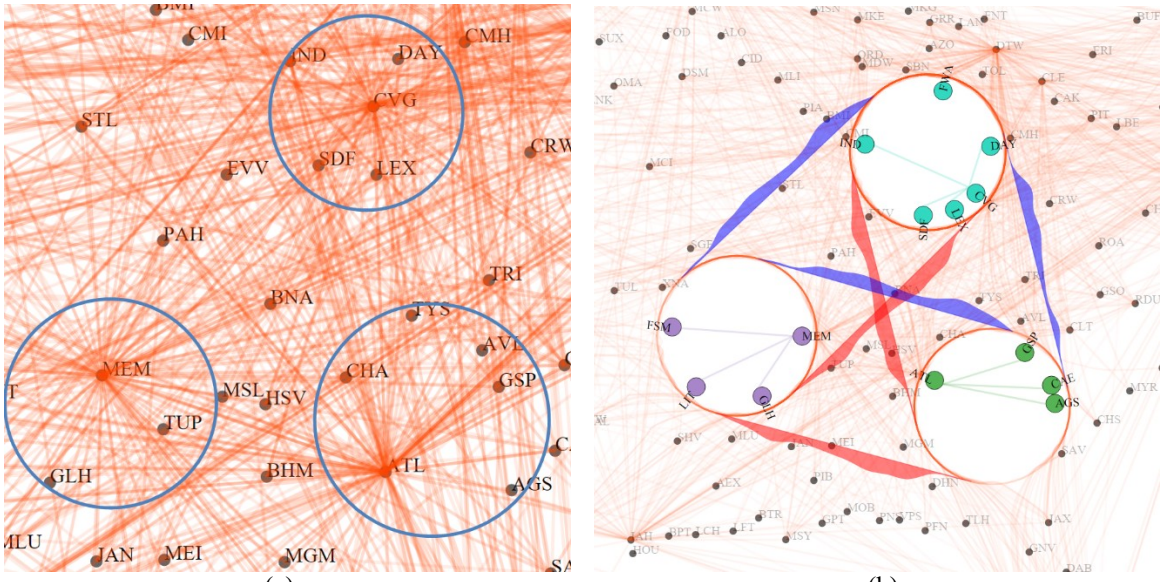
In this secenario we demonstrate our model's capability to manipulate multiple hubs in a large graph. We firstly consider the migration information between Seattle-Portland area and San Francisco Bay Area for example, as illustrated by Figure 5.5 (a).



(a) (b)

Figure 5.5: Passenger flow visualization between two hubs. (a): Hub view generated between Seattle-Portland region and San Francisco Bay Area. (b): Hub view generated between New York-Boston region and Florida.

We can see from the hub view in Figure 5.5 (a) that people seem to have a more active turnover rate during July to September. Another case is demonstrated by Figure 5.5 (b) showing aggregated passenger flows between New York – Boston region and Florida. We can observe a very obvious pattern from the flow shape that people in New York – Boston region tend to go to Florida during the time intervals from March to April, and from November to December, but they seldom travel to Florida in September. We can further infer that the high season to travel to Florida for vacation is in March, April, November and December.



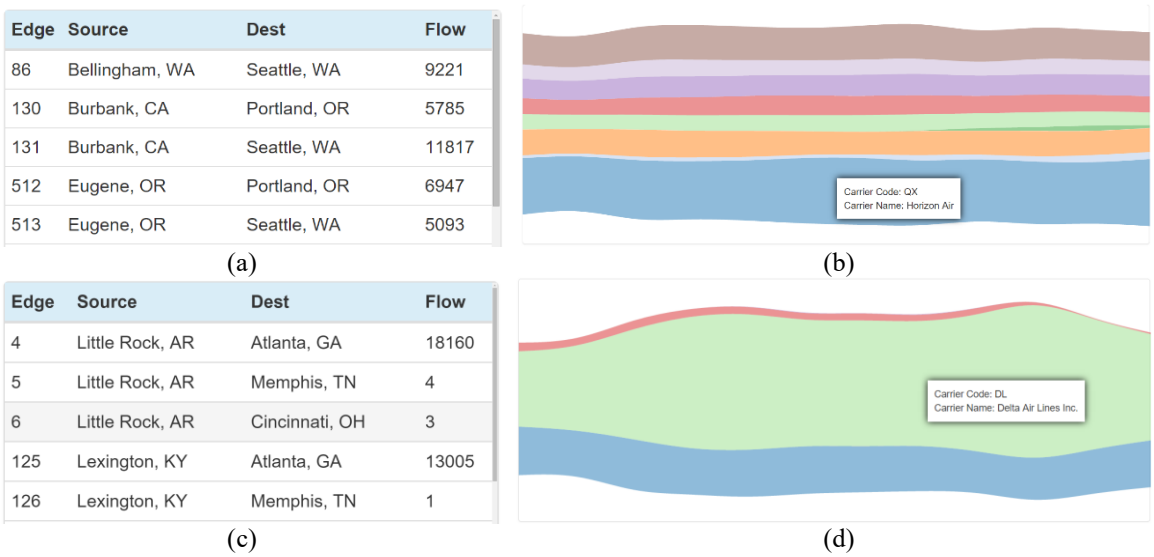

(a) (b)

Figure 5.6: Data flow between three groups of terminals: Cincinnati, Memphis and Atlanta. (a): Regional centers are highlighted by blue circles. (b): Hub view generated within the surrounding areas of the three main terminals.

The result of multiple relations is provided in Figure 5.6. CVG (Cincinnati/Northern Kentucky International Airport), MEM (Memphis International Airport) and ATL (Hartsfield–Jackson Atlanta International Airport) are three air terminals in the central United States, as described by Figure 5.6 (a) with blue circles. We build three hubs including the surrounding area of the 3 terminals and visualize them as Figure 5.6 (b). From Figure 5.6 (b) we see that the aggregate data flows among the three terminal groups are clearly reflected by the directional belts between each pair of hubs.

### 5.3.4 Scenario 4 – Presenting Aggregated Traffic Data by Various Views

Finally, we demonstrate the advantage of employing various additional views in our approach to better visualize the aggregated traffic information according to the current selection. Figure 5.7 (a) and Figure 5.7 (b) shows the corresponding detail view and hub stream view related to the result in Figure 5.5 (a). We can explore the detailed passenger flow values in Figure 5.7 (a) for every individual airline route in the current selections. From the result of the hub stream in Figure 5.7 (b), we can draw the conclusion that various airline companies have similar market shares in Seattle – South California routes, and owing to the mouse hover function, we can tell that Horizon Air occupies the largest market share among those airline companies.



| Edge | Source | Dest | Flow |
|---|---|---|---|
| 86 | Bellingham, WA | Seattle, WA | 9221 |
| 130 | Burbank, CA | Portland, OR | 5785 |
| 131 | Burbank, CA | Seattle, WA | 11817 |
| 512 | Eugene, OR | Portland, OR | 6947 |
| 513 | Eugene, OR | Seattle, WA | 5093 |

(a)



(b)

| Edge | Source | Dest | Flow |
|---|---|---|---|
| 4 | Little Rock, AR | Atlanta, GA | 18160 |
| 5 | Little Rock, AR | Memphis, TN | 4 |
| 6 | Little Rock, AR | Cincinnati, OH | 3 |
| 125 | Lexington, KY | Atlanta, GA | 13005 |
| 126 | Lexington, KY | Memphis, TN | 1 |

(c)

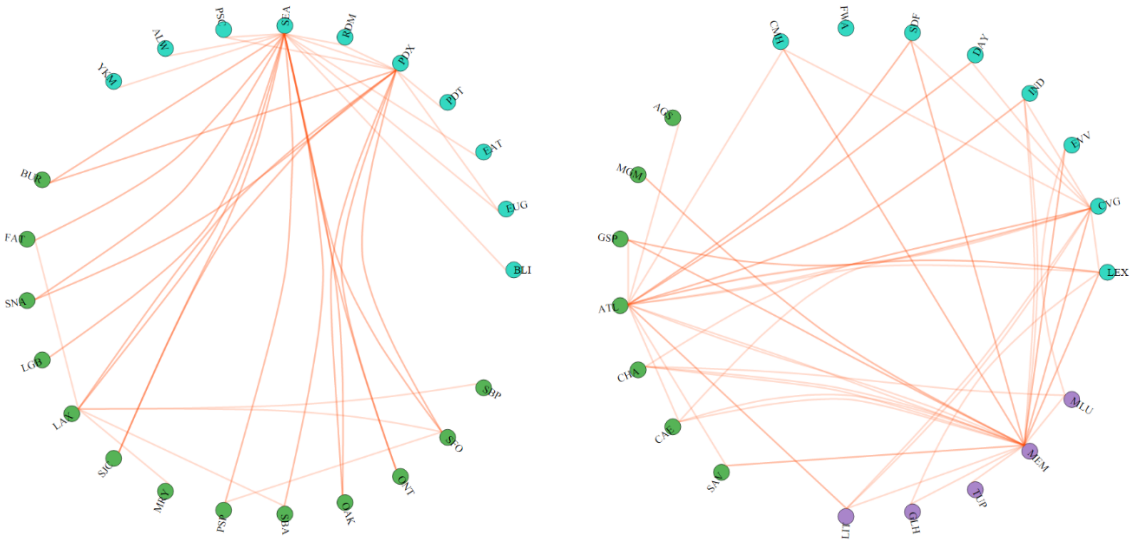


(d)

Figure 5.7: Detail view and hub stream view comparison between two different selections. (a): Part of the detail view corresponding to the selection in Figure 5.5 (a). (b): Hub stream view corresponding to the selection in Figure 5.5 (a). (c): Part of the detail view corresponding to the selection in Figure 5.6 (b). (d): Hub stream view corresponding to the selection in Figure 5.6 (b).

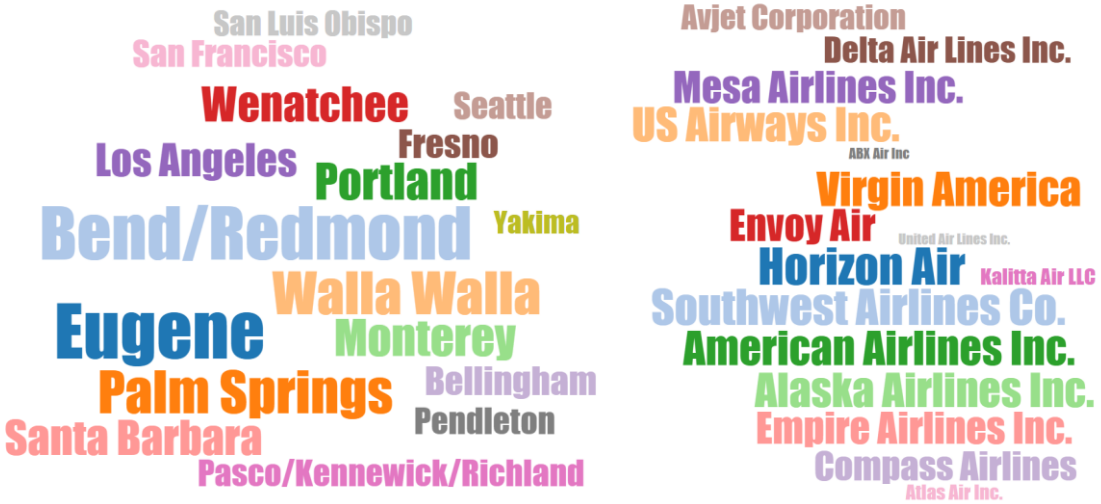
In comparison, Figure 5.7 (c) illustrates the detail view related to the selection in Figure 5.6 (b) for three different hubs, while Figure 5.7 (d) shows the hub stream view for the same selection. We can observe from Figure 5.7 (d) that the airline market around local areas near Cincinnati, Memphis and Atlanta is almost monopolized by Delta Air Lines Inc.

We also incorporate the ensemble hub view to help visualize direct connections across different airport hubs. Figure 5.8 (a) shows the corresponding ensemble hub to the selection in Figure 5.5 (a), while Figure 5.8 (b) provides the ensemble hub corresponding to the selection in Figure 5.6 (b).




(a) (b)

Figure 5.8: Ensemble hub view comparison between two different selections. (a): Ensemble hub view corresponding to the selection in Figure 5.5 (a). (b): Ensemble hub view corresponding to the selection in Figure 5.6 (b).




(a) (b)

Figure 5.9: Cloud view corresponding to the selection in Figure 5.5 (a). (a): Word cloud visualizing city names in the current selection. (b): Word cloud visualizing airline companies in the current selection.

From Figure 5.8 we note that the individual connections across groups of airports are shown clearly and the airline trend patterns are also reflected by the force-directed bundling technique. Therefore, the visualizations in the ensemble hub view and the main view are complementary to show the complete information regarding the current selections. We also offer the tag cloud technique to visualize additional tag information about the current selection to provide rapid information at a glance, as illustrated by Figure 5.9.

# Chapter 6

# Case Study: GitHub Collaboration Graph

In this chapter we present a detailed description for applying our model to the visualization of the collaboration and communication patterns in an open source software developer community. We begin by introducing the motivation of choosing the data set. Then a detailed description of the raw data and the processing of the data are given. Last but not least, we demonstrate our model's ability to visualize collaboration patterns in open source communities by several scenarios showing how our approach can facilitate the data analysis of geo-spatial social network data.

## 6.1 Motivation

Despite the traffic network and traffic flow analysis, social networks are another major topic involved in geo-spatial visualization research. A social network is defined by a collection of individuals and the social interactions among them, and it is generally visualized as a connected graph with individuals as nodes and relations as edges. Since the individuals in a social network often provide their location information in their profiles, location based data mining and visualization of geo-spatial social network studies is thus enabled.

We recognize the importance of social network analysis in geo-spatial systems, and employ the Github collaboration graph data set in this case study. Compared with the widely used general social network databases such as Twitter and Facebook that record people's opinions and interactions via text and photos, the Github community is a more professional collaboration network with a larger diversity of communication patterns. Specifically, software developers in Github community publish, manipulate, collaborate, and refer to each other's software code under the open source protocols with shared goals. The high level of data integrity and vast amounts of nodes and connections can support the demonstration of our approach, and the variety of collaboration patterns with accurate time series stamps in the data set enable us to analyze categorical data flows readily.

## 6.2 Data Preparation and Processing

We use Python scripts for all data acquisition, processing and integration, and the original data is obtained from the GitHub Archive Project [87], GitHub API Query Database [88], and Google Map Geocoding Query Database [89].

The GitHub Archive is an open source project to record the public GitHub timeline, archive it, and make it easily accessible for further analysis [87]. The archive data is encoded in the Json format as reported by the GitHub API, and each archive stores various aspects such as user information, event types, repository information, and the time stamps of the public events made by developers all over the world in a specific time period. To clarify, we explain the related data fields in the original data as follows.

Each data entry defines an event performed by an open source developer in the community, and it is a multi-level tree structure including the information about the event ID, the actor, the event type, the repo and the time stamp. We do a traversal on the tree structure and perform string manipulations on essential nodes to build the original data set. Specifically, we focus on the data fields explained as follows.

An actor node defines all related information of the user who committed the event, and a repo node defines all related information about the repository involved in the event. A repository is an open source project created by software community developers in public, and developers execute different events via repository manipulations.

The event types define the categories of collaboration or communication performed by the users. Typically, the events can be classified to the following 12 categories, namely: *ReleaseEvent, PullRequestEvent, PushEvent, ForkEvent, WatchEvent, CreateEvent, IssuesEvent, IssueCommentEvent, DeleteEvent, GollumEvent, CommitCommentEvent and PullRequestReviewCommentEvent*. All types of communication and collaboration are provided and recorded as the event types listed above, and the actual meaning and effects of these events can be found in [88]. The definition of several typical events are listed as follows.

| | |
|---|---|
| *PushEvent:* | a developer commits changes to a repository. |
| *PullRequestEvent:* | a developer tells others about changes recently pushed to a repository. |
| *WatchEvent:* | a developer marks other's repository for further update. |
| *ForkEvent:* | a developer makes a copy of other's repository to freely experiment with changes without affecting other developers' original project. |
| *IssueCommentEvent:* | a developer makes comments on tasks, bugs, or enhancements in a repository. |

The time stamp in each event entry records the created time of the current event by standard Greenwich Mean Time. For interpreting the time series data accurately, we perform data sampling intervals of the first 10 minutes in each hour during one week time period from 2015-01-01 to 2015-01-07, and we set the corresponding number of time series to 7. The illustration of data sampling is as Figure 6.1.

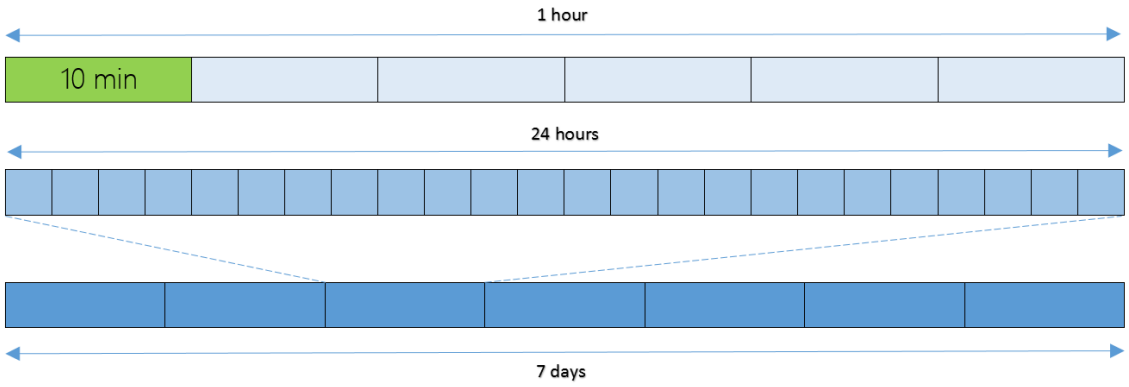



Figure 6.1: Illustration of data sampling.

We obtain around 2.5 million original data entries in the original database by sending HTTP requests. To build connections for the developer community in the original data set, we filter out all events that involve communication or collaboration as the event types listed above. However, since the original data set does not include geo-spatial information, we then obtain the location information for all developers from their profiles in each data entry by sending requests to GitHub official API server, as a subset of the developers provide

their location information in public in their GitHub account profiles, and we integrate them to the original communication data set.

However, the location strings in the open source developers' profiles are not in a unified format, and the location strings have to be mapped to corresponding numerical values in a coordinate system. We therefore use a Geocoding service from Google Map API to encode the location strings we obtained. Geocoding is the process of transforming a description of a location (such as an address, name of a place, or coordinates) to a location on the earth's surface (latitude and longitude).

The formatted location string is a city name and country code pair, and we also record full city and country names for providing detailed information in the detail view. As we build our dataset based on location strings including city and country names, the geocoding service sometimes provides responses that are too local which causes inconsistencies on the level of location strings. So we then filter out the location strings with populations less than 180,000. By performing a geocoding process on the original data, we ensure that the location strings in our visualization system are in consistent format, and are mapped accurately to latitude and longitude values on the earth.

Next, we do a Mercator projection [90] on the previously obtained data set. The reason why a Mercator projection is necessary is because that the shapes or sizes of the nodes and edges in the visualization area are distortions of the true layout of the earth's surface. Mercator projection is the standard map projection for nautical purposes [90], which maps latitude and longitude values to a rectangle area by a cylindrical projection.

Finally, we reconstruct the structure of the obtained data entries into separate files related to node, edge and detailed information in Json format. The node data records node codes, node IDs, and coordinate values, while the edge data involves node connections via source and target node IDs. A sample output data entry for a detailed edge description after processing is described in the Json format below.


```
{
    "go":{"origin": "SF, US",
          "origin_name": "San Francisco, United States",
          "dest": "Oakland, US",
          "event_dict":[{"WatchEvent":2,"IssuesEvent":12,"PushEvent":3}…
                        {"WatchEvent":10,"IssuesEvent":2,"IssueCommentEvent":2,
                        "PushEvent":20}],
          "dest_name": "Oakland, United States",
          "data": [17, 8 … 34]},
```

```
    "come":{"origin": "Oakland, US",
            "origin_name": "Oakland, United States",
            "dest": "SF, US",
            "event_dict":[{"PullRequestEvent":3,"WatchEvent":1,"IssuesEvent":1,
                           "PushEvent":1}…{"WatchEvent":21,"CreateEvent":2}],
            "dest_name": "San Francisco, United States",
            "data": [6, 14 … 23]}
}
```


## 6.3 Results and Discussion

In this section, we demonstrate our model's ability to facilitate the analysis of geo-spatial social network graphs by providing several representative use cases, and we organize the use cases by different scenarios in the following sub sections.

We conduct use cases based on the collaboration and communication data from open source community developers, with 250 major cities all over the world, more than 2,200 direct connections between them, and around 28,000 collaboration events. We first provide an overview of the sampled worldwide collaboration and communication graph in GitHub open source community, as illustrated in Figure 6.2.

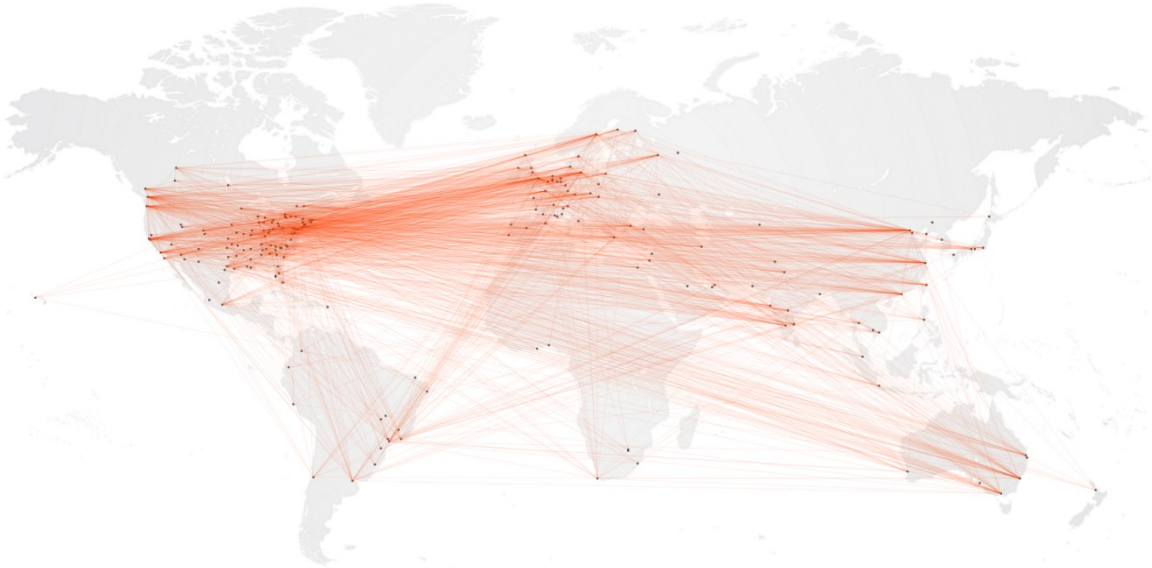


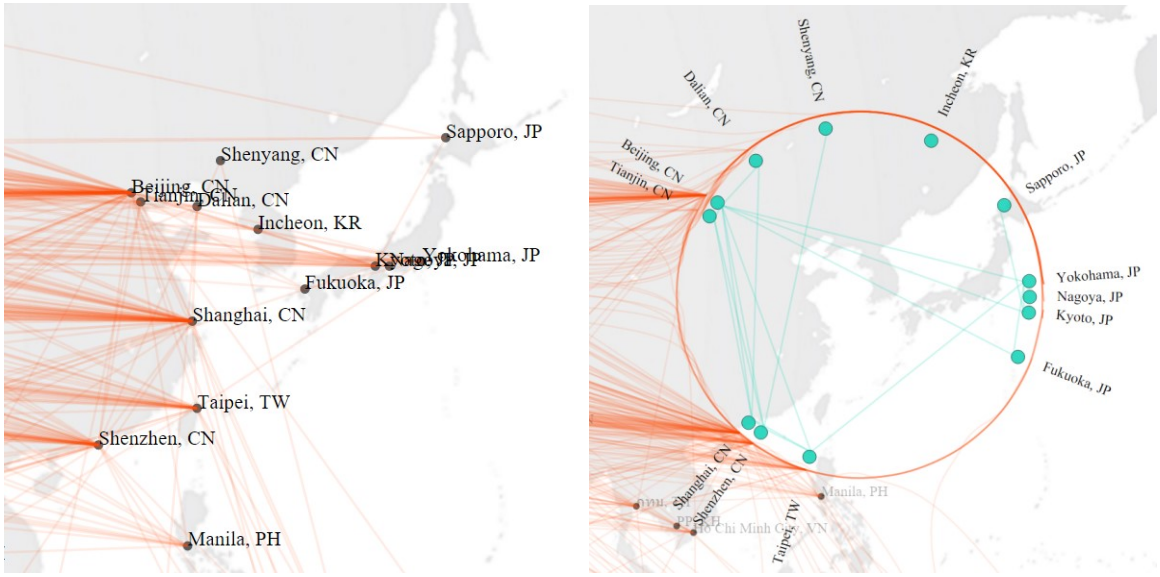
Figure 6.2: Worldwide collaboration and communication graph in GitHub open source community.

From the worldwide collaboration and communication graph we can see that a very high density of communication events are performed by developers in North America and

Europe, and developers in Asia, Africa, South America and Australia also contribute to the community from major cities.

### 6.3.1 Scenario 1 – Discovering Regional Collaboration Patterns

We begin by demonstrating the ability of our model to represent developer connections in regional areas. Figure 6.3 shows the generated circular layout with labels in East Asia, which involves CN (China), JP (Japan) and KR (South Korea). From Figure 6.3 (a) we note some obvious edge patterns that developers in Beijing, Shanghai, Shenzhen, Taipei and several large cities in Japan have strong connection bundles with other cities in western countries. However, we cannot observe the connection patterns between the regional cities clearly in the original layout, especially for a group of cities in Japan that are cluttered seriously together.




(a) (b)

Figure 6.3: Regional connection patterns with developers in eastern Asia. (a): Original layout with labels. (b): A generated circular layout by initial selection.

Figure 6.3 (b) shows the result of generating a circular layout in Figure 6.3 (a) by selection. Compared to the initial layout in Figure 6.3 (a), we can tell that the cluttered Japanese cities are Yokohama, Nagoya, and Kyoto according to Figure 6.3 (b), and the open source developers in these 3 cities have performed collaboration events with

developers in Beijing and Taiwan. We also note that developers in Incheon, Korea do not contribute to the regional community in the selected time period, while we cannot observe this pattern in the original layout, as the city is obscured with several edges going through the surrounding area in Figure 6.3 (a).

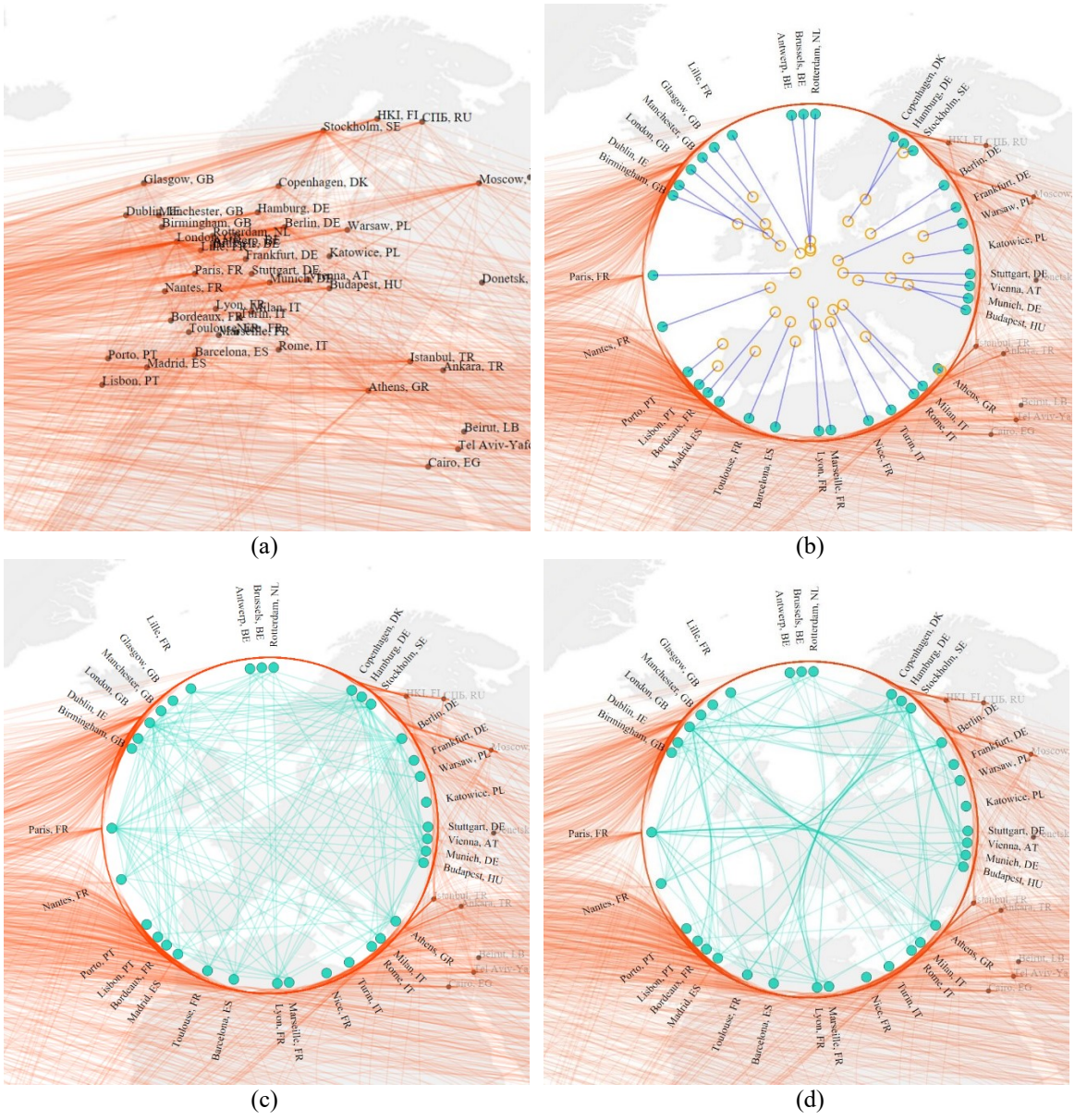



Figure 6.4: Regional connection patterns with developers in Europe. (a): Original layout with labels. (b): Visual glyphs are shown to remind the original node positions after the initial selection. (c): Linear bundle result for Europe. (d): force-direct bundle result for Europe.

Another interesting result is demonstrated by European developers, as illustrated in Figure 6.4. The software developers in Europe are one of the most active and diligent

developer groups that contribute to the open source community. Due to the large amount of communications within and without the regional area, the connection graph in Europe suffers severe clutter problems, as shown in Figure 6.4 (a).

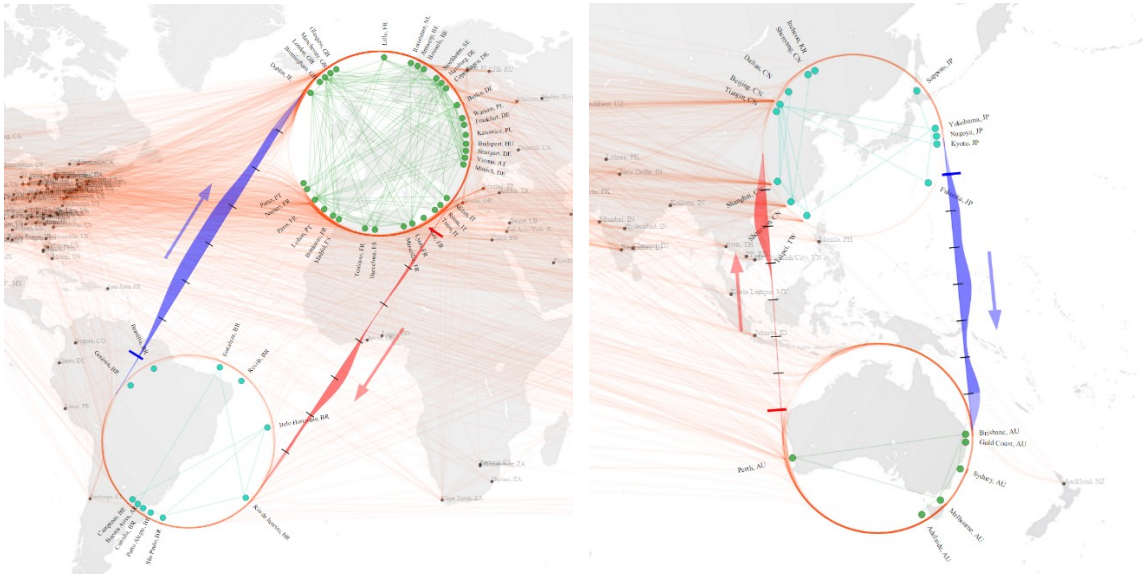
Figure 6.4 (b) shows the generated circular layout with visual glyphs to indicate the original positions of the nodes. We note from Figure 6.4 (b) that cities in Europe are neatly mapped to the circumference by a radial mapping, and the visual glyphs clearly show the actual locations for the selected cities on the layout. By observing Figure 6.4 (b) we gain some knowledge about the clustering of main cities by their locations. For example, the cities in GB (Great Britain) and IE (Ireland) are grouped to the upper left corner of the circular layout, and cities in FR (France), PT (Portugal) and ES (Spain) are located near the left and bottom of the circular layout, while cities in DE (Germany), SE (Sweden), PL (Poland) and IT (Italy) are settled to the upper right and bottom right areas, respectively.

Figure 6.4 (c) and Figure 6.4 (d) demonstrates the comparison between applying linear bundle and force-directed bundle to the regional area. Compared to Figure 6.4 (c), the force-directed bundling result in Figure 6.4 (d) reveals more information about edge trend patterns in the regional area. Specifically, we note from Figure 6.4 (d) that London, Stockholm, Paris, Lyon and Berlin leads much stronger edge bundles than other cities in Europe, that means that the open source developers in those cities are very active and they are contributing much to the regional community during the selected time period. We may further infer that those cities are assembly neighborhoods for European software developers.

### 6.3.2 Scenario 2 – Discovering Global Collaboration Flows

In this section we discuss the scenarios when applying our approach to the analysis of the high level collaboration and communication patterns across the world. Figure 6.5 (a) shows the hub view generated between open source developer groups in Europe and 2 countries in South America, namely: BR (Brazil) and AR (Argentina), during time stamp from 2015-01-01 to 2015-01-07. The directional cues in Figure 6.5 (a) clearly point out the data flow directions for the hub belts, and time series axis are shown to facilitate the recognition of time intervals.

From the hub view result in Figure 6.5 (a) we note that the developers in Brazil and Argentina committed a more uniform number of events during every day in the selected time period to European developers than European developers' feedbacks. Compared to developers in South America, who started performing events on the second day of the time series as illustrated by a blue flow, the European developers contributed much less in the first three days with a concentration of most events commitments in 2015-01-05 according to the red flow.



(a) (b)

Figure 6.5: Collaboration patterns across regions. (a): Hub view generated between city groups in Europe and South America. (b): Hub view generated between city groups in East Asia and Australia.

Another case is demonstrated in Figure 6.5 (b), with the collaboration patterns between developers in East Asian countries and Australia. We note from Figure 6.5 (b) that developers in East Asia sent more communication events to Australian developers on dates 2015-01-02, 2015-01-04 and 2015-01-07 as represented by a blue belt, while Australian developers scarcely communicate with East Asian developers during the first 5 days in 2015, as illustrated with a red belt. We may further infer that Australian developers adhere to their holiday calendars more strictly, as 2015-01-05 was the first Monday in that year, and East Asian developers may prefer more flexible schedules on holidays.

North America and Europe are 2 of the most active regions for developers to collaborate and communicate in open source communities around the world, as a high density of direct

connections and strong data flows are reflected in the collaboration graph. Figure 6.6 demonstrates the ability of our model to present complex connection and data flow patterns between regions with high density nodes and edge relationships, by showing the hub view with force-directed bundles between developer groups in North America and Europe. We note that the collaboration flows from Europe to North America, as illustrated by the red belt, showed a steady and slow growth along the time period, while collaboration flows in the opposite direction witnessed an obvious increase from 2015-01-04 to 2015-01-05.

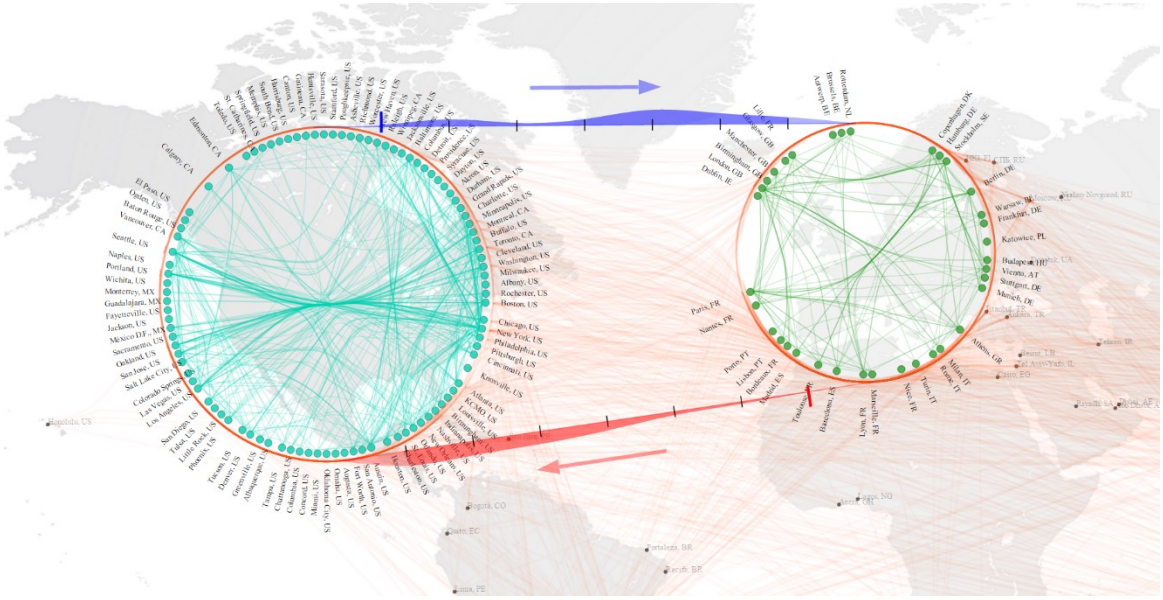


Figure 6.6: Collaboration patterns between open source developers in North America and Europe.

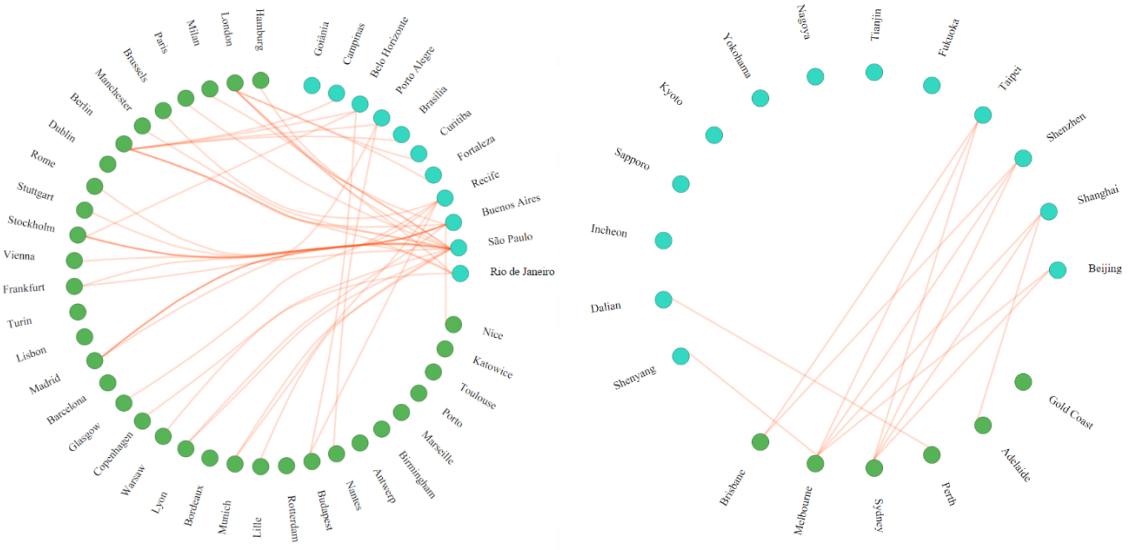
### 6.3.3 Scenario 3 – Discovering Aggregated, Detailed and Categorized Patterns

Several additional features can also be utilized with the GitHub open source data set to demonstrate aggregated and categorized collaboration and communication patterns. Two typical cases about the analysis of direct connection patterns between hubs are provided as the ensemble hubs in Figure 6.7. Figure 6.7 (a) illustrates the force-bundled ensemble hub for South America and Europe, corresponding to the selection in Figure 6.5 (a).

We can observe some communication patterns between two groups of cities in Figure 6.7. The green nodes in Figure 6.7 (a) denotes cities in Europe while blue nodes represent cities in Brazil and Argentina, and we note that Sao Paulo and Buenos Aires gather stronger

edge bundles than other cities, which means the developers in those areas have more collaborations with European developers than other cities, and the major European cities that Brazilian and Argentine developers frequently communicate to include Berlin, Stockholm, Madrid and London, as illustrated by the bundle patterns in Figure 6.7 (a).

Figure 6.7 (b) shows the ensemble hub result corresponding to the selection in Figure 6.5 (b), which involves direct connections between East Asian cities and Australian cities. We note an interesting pattern in Figure 6.7 (b) that Japanese developers seldom communicate with Australian and Chinese developers in the specific time interval, while Australian and Chinese developers interact with each other frequently as illustrated by the edges in the bottom right of the ensemble hub.




(a) (b)

Figure 6.7: Two typical ensemble hub cases for the GitHub collaboration data set. (a): Ensemble hub between Brazilian/Argentine, and European cities, corresponding to the selection in Figure 6.5 (a). (b): Ensemble hub between East Asian cities and Australian cities, corresponding to the selection in Figure 6.5 (b).

Detailed information about open source collaboration locations and events can be visualized as cloud views and detail views, as demonstrated in Figure 6.8 and Figure 6.9 (a). Figure 6.8 (a) shows the tag cloud for related nodes in Brazil and Argentina, corresponding to the selection in Figure 6.5 (a), and Figure 6.8 (b) displays various event types committed by the developers in the same selection. Detailed information for each connected edge are also provided in the detail view by showing full names of cities and

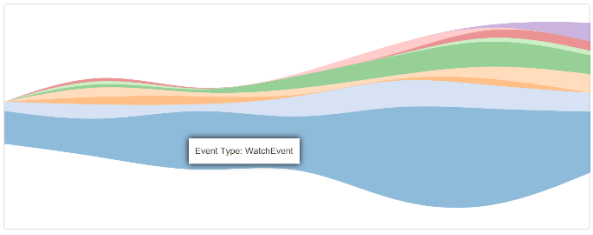
countries and exact event flow value, as demonstrated by Figure 6.9 (a), corresponding to the selection in Figure 6.5 (b).


(a) (b)

Figure 6.8: Cloud views for city names and event types performed by Brazilian, Argentine and European developers, corresponding to the selection in Figure 6.5 (a). (a): Cloud view for related city names. (b): Cloud view for related event types.

Apart from the aggregated direct connection information shown by the ensemble hub views, additional aggregated and categorical information can also be demonstrated by hub stream views, as illustrated by Figure 6.9 (b), which shows the aggregated event flow information for all of the current selected elements categorized by different event types. For example, the navy blue stream in Figure 6.9 (b) denotes the flow pattern for "WatchEvent" and we can see that "WatchEvent" occupies a large percentage among all types of events performed by Australian and East Asian developers.

| Edge | Source | Dest | Flow |
|---|---|---|---|
| 30 | Shanghai, China | Beijing, China | 119 |
| 76 | Shenzhen, China | Beijing, China | 56 |
| 80 | Shenzhen, China | Brisbane, Australia | 2 |
| 86 | Taipei, Taiwan | Melbourne, Australia | 1 |
| 121 | Sydney, Australia | Melbourne, Australia | 13 |
| 365 | Shanghai, China | Shenzhen, China | 35 |




(a) (b)

Figure 6.9: Detail view and hub stream view for Australian and East Asian developers, corresponding to the selection in Figure 6.5 (b). (a): Detail view. (b): Hub stream view.

# Chapter 7

# Conclusions and Future Work

In this thesis we propose a geo-spatial approach that visualizes both local and global information by introducing dynamic circular layouts to reduce the visual clutter in local areas in geo-spatial network graphs. The capabilities of our approach is enabled by a series of algorithms and features as follows.

An edge curving algorithm is studied to smooth the edge trend patterns around the circular layouts. Node positioning algorithms and edge bundling techniques are proposed and utilized to make good use of the advantage brought by the generation of circular layouts, to show underlying connection patterns among local data. Peapod and hub models are established to visualize data flow patterns at both the local and global scales based on the previous dynamic approach. Furthermore, the visualization system is supported by a series of additional features to show categorized, detailed and aggregated information as well.

All of the main algorithms related to our model are examined by complexity and quantitative analysis for performance considerations, and the analysis results show that the efficiency and scalability of the algorithms in our approach is at an acceptable level. The approach is further examined by two significant case studies in different application fields, with which the effectiveness of our model is demonstrated for assisting geo-spatial data exploration.

Nevertheless, we admit that limitations exist in consideration of the following aspects. First, a tradeoff is made between the clarity offered within the circular layout and the use of the narrow space around it, by pushing unrelated edges together. This technique has the limitation of causing clutter in a narrow area around the circular layouts. However, this technique operates under the assumption that this dynamic temporary deformation is desirable in order to more closely examine a set of nodes that are of particular interest at present. So, we argue that it is not unreasonable to impose a temporary deformation on the remainder.

A similar tradeoff is made by generating a radial layout of the selected nodes, which trades the retention of some positional information for visualizing connection patterns

among the nodes when needed, and we note that the uniformed layout may further trade more positional information for clarity than the radial layout. The limitation of this technique is that we may lose the precise location values of the nodes temporarily by repositioning the nodes.

From the perspective of geographic information system design, a number of other systems have moved the locations of points and have even distorted the shape of countries and regions [94, 95， 82]. It is a well-established approach in geographic information systems to clarifying relationships and data. For example, various types of fish eye lenses distort the relative distances between objects [24, 56, 57, 58, 59, 60], and the well-known "London Underground Map" [93] distorts both the shape of the subway lines and the distances between stations, to improve clarity. And at a more fundamental level, all 2D planar projections of the 3D earth are already distortions of what is really on the world which is a sphere [90].

Apart from the design concepts of geographic information systems, we incorporate both static and dynamic approaches to mitigate this issue. To establish a static solution, we introduce an optional feature for recalling the original spatial information of the related nodes by visual glyphs when needed. Furthermore, the animation process is provided to illustrate the displacement of related nodes during layout generation and dynamic placement, which also somewhat weakens the limitation mentioned above.

Second, the force-directed technique for inner edge bundling of the circular layouts suffers a certain degree of performance issues when applied to node-link graphs with extraordinarily high edge densities, due to the fourth-order computational complexity possessed by the original approach [19]. Although acceptable performance can be achieved on most common datasets, as demonstrated in Chapter 4, the bottleneck of our approach may appear in the future since the visualization of massive data is becoming increasingly significant as the time passes. Therefore, acceleration in GPU processors with parallel computing, or other progress in edge bundling techniques which allow less time consumption might be explored in further studies.

Third, our approach is not fully examined from the perspective of formal user experiments which can expose potential accessibility issues. By working closely with several peers in our laboratory while collecting informal feedback, endeavored to build a

model with a more convenient user experience. However, the usability cannot be demonstrated without further research, which is based on the analysis of accurate user data by statistical approaches. Though given the complexity of the system would make this challenging, future research directions may include further usability testing approaches discussed in [83].

Throughout the development of the system, some useful suggestions were obtained from peer researchers in our group, and some further novel ideas are identified by ourselves as well, which could possibly enhance our visualization approach in the future. We herewith summarize two directions for future improvements as follows.

First, a set of more flexible and representational elements and forms might still be incorporated. For example, “peas” that represent peak values or events in a data flow might be integrated to both peapod model and hub stream view, to convey more obvious fluctuation patterns. Moreover, further radial variations might make use of the pie chart like nature of the circular layouts, which employs a center focus node and visual glyphs occupying the empty spaces to maximize space utilization.

Second, the interaction techniques in our approach may be further studied and expanded on the following aspects. The initial selection process might be improved with several other options. For example, the selection approach may be augmented by allowing the selection of arbitrary areas and nodes, which offers a more accurate control of the areas of interest. Moreover, clustering approaches might also be integrated to the model, by automatically generating a group of closely related nodes for a single node selection. Furthermore, a “cell model” might be explored to address the interaction between different circular layouts, which imitates the biological features of cell synthesis and cell division.

## Bibliography

[1] A. M. MacEachren, M. Gahegan, W. Pike, I. Brewer, G. Cai, E. Lengerich, and F. Hardistry, "Geovisualization for knowledge construction and decision support," *Computer Graphics and Applications*, IEEE, vol. 24, no. 1, pp. 13–17, 2004.

[2] P. Shanbhag, P. Rheingans, and M. DesJardins, "Temporal visualization of planning polygons for efficient partitioning of geo-spatial data," in *IEEE Symposium on Information Visualization*, pp. 211–218, IEEE, 2005.

[3] C. S. Renschler, "Designing geo-spatial interfaces to scale process models: The geowepp approach," *Hydrological Processes*, vol. 17, no. 5, pp. 1005–1017, 2003.

[4] A. Altmaier and T. H. Kolbe, "Applications and solutions for interoperable 3D geo-visualization," in *Proceedings of the photogrammetric week*, pp. 251–267, 2003.

[5] S. G. Eick, "Aspects of network visualization," *Computer Graphics and Applications*, IEEE, vol. 16, no. 2, pp. 69–72, 1996.

[6] T. Nagel, E. Duval, and A. Vande Moere, "Interactive exploration of geospatial network visualization," in *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pp. 557–572, ACM, 2012.

[7] A. Pang, "Visualizing uncertainty in geo-spatial data," in *Proceedings of the Workshop on the Intersections between Geospatial Information and Information Technology*, pp. 1–14, National Research Council Arlington, VA, 2001.

[8] M. P. Kwan, "Gis methods in time-geographic research: Geocomputation and geovisualization of human activity patterns," *Geografiska Annaler: Series B, Human Geography*, vol. 86, no. 4, pp. 267–280, 2004.

[9] A. M. MacEachren, "An evolving cognitive-semiotic approach to geographic visualization and knowledge construction," *Information Design Journal*, vol. 10, no. 1, pp. 26–36, 2001.

[10] O. Ersoy, C. Hurter, F. V. Paulovich, G. Cantareiro, and A. Telea, "Skeletonbased edge bundling for graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2364–2373, 2011.

[11] C. Hulse, "Geotagging visualization of philadelphia," http://www.geekadelphia.com/2011/02/16/geotagging-visualization-of-philadelphia/, 2011.

[12] Finnish Geospatial Research Institute, "Analysis and visualisation of geospatial data," http://www.fgi.fi/fgi/research/researchgroups/analysis-andvisualisation-geospatial-data-geova, 2016.

[13] A. Disney, "Keylines v2.11," http://cambridge-intelligence.com/blog/, 2015.

[14] E. Malykhina, "Government data + maps: 10 great examples," http://www.informationweek.com/government/cloud-computing/government-data-+-maps-10-great-examples/d/d-id/1269530, 2014.

[15] A. M. MacEachren and M. J. Kraak, "Research challenges in geovisualization," *Cartography and Geographic Information Science*, vol. 28, no. 1, pp. 3–12, 2001.

[16] S. Elwood, "Geographic information science: new geovisualization technologies–emerging questions and linkages with giscience research," *Progress in Human Geography*, 2008.

[17] R. J. Phillips and L. Noyes, "An investigation of visual clutter in the topographic base of a geological map," *The Cartographic Journal*, vol. 19, no. 2, pp. 122–132, 1982.

[18] R. Rosenholtz, Y. Li, and L. Nakano, "Measuring visual clutter," *Journal of vision*, vol. 7, no. 2, pp. 17–17, 2007.

[19] D. Holten and J. J. Van Wijk, "Force-directed edge bundling for graph visualization," in *Computer Graphics Forum*, vol. 28, pp. 983–990, Wiley Online Library, 2009.

[20] G. Ellis and A. Dix, "A taxonomy of clutter reduction for information visualisation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1216–1223, 2007.

[21] W. Peng, M. O. Ward, and E. A. Rundensteiner, “Clutter reduction in multidimensional data visualization using dimension reordering,” in *IEEE Symposium on Information Visualization*, pp. 89–96, IEEE, 2004.

[22] N. Wong, S. Carpendale, and S. Greenberg, “Edgelens: An interactive method for managing edge congestion in graphs,” in *IEEE Symposium on Information Visualization*, pp. 51–58, IEEE, 2003.

[23] G. Ellis and A. Dix, “Enabling automatic clutter reduction in parallel coordinate plots,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 717–724, 2006.

[24] G. W. Furnas, Generalized fisheye views, vol. 17. ACM, 1986. [25] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li, “Geometry-based edge clustering for graph visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1277–1284, 2008.

[26] D. Holten, “Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, 2006.

[27] S. Peterson, M. Axholt, and S. R. Ellis, “Managing visual clutter: A generalized technique for label segregation using stereoscopic disparity,” in *IEEE Virtual Reality Conference*, pp. 169–176, IEEE, 2008.

[28] M. Bostock, V. Ogievetsky, and J. Heer, “D3 data-driven documents,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301– 2309, 2011.

[29] H. Thorvaldsd´ottir, J. T. Robinson, and J. P. Mesirov, “Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration,” *Briefings in bioinformatics*, vol. 14, no. 2, pp. 178–192, 2013.

[30] K. L. Ma and D. M. Camp, “High performance visualization of time-varying volume data over a wide-area network,” in *ACM/IEEE 2000 Conference on Supercomputing*,, pp. 29–29, IEEE, 2000.

[31] S. Ko, S. Afzal, S. Walton, Y. Yang, J. Chae, A. Malik, Y. Jang, M. Chen, and D. Ebert, “Analyzing high-dimensional multivariate network links with integrated anomaly detection, highlighting and exploration,” in *IEEE Conference on Visual Analytics Science and Technology*, pp. 83–92, IEEE, 2014.

[32] C. Hurter, O. Ersoy, S. I. Fabrikant, T. R. Klein, and A. C. Telea, “Bundled visualization of dynamicgraph and trail data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 8, pp. 1141–1157, 2014.

[33] D. Selassie, B. Heller, and J. Heer, “Divided edge bundling for directional network data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2354–2363, 2011.

[34] A. Lambert, R. Bourqui, and D. Auber, “3D edge bundling for geographical data visualization,” in *14th International Conference on Information Visualisation*, pp. 329–335, IEEE, 2010.

[35] M. P. Consens, I. F. Cruz, and A. O. Mendelzon, “Visualizing queries and querying visualizations,” *ACM Sigmod Record*, vol. 21, no. 1, pp. 39–46, 1992.

[36] S. Mukherjea and J. D. Foley, “Visualizing the world-wide web with the navigational view builder,” *Computer Networks and ISDN Systems*, vol. 27, no. 6, pp. 1075–1087, 1995.

[37] Palo Alto Networks, “Application Usage & Threat Report,” http://researchcenter. paloaltonetworks.com/app-usage-risk-report-visualization/, 2016.

[38] N. Henry, J. D. Fekete, and M. J. McGuffin, “Nodetrix: a hybrid visualization of social networks,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1302–1309, 2007.

[39] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, “Algorithms for drawing graphs: an annotated bibliography,” *Computational Geometry*, vol. 4, no. 5, pp. 235–282, 1994.

[40] I. Herman, G. Melan¸con, and M. S. Marshall, “Graph visualization and navigation in information visualization: A survey,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 24–43, 2000.

[41] G. J. Wills, “Nicheworks – interactive visualization of very large graphs,” *Journal of computational and Graphical Statistics*, vol. 8, no. 2, pp. 190–212, 1999.

[42] J. Abello, F. Van Ham, and N. Krishnan, “Ask-graphview: A large scale graph visualization system,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 669–676, 2006.

[43] J. Heer and D. Boyd, “Vizster: Visualizing online social networks,” in *IEEE Symposium on Information Visualization*, pp. 32–39, IEEE, 2005.

[44] A. G. F.Teacher and D. J. Griffiths, “Hapstar: automated haplotype network layout and visualization,” *Molecular Ecology Resources*, vol. 11, pp. 151–153, 2011.

[45] Q. V. Nguyen and M. L. Huang, “A space-optimized tree visualization,” in *IEEE Symposium on Information Visualization*, pp. 85–92, IEEE, 2002.

[46] H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen, “Energy-based hierarchical edge clustering of graphs,” in *IEEE Pacific Visualization Symposium*, pp. 55–61, IEEE, 2008.

[47] A. Lambert, D. Auber, and G. Melan¸con, “Living flows: enhanced exploration of edge-bundled graphs based on gpu-intensive edge rendering,” in *14th International Conference on Information Visualisation*, pp. 523–530, IEEE, 2010.

[48] D. Peng, N. Lu, W. Chen, and Q. Peng, “Sideknot: Revealing relation patterns for graph visualization,” in *IEEE Pacific Visualization Symposium*, pp. 65–72, IEEE, 2012.

[49] D. Zhu, K. Wu, D. Guo, and Y. Chen, “Parallelized force-directed edge bundling on the GPU,” in *11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science*, pp. 52–56, IEEE, 2012.

[50] Q. Nguyen, S. H. Hong, and P. Eades, “TGI-EB: A new framework for edge bundling integrating topology, geometry and importance,” in *Graph Drawing*, pp. 123–135, Springer, 2011.

[51] C. Hurter, O. Ersoy, and A. Telea, “Smooth bundling of large streaming and sequence graphs,” in *IEEE Pacific Visualization Symposium*, pp. 41–48, IEEE, 2013.

[52] A. Telea and O. Ersoy, “Image-based edge bundles: Simplified visualization of large graphs,” in *Computer Graphics Forum*, vol. 29, pp. 843–852, Wiley Online Library, 2010.

[53] Q. Nguyen, P. Eades, and S. H. Hong, “StreamEB: Stream edge bundling,” in *Graph Drawing*, pp. 400–413, Springer, 2012.

[54] K. P. Yee, D. Fisher, R. Dhamija, and M. Hearst, “Animated exploration of dynamic graphs with radial layout,” in *InfoVis*, p. 43, IEEE, 2001.

[55] M. J. McGuffin and I. Jurisica, “Interaction techniques for selecting and manipulating subgraphs in network visualizations,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 937–944, 2009.

[56] T. A. Keahey and E. L. Robertson, “Techniques for non-linear magnification transformations,” in *InfoVis*, p. 38, IEEE, 1996.

[57] Y. K. Leung and M. D. Apperley, “A review and taxonomy of distortion-oriented presentation techniques,” *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 2, pp. 126–160, 1994.

[58] Senseable City Lab, “Visual explorations of urban mobility,” http://senseable.mit.edu/visual-explorations-urban-mobility/data-lenses.html, 2016.

[59] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, “Toolglass and magic lenses: the see-through interface,” in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pp. 73–80, ACM, 1993.

[60] M. Sarkar and M. H. Brown, “Graphical fisheye views of graphs,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 83–91, ACM, 1992.

[61] T. Klein, M. van der Zwan, and A. Telea, “Dynamic multiscale visualization of flight data,” in *2014 International Conference on Computer Vision Theory and Applications*, vol. 1, pp. 104–114, IEEE, 2014.

[62] F. Van Ham and A. Perer, “search, show context, expand on demand: Supporting large graph exploration with degree-of-interest,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 953–960, 2009.

[63] S. Havre, B. Hetzler, and L. Nowell, “Themeriver: Visualizing theme changes over time,” in *IEEE Symposium on Information Visualization*, pp. 115–123, IEEE, 2000.

[64] S. Havre, B. Hetzler, and L. Nowell, “ThemeRiver™: In search of trends, patterns, and relationships,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, pp. 9–20, 2002.

[65] L. Byron and M. Wattenberg, “Stacked graphs – geometry & aesthetics,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1245– 1252, 2008.

[66] W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. J. Gao, H. Qu, and X. Tong, “Textflow: Towards better understanding of evolving topics in text,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2412– 2421, 2011.

[67] J. J. Van Wijk and E. R. Van Selow, “Cluster and calendar based visualization of time series data,” in *IEEE Symposium on Information Visualization*, pp. 4–9, IEEE, 1999.

[68] M. Weber, M. Alexa, and W. M¨uller, “Visualizing time-series on spirals,” in *InfoVis*, p. 7, IEEE, 2001.

[69] N. Kumar, V. N. Lolla, E. J. Keogh, S. Lonardi, and C. A. Ratanamahatana, “Time-series bitmaps: a practical visualization tool for working with large time series databases.,” in *Proceedings of the 2005 SIAM International Conference on Data Mining*, pp. 531–535, SIAM, 2005.

[70] A. Buja, J. A. McDonald, J. Michalak, and W. Stuetzle, “Interactive data visualization using focusing and linking,” in *IEEE Conference on Visualization*, pp. 156–163, IEEE, 1991.

[71] S. Cheng, P. De, S. H. C. Jiang, and K. Mueller, “TorusVis$^{ND}$: Unraveling high-dimensional torus networks for network traffic visualizations,” in *2014 First Workshop on Visual Performance Analysis*, pp. 9–16, IEEE, 2014.

[72] The jQuery Foundation, “jQuery JavaScript Library,” https://jquery.com/, 2016.

[73] M. Otto et al., “Bootstrap Framework,” http://getbootstrap.com, 2016.

[74] J. Patokallio and Contentshare Pte Ltd., “OpenFlights,” http://openflights.org/, 2016.

[75] D. Zill, W. S. Wright, and M. R. Cullen, *Advanced engineering mathematics*. Jones & Bartlett Learning, 2011.

[76] I. J. Schoenberg, “Cardinal spline interpolation,” *CBMS-NSF Regional Conference Series in Applied Mathematics*, vol. 12, 1973.

[77] United States Department of Transportation, “T-100 Domestic Market Database,” http://www.transtats.bts.gov/, 2016.

[78] L. Verlet, “Computer” experiments” on classical fluids. I. thermodynamical properties of lennard-jones molecules,” *Physical review*, vol. 159, no. 1, p. 98, 1967.

[79] J. Barnes and P. Hut, “A hierarchical O(NlogN) force-calculation algorithm,” *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[80] E. R. Tufle, *The visual display of quantitative information*. Graphics Press, Cheshire, Connecticut, 1983.

[81] C. Johnson, “Top scientific visualization research problems,” *Computer graphics and applications*, IEEE, vol. 24, no. 4, pp. 13–17, 2004.

[82] C. Panse, M. Sips, D. A. Keim, and S. C. North, “Visualization of geo-spatial point sets via global shape transformation and local pixel placement,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 749– 756, 2006.

[83] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale, “Empirical studies in information visualization: Seven scenarios,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 9, pp. 1520–1536, 2012.

[84] G. Sun, Y. Wu, S. Liu, T. Q. Peng, J. J. H. Zhu, and R. Liang, “Evoriver: Visual analysis of topic coopetition on social media,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1753–1762, 2014.

[85] N. Cao, Y. R. Lin, X. Sun, D. Lazer, S. Liu, and H. Qu, “Whisper: Tracing the spatiotemporal process of information diffusion in real time,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2649–2658, 2012.

[86] Du’up Magazine, “Peapod clip art,” http://www.duup.co.uk/x/f65a50d996, 2011.

[87] I. Grigorik, “GitHub Archive,” http://www.githubarchive.org, 2016.

[88] GitHub Inc., “GitHub API v3,” https://developer.github.com/v3/, 2016.

[89] Google Inc., “The Google Maps Geocoding API,” https://developers.google.com /maps/documentation/geocoding/introGeocoding, 2016.

[90] J. P. Snyder, *Map projections - A working manual*, vol. 1395. US Government Printing Office, 1987.

[91] L. W. Johnson and R. D. Riess, *Numerical analysis*. Addison-wesley, 1982.

[92] M. Bostock, "Force Layout API Reference," https://github.com/mbostock/d3/wiki/Force-Layout, 2011.

[93] Transport for London, "London Underground Map," https://tfl.gov.uk/maps/track/tube, 2016.

[94] D. A. Keim, C. Panse, M. Sips, and S. C. North, "Visual data mining in large geospatial point sets," *Computer Graphics and Applications*, IEEE, vol. 24, no. 5, pp. 36–44, 2004.

[95] P. Bak, M. Schaefer, A. Stoffel, D. A. Keim, and I. Omer, "Density equalizing distortion of large geographic point sets," *Cartography and Geographic Information Science*, vol. 36, no. 3, pp. 237–250, 2009.