BIG DATA VISUALIZATION USING GRAPH-BASED REPRESENTATIONS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Yi Gu

Chaoli Wang, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana

June 2016

BIG DATA VISUALIZATION USING GRAPH-BASED REPRESENTATIONS

Abstract

by

Yi Gu

Nowadays, with the extensive use of computers, huge amount of data are generated everyday. The need to understand large, complex and information-rich data sets is very important to many fields in business, science and engineering. However, due to the complicated patterns, complex hidden relationships, and multidimensional and multimodal natures of data, it is not easy for users to glean insights. These issues have become critical with the exponential growth of data over the years. In this dissertation, I focus on graph-based techniques to tackle those issues because graphs can be used to naturally encode various kinds of relationships, reveal complicated patterns, and allow users to easily identify features of interest. I demonstrate the effectiveness of graph-based representations by applying them to three kinds of data including time-varying volumetric data, image-text collections, and eye-tracking data.

Time-varying volume visualization plays an essential role in many scientific, engineering and medical disciplines. Many works have presented novel algorithms and techniques for processing, managing and rendering time-varying volume data. However, two critical challenges still remain. The first challenge is addressing visual occlusion caused by projecting the 3D data to a 2D image during rendering. This problem is exacerbated when the time and variable dimensions are considered. The second challenge is the lack of capability to help users analyze and track data change or evolution. As the size of data keeps increasing, these challenges would only become more severe. This dissertation proposes

three solutions that leverage the generalized, understandable and familiar form of graphs to address a wider range of problems for visualizing time-varying scientific data. First, I will present TransGraph that maps the evolution of a 4D space-time data set to a 2D graph representation so that the transition relationships among data could be revealed in a single graph. Through brushing and linking, users can track the evolution of spatiotemporal data over time. Second, in many cases, users can only rely on low-level visual hints from transition graphs, such as the sizes and densities of nodes and edges, to figure out interesting features to explore. This entails a heavy burden on users to identify interesting graph features and make connections to data features. I will therefore introduce graph analytics techniques, such as graph simplification, community detection, and visual recommendation, to help users explore the transition graph and understand the time-varying volumetric data. Third, I will present an indexable tree named iTree for time-varying data visualization which integrates efficient data compacting, indexing, querying and classification into a single framework.

Visualization of image-text collections is very practical in our daily lives. A significant challenge is the lack of the ability to browse, navigate and explore the large heterogeneous collection in an adaptive manner. This dissertation presents a graph-based framework called iGraph that shows the relationships among images and texts in a dynamic fashion with progressive drawing capability. iGraph also supports node comparison and visual recommendation for guided navigation and exploration.

Eye-tracking data visualization is important for understanding the patterns of eye-movements. There are two major challenges that I aim to address. First, due to the revisiting of the screen, the scanpath may overlap with each other which not only causes visual occlusion, but also obscures the reading patterns. Second, as the number of participants increases, classification and cross-comparison of the participants become increasingly difficult. I introduce ETGraph, a graph-based analytics framework that visualizes the reading patterns, identifies saccade outliers, and classifies the participants.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

During my PhD study and research, I have received great guidance, support and encouragement from a large number of individuals.

Dr. Chaoli Wang has been an excellent advisor and friend, who taught me how to convert an idea into a publication. At the beginning of my PhD life, he even provided guidance to improve my programming skills. He taught me how to manage time and shared his life experiences with me which benefited and will continue to benefit me in my life. Dr. Tom Peterka has been a great mentor during my internship at Argonne National Laboratory. He broadened my vision of research, showed me another way of doing research, and improved my knowledge in parallel computing.

I am grateful to Drs. Ching-Kuang Shene and Nilufer Onder who taught me how to design an intuitive user interface and simplify the interactions to reduce user efforts. I am also thankful to Dr. Denis Parra who enriched my knowledge of recommendation systems and Dr. Sidney D'Mello who taught me how to design a good user study. I would like to thank Drs. Robert Nemiroff, Robert Jacob, Seung Hyun Kim, and David Kao for their comments and suggestions to my projects and papers.

I am grateful to Drs. Aaron Striegel and Dong Wang for serving in my dissertation committee and providing a number of helpful comments to improve this dissertation.

I am grateful to my collaborator Jun Ma and Robert Bixler. Our joint work would not be finished without your effort.

I am thankful to my family, especially my wife Zhe Liu, for their support.

I would like to thank my colleagues, Yifei Li, Jun Tao, Man Wang, Yang Shen for accompanying me in the lab and providing suggestions for my research.

Finally, this research was supported in part by the U.S. National Science Foundation through Grants Nos. 0905008, 1017935, 1140512, 1229297, 1319363, 1456763, 1455886, the U.S. Department of Energy with Agreement No. DE-FC02-06ER25777, and the Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract No. DE-AC02-06CH11357.

CHAPTER 1

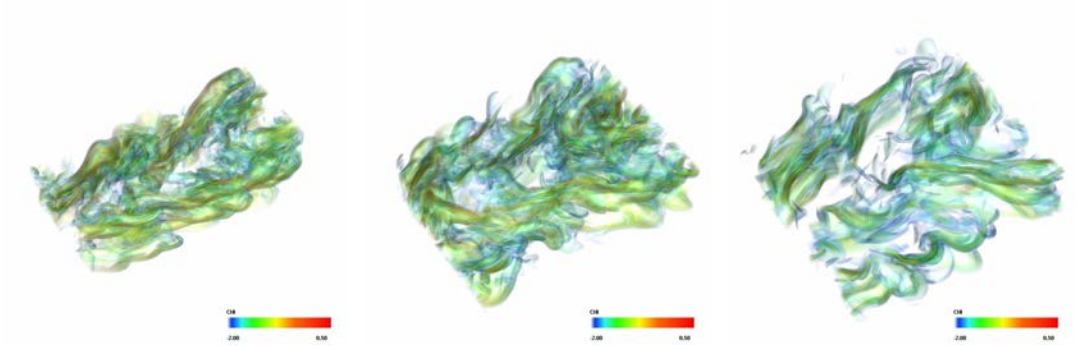
INTRODUCTION

Nowadays, with the extensive use of computers, huge amount of data are generated everyday in many different disciplines, including scientific and engineering fields, economics, and social sciences, etc. The need to understand and develop insights from the big data is urgent. However, the data are large, complex, and often unstructured, which makes it difficult for us to discover the underlying patterns and reveal the hidden relationships. Unlike the traditional analysis which aggregates the data into a few numbers, visualization approaches couple human and machine analysis by leveraging the superior human visual system to assist in knowledge discovery. It allows more aspects of the data to be observed and provides interactions to enable more purposeful exploration. In this dissertation, I focus on developing graph-based visualization approaches to understand various kinds of data. The use of graphs provides a general means to transform the data and their relationships into an abstract view for exploring complex relationships and improving data understanding. Meanwhile, graphs can also be adjusted flexibly to answer specific questions based on the distinctive characteristics of the data. I demonstrate the effectiveness of graph-based representations by applying them to three different kinds of data: time-varying volumetric data, image-text collections, and eye-tracking data.

Generating graph-based representations for different kinds of data is challenging due to the following reasons. First of all, users are interested in distinctive relationships derived from different types of data. For multivariate data sets, they may be interested in studying the relationships among variables. For example, when the auto market is studied, they may want to know the correlation between price and brand. For time-varying data sets,

the interest may shift to time evolution. For example, when the stock market is studied, it may be more interesting to predict the price of a stock based on its history. These examples indicate that the basic elements and their relationships may vary significantly for different data sets. Therefore, the definitions of nodes and edges in a graph should be designed carefully in order to extract the relationships purposefully. Second, the increasing data complexity requires more analysis effort in building connections among data points. Today's data may come from various of sources. Therefore, how to clean them, link them, and find out their relationships poses a big challenge. For instance, for a typical webpage, there could be four different kinds of objects: images, meta-tagged keywords, paragraphs of explanations, and links to other webpages. As a result, we need to deal with these four kinds of objects at least, and even more kinds of relationships among them, e.g., similarities between images, cooccurrences between keywords and between images and keywords, links between webpages, etc. Each kind of relationships may need a specific metric to measure, which significantly increases the difficulty of constructing the graphs. Finally, the ever-growing data size not only amplifies the above two issues, but also brings up other problems. The human cognition ability is limited, and therefore, a larger graph does not always translate to richer insight. Sometimes, it could do the reverse by hindering the most important relationships and patterns to be perceived by users. This calls for the need to simplify the graphs, provide concise representations, and allow users to discover the details on demand. In addition, the growing size of data also entails more computation power and more sophisticated analysis to remove noise from the data, or reduce the size so that existing approaches can be readily applied.

Besides the above challenges, there are other problems to solve for a specific kind of data. Even for the same problem, the causes and solutions may be different for various kinds of data. For example, visual occlusion is a common problem in data visualization. However, the reasons that cause the occlusion are different for the three kinds of data investigated in this dissertation. In volume rendering, visual occlusion is inevitable due to



(a) (b) (c)

Figure 1.1. Volume rendering of the combustion data set at three different time steps.

the projection of 3D volume data to 2D screen. It becomes more severe when the time and variable dimensions are considered. For visualizing image-text collections, the occlusion is caused by presenting a large number of images and texts simultaneously on the screen. For the eye-tracking data, the occlusion is unavoidable due to the various revisiting behaviors of the participants which result in the overlaps of fixations (i.e., gaze locations) and saccades (i.e., eye movements). Therefore, we need to understand the nature of a particular kind of data sets and provide specific solutions. In the following sections, I will discuss for each of the three kinds of data, the backgrounds, challenges and my proposed solutions.

1.1 Visualization of Time-Varying Volumetric Data

Volumetric data are widely used in scientific research. Examples are CT data in medical research, observed ocean current data in oceanography, and climate simulation data in meteorology, etc. These data are typically physically based and normally structured as regular 3D volume grids. Volumetric data are typically in the form of scalar field data, since a scalar value is assigned to each voxel (i.e., volume element). If the values change over time, this kind of data is called time-varying volumetric data. Figure 1.1 displays volume

rendering of the combustion data set at three different time steps. Visualizing volumetric data at a single time step allows scientists to observe the physical structure of the data, while visualizing time-varying volumetric data allows them to observe the evolution of data over time, verify their hypotheses, and improve computation models. Therefore, critical tasks in volume visualization are how to present the physical structure clearly so that the regions of interest can be observed and their relationships can be discovered, and how to track the changes over time. In the following subsections, I will discuss the existing approaches, the remaining challenges, and our solutions.

1.1.1 Existing Approaches

To help scientists meet their visual data analysis needs, scientific visualization emerged around the late 1980s. One of the major tasks of scientific visualization was rendering the volume data. There are two commonly used approaches for volume rendering. The first approach is *isosurface extraction* (i.e., extracting surfaces with the same scalar value in the volume), and using triangles or polygons to display the surfaces. The second approach is *direct volume rendering* which utilizes transfer functions to assign colors and opacities to voxels and renders the volume as an entire block of data. To render the time-varying data, animation is a commonly used approach.

However, rendering the time-varying data is not enough for scientists to get sufficient insight mainly because of the visual occlusion and complex underlying relationships. The occlusion is inevitable due to the projection of 3D volume to 2D screen. Considering time and variable dimensions even amplifies this problem. In addition, the underlying relationships are too complex for scientists to track. To solve these issues, researchers presented approaches for data transformation and visual representation. These representations are usually 2D representations which naturally reduce the occlusion. Besides that, researchers extracted the relationships of interest and revealed them in a graphical format so that scientists can intuitively identify and understand the underlying relationships.

For the approaches using isosurfaces, topological methods based on Morse theory were introduced to illustrate the relationships shown by isosurfaces, which include several abstract representations: contour trees, Reeb graphs, and Morse-Smale complexes. Morse theory shows that topological changes in scalar field data defined on manifolds occur at distinct isolated points, i.e., critical points. The Reeb graph shows the evolution of individual contours using these critical points and their relationships. The contour tree is a representation that records changes in the topology of the level sets (i.e., isocontours) of a scalar field. It is a special case of the Reeb graph where the graph forms a tree structure for simply connected domains. The Morse-Smale complex is a partition of the domain into cells according to the gradient of the scalar field. To handle the time dimension, researchers extended the above topological methods to time-varying versions. For example, Soho and Bajaj [114] applied the contour tree for time-varying contour tracking. They first constructed the contour tree at every time step and computed the correspondence information among contour trees across time steps given an isovalue. The topology change graph is constructed by creating a node for every intersection point and connecting each pair of intersection points where their representative contours correspond to each other. Bremer et al. [14] constructed a hierarchical merge tree for each time step with augmented attributes using a streaming algorithm and then created tracking graphs to capture the temporal evolution of features.

Besides representations of the relationships of isosurfaces, there are also plenty of representations developed for direct volume rendering. For instance, image graphs [90] is the first work that visualizes the visualization process. In this graph, each node represents an image, and an edge represents the rendering parameter changes (e.g., color, opacity, rotation, zoom, shading, etc.). Jankun-Kelly and Ma [68] introduced a spreadsheet-like interface for visualization exploration that illustrates a clear correspondence between parameters and results through spreadsheet-like organization. Wang and Shen designed hierarchical navigation interface [131] and LOD map [132] to provide an interface that helps

users navigate through multiresolution volume visualization. To visualize the evolution of the time-varying data, researchers have developed different solutions. Among them, storytelling treats the time evolution as a flow, identifies the most important time steps, and renders the corresponding volumes in the flow as milestones. For instance, Lu and Shen [87] presented an interactive storyboard for time-varying data visualization. They drew a line to represent the time flow and embedded the rendered images of sampled time steps to summarize complex data dynamics. While storytelling focuses on the most important time steps, other works treat each time step equally and visualize the detail of changes over time. Jänicke and Scheuermann [65] introduced a directed graph representation named ε-machine. This ε-machine represents the possibility that a configuration of voxels transits to another. Woodring and Shen [145] identified the features at each time using a clustering algorithm and created a directed graph to connect the clusters over time.

1.1.2 Remaining Challenges

Several major issues remain for time-varying volume data visualization. First, the occlusion cannot be completely eliminated due to the projection of 3D data to 2D images in the viewing. For the approaches using isosurfaces, the isosurfaces of several scalar values are selected for rendering, but the distribution of other scalar values is missing. For direct volume rendering, it often depends on user input to design the transfer functions. However, it may not be easy for users to select appropriate values corresponding to the features. Even for the approaches that generate transfer functions automatically, they may not be able to fulfill specific needs from different users. Second, when the data are visualized in an animated fashion, tracking data items or features over time could be very difficult. The data items and features may be occluded during some time steps. It will be difficult for users to make mental connections of the same feature when it reappears again. In addition, if there are multiple features in the data set, it is almost impossible for users to manually track all of them. Interactions are often required to track a particular feature, but they are usually

unavailable during animation. Third, typical level-of-detail or hierarchical exploration of time-varying data only allows users to navigate through the spatial and/or temporal domain in a coarse-to-fine fashion. The benefit of finding a good tradeoff between processing speed and visualization quality, however, may not actually help users in terms of gaining more control in the analysis or leading to more insight from the data. Fourth, there are few visual interfaces designed to explore complex relationships and improve data understanding for scientific data. Furthermore, among these interfaces, most of them lack enough guidance for user exploration and navigation. Usually trial and error is required to achieve a good result which requires painstaking effort. Finally, all these issues are exacerbated by the ever-growing size and complexity of time-varying data we need to deal with.

1.1.3 Our Contributions

In this dissertation, I focus on the development of graph-based representations that allow users to clearly observe volumetric data from different aspects and interact with them to meet different needs. The use of graphs allows all data blocks and their relationships to be observed without occlusion, and enables features to be easily selected through brushing and linking. TransGraph [51] transfers the transition relationships of spatiotemporal blocks into a static global graph view. Then, graph mining techniques are introduced to reduce user efforts when exploring large transition graphs [55]. Finally, iTree [52] is designed to integrate data compacting, indexing and querying into a single framework. In the following, I will present high-level descriptions of these approaches, and discuss their improvements over previous works.

In Chapter 3, I present TransGraph that maps transition relationships extracted from a 4D space-time data set to a 2D graph, and guides data exploration and tracking. TransGraph first partitions the volume at each time step into data blocks, then it groups the similar blocks along spatial and temporal directions. Transition relationships are introduced to represent the time evolution between data groups. In TransGraph, each node

represents a group of blocks, and each edge represents the transition relationships between two groups. To handle large-scale data sets, hierarchical graph structures are introduced. TransGraph not only provides a visual mapping that abstracts data evolution over time in different levels of detail, but also serves as a navigation tool that guides data exploration and tracking. Users interact with TransGraph and make connections to the volumetric data through brushing and linking. A set of intuitive queries is provided to enable knowledge extraction from time-varying data. Compared to animation-based techniques, TransGraph visualizes transitions of all time steps simultaneously in a single graph view. Therefore, it does not rely on the user's mental map to build connections among data groups.

In Chapter 4, I present a graph mining approach that automatically extracts features from a transition graph for understanding time-varying data. Beyond straightforward graph properties, users are given further guidance through a series of graph analysis techniques including graph simplification, community detection, and visual recommendation. Graph simplification condenses a large graph to a smaller one by abstracting known structures, such as fan, connector, and clique, presenting a less cluttered view for quick comprehension of the overall graph structure. Community detection organizes nodes with close relationships into groups, allowing visual comparison between groups of nodes instead of individual nodes. Visual recommendation automatically highlights individual nodes or node groups similar to user selected items, enabling users to spend more time on the actual analysis instead of painstaking interaction. The similar node groups are identified according to their structural relationships, regardless of their volume values, spatial regions, and temporal ranges. Unlike previous works, where users can only rely on low-level visual hints (such as the size and density of nodes and edges) to figure out the relationships with the underlying data through brushing and linking, this graph analytics approach introduces high-level functions, providing the intuitiveness, convenience and unprecedented capabilities for graph exploration and data comprehension.

In Chapter 5, I present my approach to construct an indexable tree called iTree for time-

(a) (b) (c)

Figure 1.2. (a) an example of tag cloud, (b) a screenshot of iMap [139], (c) iGraph that visualizes the relationships among images and texts.

varying data visualization. iTree integrates efficient data compacting, indexing, querying and classification into a single framework to support analysis and visualization of big time-varying volumetric data. This is achieved by transforming the time-activity curves (TACs) representation of a time-varying data set into a hierarchical representation called symbolic aggregate approximation (SAX). An indexable version of the data hierarchy named iSAX is further built, from which the iTree is created for visual representation of the time-varying data. A hyperbolic layout algorithm is employed to draw the iTree with a large number of nodes and provide focus+context visualization for interaction. Multiple coordinated views are enabled, consisting of the iTree, symbolic view, and spatial view, to support effective querying, searching and tracking of the time-varying data set.

1.2 Visualization of Image-Text Collections

With the booming of digital cameras and image archiving and photo sharing websites, browsing and searching through large online image collections has become a notable trend. In addition, these image collections often come with tags, such as names, keywords and hyperlinks, which explain image contents. Providing the capability of collection overview and detailed exploration is critical for understanding a large image-text collection. For

example, exploring the image-text collection on Facebook may inform users about the recently popular topics, and narrowing down to a particular user on Facebook may lead to her hobbies and social life, etc. Existing approaches analyze text collections to discover important topics (Figure 1.2 (a)), or analyze image collections to show similar images based on a user-specified image (Figure 1.2 (b)). However, the connections among images and texts are not fully utilized for users to navigate the large heterogeneous collection in an adaptive manner. In the following subsections, I will briefly introduce the existing approaches, discuss the remaining challenges, and describe an overview of my solutions and contributions.

1.2.1 Existing Approaches

Traditional approaches for visualizing image-text collections mainly focus on visualizing images or texts separately, or consider images or texts as a document. Research in this direction organizes images in various visual forms based on their similarities or other attributes (e.g., time, content, and size), so that the relationships among images can be better observed [8, 16, 118, 138]. Similar to image visualization, text visualization arranges the texts on the screen to show their attributes, such as importance, frequencies, etc. Examples include Wordle [125], ManiWordle [74], and context-preserving dynamic word cloud [29]. Other representations were developed to visualize the evolution of texts, such as the classic ThemeRiver [56], SparkClouds [81], and TIARA [143]. Although most of the research efforts were spent on visualizing images or texts separately, there are several techniques that consider images and texts together to illustrate a document, such as paper-to-PDA [15], SmartNails [9], and Document Card [115].

1.2.2 Remaining Challenges

Several challenges still remain for visualizing image-text collections. First, viewing images separately as individuals is no longer enough. In many cases, we now need the

capability to explore these images together as collections to enable effective understanding of large image data. However, due to the large size of the collection, it is impossible to clearly display all the images in the same screen. A common solution is "overview first and details on demand". In this case, how to select a subset from the image collection to provide an overview, and how to guide users to explore details of interest are critical. Second, the relationships among images and texts are not fully utilized. The existing approaches usually consider the similarities among images and the importance of texts separately, and display them in appropriate orders. In fact, the associated tags of an image can explain the image content, and several images related to one tag can provide examples of that tag. When exploring an image collection, it is desired to reveal different kinds of relationships simultaneously, i.e., not only the relationships among images and images, or tags and tags, but also those among images and tags. In addition, when two images are compared, the associated texts can also be included to provide extra information. For example, a small object may appear in both images, which is difficult to catch. But with the additional text information, users can easily identify that object by comparing their tags. Similarly, when two tags or a tag and an image are compared, their associated tags and images will be considered as well. Third, the topics of an online image collection may change over time. But it is impossible for users to discover the trend by just navigating through the images, especially when the collection is huge. Therefore, detecting the popular topics over time, and allowing users to observe the changes are of particular interest.

1.2.3 Our Contributions

All these challenges demand a flexible graph layout that dynamically and smoothly displays relevant information content at various levels of detail. To fulfill this need, I present iGraph [53, 54], a visual representation and interaction framework that utilizes a compound graph to visualize the relationships among images and texts, as shown in Figure 1.2 (c). Initially, the graph consists of exemplar images selected by the affinity propagation

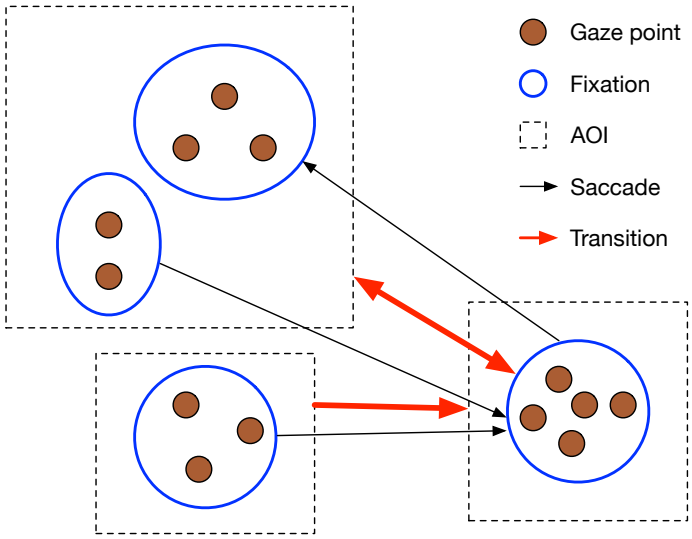
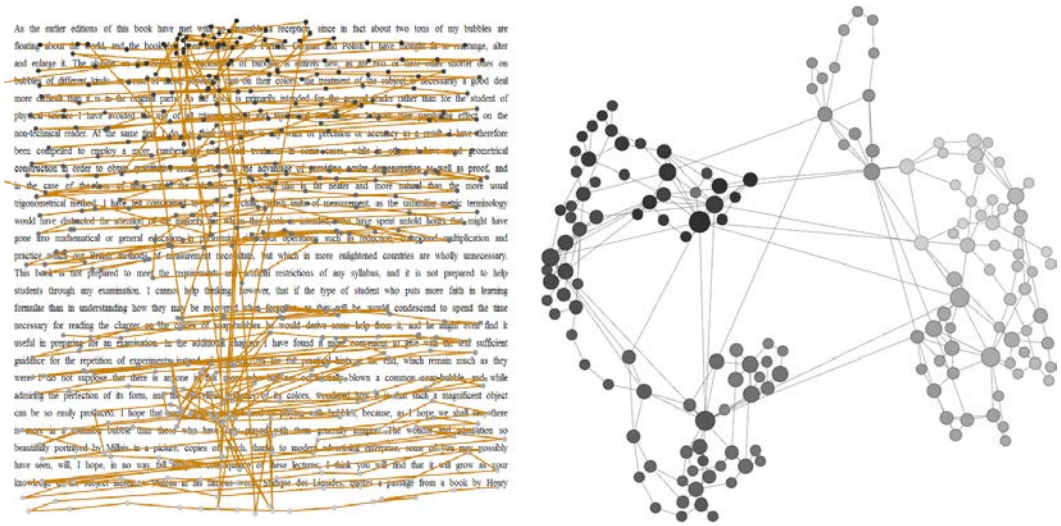


Figure 1.3. Gaze points are spatially and temporally grouped into fixations. Fixations could be grouped into areas of interest (AOI). Fixations are connected by saccades. A saccade from one AOI to the next is called a transition. A complete sequence of fixations and saccades is called a scanpath.

clustering algorithm and keywords with highest appearance frequencies. Users can then select one node (image/text), and associated nodes will be automatically recommended and displayed in the visualization, so that users can explore the entire collection in a progressive manner. They may also compare multiple nodes at the same time. The associated node group of each selected node is highlighted using a bubble set, so that users can easily identify the shared nodes and the differences among node groups. In addition, text outlier detection automatically identifies the events of interest and their time of appearance in the data. Following these events, users can discover the changes of popular topics. A parallel implementation is developed to handle large data sizes and visualize the large collection on a display wall. Finally, a user study is conducted to evaluate the effectiveness of iGraph. The details will be described in Chapter 6.



(a) (b)

Figure 1.4. (a) The scanpath of a participant's reading of a page. (b) ETGraph that shows the reading pattern using a graph representation.

1.3 Visualization of Eye-Tracking Data

With advances of the eye-tracking technology, eye-trackers are getting increasingly affordable for use in research and education. Eye tracking devices record gaze points of a participant as raw data. However, a tiny eye movement e.g., a blink, would generate a number of gaze points, which convey little information. Therefore researchers usually do not directly analyze gaze points. Rather, they would group the gaze points to produce larger forms in order to preserve information and reduce noise. Figure 1.3 shows the different terms commonly used for eye-tracking data analysis, i.e., fixation, saccade, scanpth, AOI, and transition. *Fixations* are the aggregations of gaze points based on their spatial and temporal closeness which represent the small areas that participants focus on. Rapid eye movements between fixations are defined as *saccades*. A *scanpath* is a sequence of fixations and saccades which is usually displayed as a degenerated line graph where nodes represents fixations and edges indicate saccades, as shown in Figure 1.4 (a). Since the loca-

tions of fixations do not convey intuitive information, some researchers focused on *areas of interest* (AOIs) that represent important regions on the screen, e.g., apples in a picture. Similar to saccades, the eye movements between AOIs are called *transitions*. *Mind wander(ing)* (MW) or zoning out is a ubiquitous phenomenon where attention involuntary shifts from task-related processing to task-unrelated thoughts [113].

Visualization can be used to display these eye movements to help scientists study the human behaviors. The regular eye movement may illustrate the normal patterns while the outliers may indicate abnormal phenomena, such as MWs, revisitings, and repeated readings. It is demanding for visualization tools to differentiate the outliers from the regular ones. For example, there are many limitations of measuring MW via self-report, which is the most commonly used way. It is beneficial to obtain visual behavioral indicators of MW. It is also helpful for visualization to classify the participants based on their behaviors, such as fast readers or slow readers, etc. In the following subsections, I will discuss the existing approaches along with their goals, the remaining challenges to tackle, and my solutions.

1.3.1 Existing Approaches

Heat map and scanpath are the most commonly used techniques to visualize eye-tracking data. Heat map focuses on reducing visual occlusion and capturing overall behaviors. It counts the number of fixations in a small area and utilizes colors to represent the number. Therefore, heat map nicely visualizes the densities of fixations indicating regions of interest. In contrast, scanpath visualizes the time information in saccades. The scanpath is a graph representation where each node represents a fixation and an edge represents a saccade. Therefore, it preserves the order of fixations and illustrates the overall behaviors more precisely. Furthermore, animating the scanpath makes it possible for detailed examination of eye movements and identification of moving patterns. However, due to the revisiting behaviors, visual occlusion may appear in a scanpath, which makes the patterns difficult to be perceived by human. To reduce visual clutter, some researchers

shifted their interest to AOIs. Tory et al. [119] studied the relationships between AOIs using a directed graph visualization. In such a graph, each node represents one AOI and an edge represents transitions between two AOIs. The edge thickness depicts the number of transitions. However, since many eye-tracking data do not contain AOIs, the graphs based on AOIs could not be applied to all kinds of eye-tracking data sets.

1.3.2 Remaining Challenges

Visualizing eye-tracking data is still challenging for the following reasons. First, preserving the detailed eye movement information and reducing visual occlusion are often conflicting. Heat map aggregates the density of fixations over time, and does not suffer from visual occlusion. But it loses all the saccade information, and fails to answer the questions associated with eye movements. AOI graphs denote the relationships among AOIs, which lose the precision of fixation locations. Therefore, they can only be used to discover high-level patterns. The scanpath preserves the precise locations of fixations, but visual occlusion could be severe. In addition, it normally depends on human effort to identify different eye movement patterns. Second, few works focus on participant classification. Currently, the different categories of participants are identified manually by researchers through observing their eye movement behaviors. There are few quantitative metrics to measure the similarities among participants. Third, to the best of my knowledge, few works have been done to visualize and identify MWs. The most commonly used approach to identify MWs is via self-report. However, this can be inaccurate, since participants may not notice the MWs and fail to report them. Therefore, automatically detecting and identifying MWs through visual patterns is critical.

1.3.3 Our Contributions

In Chapter 7, I present ETGraph (i.e., eye-tracking graph) that transforms the eye-tracking data gathered from a reading study into a graph view to tackle the above chal-

lenges. As shown in Figure 1.4 (b), I first cluster fixations into clusters, so that fixations within each clusters are spatially close. Each fixation cluster is represented by a node in ETGraph, and a transition between two clusters is represented by an edge. Similar to AOI graphs, ETGraph simplifies the scanpath, so that visual occlusion is reduced. Unlike AOI graphs, the granularity of locations in ETGraph can be better controlled through tuning clustering parameters. Furthermore, multiple coordinated views are utilized to dynamically link the graph view with the standard page view during interaction, so that users can switch between the simplified ETGraph and the original scanpath. In ETGraph, the normal reading patterns form a well-behaved graph structure while complex graph structures indicate anomalous reading patterns. In addition, ETGraph automatically detects saccade outliers and repeated scanpaths to identify the patterns of interest. ETGraph also considers the scanpaths of all participants and constructs a supergraph to represent the reading behaviors of all participants. This supergraph not only reveals the common reading patterns of all participants using edge bundling, but also allows users to study the detailed saccade outliers. Besides showing the patterns of all participants, ETGraph can be used to compare the graphs of two participants and to classify participants based on their similarities between corresponding graphs. Finally, a user study is conducted and the results show that ETGraph helps users identify the reading patterns, including MWs.

1.4 Organization

This dissertation is organized as follows. Chapter 2 introduces the related works and discusses how my approaches relate to and differ from these approaches. Chapter 3 presents TransGraph that maps extracted transition relationships from a 4D data set to a 2D graph for guiding relationship exploration and feature tracking. Chapter 4 introduces graph analytics techniques, such as graph simplification, community detection, and visual recommendation, for exploring large transition graphs produced from big time-varying volumetric data. Chapter 6 presents iGraph which visualizes the relationships among images and

texts, provides recommendation to help users navigate through the collection, and supports comparison to help users investigate the hidden relationships among images or texts of interest. Chapter 7 introduces ETGraph that transforms the eye-tracking data gathered from a reading study into a graph view for visual analytics. Finally, Chapter 8 summarizes the works I have completed.

1.5 Publications

My publications are listed as follows. This dissertation is based on the following publications J1 to J5, C1 and C3.

Peer-reviewed journal articles:

- J1. Yi Gu, Chaoli Wang, Robert Bixler, and Sidney D'Mello. ETGraph: A Graph-Based Approach for Visual Analytics of Eye-Tracking Data. *Computers & Graphics*, Under Review.

- J2. Yi Gu, Chaoli Wang, Jun Ma, Robert J. Nemiroff, David L. Kao, and Denis Parra. Visualization and Recommendation of Large Image and Text Collections toward Effective Sensemaking. *Information Visualization*, Accepted.

- J3. Yi Gu, Chaoli Wang, Tom Peterka, Robert Jacob, and Seung Hyun Kim. Mining Graphs for Understanding Time-Varying Volumetric Data. *IEEE Transactions on Visualization and Computer Graphics (IEEE SciVis 2015)*, 22(1):965-974, Jan 2016.

- J4. Chaoli Wang, John P. Reese, Huan Zhang, Jun Tao, Yi Gu, Jun Ma, and Robert J. Nemiroff. Similarity-Based Visualization of Large Image Collections. *Information Visualization*, 14(3):183-203, Jul 2015.

- J5. Yi Gu and Chaoli Wang. TransGraph: Hierarchical Exploration of Transition Relationships in Time-Varying Volumetric Data. *IEEE Transactions on Visualization and Computer Graphics (IEEE Vis 2011)*, 17(12):2015-2024, Dec 2011.

Peer-reviewed full conference papers:

- C1. Yi Gu, Chaoli Wang, Jun Ma, Robert J. Nemiroff, and David L. Kao. iGraph: A Graph-Based Technique for Visual Analytics of Image and Text Collections. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, San Francisco, CA, 15 pages, Feb 2015.

- C2. Yi Gu, Nilufer Onder, Ching-Kuang Shene, and Chaoli Wang. FPAvisual: A Tool for Visualizing the Effects of Floating-Point Finite-Precision Arithmetic. In *Proceedings of American Society for Engineering Education Annual Conference*, Indianapolis, IN, 17 pages, Jun 2014.

- C3. Yi Gu and Chaoli Wang. iTree: Exploring Time-Varying Data Using Indexable Tree. In *Proceedings of IEEE Pacific Visualization Symposium*, Sydney, Australia, pages 137-144, Feb 2013.

- C4. Yi Gu and Chaoli Wang. A Study of Hierarchical Correlation Clustering for Scientific Volume Data. In *Proceedings of International Symposium on Visual Computing*, Las Vegas, NV, pages 437-446, Nov 2010.

CHAPTER 2

RELATED WORK

In this chapter, we will discuss previous efforts for the visualization of scientific data, image and text collections, and eye tracking data in Section 2.1, Section 2.2, and Section 2.3, respectively. In each section, we give a general description of related works, followed by discussion of each work. Finally, we make connections between previous works and our approach, and point out the differences.

2.1 Visualization of Scientific Data

Wang and Tao [127, 133] classified the techniques using graph-based representations in scientific visualization based on their applications into four categories: partition-wise, relationship-wise, structure-wise, and provenance-wise. Works related to this dissertation mainly fall into the categories of partition-wise and relationship-wise. We organize these related works in four subsections based on the relationships they analyze, so that the connections between previous works and our approaches can be better understood. Specifically, the four relationships we review here are: hierarchical relationships, data evolution, variable relationships, and relationships among field lines.

2.1.1 Exploring Hierarchical Relationships

Tree is a special type of graph which does not contain cycles. It can be naturally used to represent hierarchical relationships. In this subsection, we mainly focus on two types of hierarchy constructions: data partitioning and data clustering.

Data Partitioning. Data partitioning hierarchically partitions the spatiotemporal data into smaller pieces which allows users to examine the data in an adaptive manner. Octree is a widely used structure to partition a single volume. For example, Wang et al. [136] introduced an application-driven approach to compress and render large-scale scientific simulation data. During preprocessing, they first partitioned the data set into space-time blocks using an octree for individual compression. Then, they packed data blocks into the graphics memory, reconstructed and rendered the data in the GPU. For time-varying volumetric data sets, time-space partitioning (TSP) tree [41, 107] considers the space dimension first and then the time dimension for time-varying data partition. Space-partitioning time (SPT) tree [35] argues that the time dimension should be partitioned first before the space dimension and compares its results with those derived from the TSP tree. Different from octree, TSP tree, and SPT tree that focus on the partition of the space-time domain, the wavelet tree [47, 134] focuses on data compression. Combining the wavelet tree with TSP tree, the wavelet-based time-space partitioning (WTSP) tree [130, 135] gains the benefits from both sides.

Besides serving as internal structures for partition, trees also serve as interfaces that guide users in their exploration. Wang and Shen [131] introduced hierarchical navigation interface, an interface with multiple coordinated views for level-of-detail (LOD) selection and rendering. There are three views, the volume rendering view, the tree view that displays the hierarchy cut, and a treemap that helps users pinpoint the target regions for exploration. Wang and Shen [132] developed LOD map to help users navigate multiresolution volume visualization. LOD map first measures LOD quality based on the formulation of entropy from information theory. The quality measure takes into account the distortion and contribution of multiresolution data blocks. Then they used a treemap representation to indicate the quality of LODs in an intuitive way, and provide immediate suggestions for possible LOD improvement through visually-striking features (such as the colors and sizes of bounding rectangles that represent data blocks in the current LOD).

Data Clustering. Besides partitioning the domain, other approaches apply clustering techniques to group data pieces and form hierarchies which could be naturally represented by trees. Lin et al. [86] utilized a cluster tree to represent the clustering of the surfaces in a multifield volume data set. They first converted the multifield particle volume data to a high-dimensional domain. After partitioning the domain into cells, they computed the cell density as the number of particles within the cell. Then they clustered the cells and stored the results in a high density cluster tree where each cluster corresponds to a surface in the object space. Gu and Wang [50] applied three hierarchal clustering algorithms to group the sample points based on their correlations and stored the hierarchy using trees. Parallel coordinates are utilized to visualize the correlations between sample points for each level of the tree. Ip et al. [64] introduced a tree-based representation to help users explore a 3D intensity field in a coarse-to-fine manner. They converted a 3D intensity field to a 2D intensity-gradient histogram and utilized the normalized-cut image algorithm to partition the histogram into segments. By iteratively partitioning the histogram, they created the hierarchy and stored it in a tree structure. This tree serves as an interface for users to explore the embedded structures clearly.

2.1.2 Understanding Data Evolution

Studying the evolution in a time-varying volumetric data is important in many scientific and engineering fields. It allows scientists to observe the changes and verify their hypotheses. Graph-based visualization aims at assisting scientists to capture the evolution in the data through simplifying the underlying evolution relationships and visualizing them as graphs. A major challenge is to identify features and extract the relationships. One commonly used solution is identifying user-defined features and match them in neighboring time steps. However, this assumes that feature changes are small, which may not always be the case. Another solution is to study the probability of changing from one feature to another, which allows dramatic feature changes.

Feature Matching. For the feature matching approaches in 3D volume tracking, features are matched through their attributes or locations, under the assumption that the changes in neighboring time steps are small. A commonly adopted strategy extracts threshold bounded features from each time step and then associates them over time. This approach requires storing the feature information for all the time steps and even calculate feature combinations for comparison.

Samtaney et al. [104] first extracted features which are the regions that fulfill certain criteria, then they performed postprocessing to map the features in neighboring time steps. Due to the memory restrictions, their program could not store the actual regions for each feature. Therefore, they stored the attributes (centroid, volume, mass) of the features. By matching the attributes, features are matched. For the neighboring two time steps, every pair of the features is compared. If the features are matched, then they are considered to be continuous. After removing all the continuous features, the combination of multiple objects is compared to test whether they are bifurcation or amalgamation. Those features without matching are considered to be disappearance or creation. Because all the feature combinations need to be compared for the decision of bifurcation and amalgamation, the computation cost is high. To reduce the cost, Silver and Wang [110] [112] utilized octrees to store all the features, which allows them to easily find the spatial overlapping between features. For a feature in one time step, they identified those features in the next time step who have the spatial overlaps with the selected one. Finally, they utilized a normalized volume difference test to match the features.

However, storing all the feature information for all time steps for comparison is very costly. To get around this problem, researchers predicted the features in the current time step based on their presence in a few previous time steps. This approach does not require all information to be stored and compared, which leads to low computation and memory cost. Reinders et al. [101] introduced a prediction verification tracking technique that predicts the features in the next time step by linear extrapolation. If there is one feature

matching the prediction, it is considered to be a feature following the previous trend. Using linear behavioral rules, they predicted the attributes of each feature in the next time step based on previous time steps. Then all the features in the next time step will be verified for the matching. However, the prediction in this approach would not be always correct. Muelder and Ma [92] presented a prediction-correction method that makes a best guess of the feature in the next time step based on the previous time steps. Because there are always differences between guesses and the actual features, they grew or shrunk the feature regions in the subsequent time steps to correct the guessing results.

Another solution is to use parallel implementation for feature prediction. Then, how to store the partial information of features and how to track them become the problems. Wang et al. [140] partitioned the volume into blocks and allocated them to processors. Each processor calculates the partial features within its block and creates a local connectivity tree with six children which corresponds to its six spatial neighbors. Then it communicates with the six neighbors to exchange the centroids and the minimal and maximal region boundaries of the features. Such information is used to match the features with the neighbors in the current time step. Then a global connectivity graph is created based on the local connectivity tree which stores all the information of all the features in that time step. This entire process is repeated in the next time step. Assuming that features change little in adjacent time steps, the movements of the features could be identified through the movements of features among processors in the global connectivity graph.

Transition Probability. Sometimes, it is quite difficult for users to match features exactly in the neighboring time steps, since the matching can be inaccurate when the changes are significant. In that case, using probability to connect features is a more practical solution. Woodring and Shen [145] identified the features at each time step using a k-means clustering algorithm. They first generated k clusters for each time step and then created a directed graph to connect the clusters over time. To identify the links between features, they estimated the probability of transferring one feature to another by computing the time

histogram distance between the two clusters. They simply filtered the edges to reduce visual occlusion in the graph.

Jänicke and Scheuermann [65] introduced a directed graph representation named ε-machine. They defined a causal state as all the required information in a given position to predict its future. For simplicity, they utilized a light-cone structure which consists of a spatiotemporal pattern of a given point. The light-cone is divided into two parts: a past configuration and a future configuration. They put all the past and future configurations into a high-dimensional space and utilized the Voronoi tessellation to partition the space based on the density. The resulting clusters are called causal states. The transition probability is calculated if a state transfers to anther. In the ε-machine, a node represents a causal state and an edge represents the transition probability between two nodes. Finally, a force-directed graph layout algorithm is used to draw the ε-machine. Later on, Jänicke and Scheuermann [66] enhanced the ε-machine with three other representations. They utilized the cone window to show the past and future configurations for each selected position, applied the matrix view to illustrate the transition probabilities, and leveraged the circular graph layout to show the transitions among states.

2.1.3 Investigating Relationships among Variables

Graphs can also be used to represent the relationships between variables. In this kind of graphs, a node usually represents a variable and an edge often represents the relationship between the two variables.

Correlation relationships have been well studied for time-varying multivariate data analysis and visualization [21, 116]. Qu et al. [99] utilized an undirected weighted graph to visualize the correlation between variables. Wang and Shen [129] discussed the use of information theory in scientific visualization. For example, mutual information in information theory can be used to calculate the shared information between two variables. Biswas et al. [10] constructed a graph where each node represents a variable, and an edge denotes

the mutual information between two nodes. Wang et al. [137] studied information transfer between variables. Since information transfer carries directional information, they utilized a directed weighted circular graph to show the information transfer between variables.

Sauber et al. [106] used a node to represent a set of variables, so that users could see the relationships between variable sets. They designed multifield-graphs to show the correlations of multifield data sets where a graph-based representation was used to show the hierarchy of variables. They computed the correlation fields between variables. These fields are placed level by level. The first level is the field which considers all the variables. The next level consists of the fields with one less variable. This process continues and the leaf nodes are the fields that compare exactly two variables.

Instead of directly studying the relationships between variables, Jänicke et al. [67] proposed a graph-based representation called attribute cloud to show the closeness of points in a high-dimensional attribute space. They first projected all the points to the attribute space, then utilized an Euclidean minimum spanning tree to connect the points based on their closeness. A threshold is provided so that the nodes whose distances are less than the threshold are also connected by edges. To visualize such a graph in the high-dimensional attribute space with less visual occlusion, they utilized a force-directed layout algorithm to draw the graph in a 2D screen.

2.1.4 Studying Field Lines

Other than volumetric data, graphs are also applied to analyze flow fields, which is extensively studied in scientific visualization as well. Similar to the use of edges to indicate transition relationships in volumetric data, the edges are used to capture relationships among field lines and spatiotemporal blocks in flow fields.

Xu and Shen [147] presented Flow Web, a graph-based representation which provides users an overview of the 3D flow field and helps users locate regions of interest. They first partitioned the domain into subregions using an octree. They then dropped random seeds

into each subregion and traced the corresponding streamlines. The edges between the nodes indicate how the flow passes through the subregions and their edge weights reflect the numbers of streamlines going through the subregions.

Ma et al. [88] designed FlowGraph, a compound graph representation for studying streamline clusters, spatial blocks and their relationships in a hierarchical manner. They defined two types of nodes (L-nodes and R-nodes) and three kinds of edges (L-L edges, R-R edges, and L-R edges). An R-node represents a spatial region. An octree partition is used to build the R-node hierarchy. A leaf L-node corresponds to a single streamline, and a non-leaf L-node represents a cluster of streamlines. The similarities between streamlines are calculated based on their shared leaf R-nodes (spatial blocks that the streamlines go through) and their mean of closest region distances. With the similarities between streamlines, they constructed the L-node hierarchy. An R-R edge is formed between two R-nodes at the same level of R-node hierarchy whose weight indicates the number of streamlines shared by these two R-nodes. An L-L edge is formed between two L-nodes at the same level of the L-node hierarchy, and the edge weight indicates the number of common leaf-level blocks traversed in order by these two L-nodes. An L-R edge is formed between a L-node and a R-node to show their interconnection whose weight indicates the number of streamlines in the L-node passing through the R-node.

Later on, Ma et al. [89] extended FlowGraph to a 3D unsteady flow field. Now an R-node represents a spatiotemporal regions. They achieved this using a 16 tree (4D tree). Moreover, they treated the unsteady field as a static 4D field, so that a 3D pathline could be treated as a 4D streamline. Finally, they used a similar approach to construct FlowGraph as a compound hierarchical graph.

While the Flow Web and FlowGraph focus on designing an interface for users to explore the vector field, other researchers applied similar strategies to improve the performance of streamline and pathline tracing. As the data size exceeds the memory size of a machine, researchers have to partition the data into blocks and load the blocks into mem-

ory when necessary. Therefore, how to organize the blocks to reduce the data fetch cost becomes a popular research topic. Nouanesengsy et al. [95] designed flow graph, a graph-based representation to estimate the computation cost of tracing particles in streamline generation. First, the vector field is partitioned into blocks which are then assigned to processors. Similar to Flow Web [147], each block is represented as a node. Then the seeds are randomly placed in the whole domain. While Flow Web counts the number of particles traveling through the boundaries of blocks, flow graph calculates the corresponding probabilities and treats them as the edge weights. This graph is used as a guidance for estimating particle movements. However, because this graph stores the information of traveling through blocks, it misses the information of the tracing within each block, for instance, the lengths of streamlines in the block. Therefore, during each round of tracing, the system combines the flow graph with the average number of tracing steps per particle to estimate the workload per block for workload balancing.

The access dependency graph (ADG) [24] was derived based on Flow Web [147] and flow graph [95] to guide the reorganization of the data for the reduction of miss ratio. ADG first partitions the data into blocks so that each block can be fetched into memory. Each block is stored in a file and a linear ordering of these files is used so that when a file is needed and uploaded to main memory, its neighboring files would also be fetched to reduce the I/O cost. To reduce the miss ratio, they needed to reorder the files. Similar to the flow graph, they constructed a graph and traced the streamlines to calculate the probabilities of moving one particle to another block. When a particle moves from one block to another, it is called one-hop. Since the one-hop ADG is not accurate enough to predict the next few hops, they considered the n-hop graphs. By considering the graphs from one-hop to n-hop together, they measured the miss ratio based on the order of all the blocks. As a result, by minimizing this ratio, they could get an linear order of ordering the blocks.

Later on, Chen et al. [23] extended the ADG [24] to time-varying flow fields for path-line tracing. Each node in the ADG is called a time block which consists of the blocks

at the same location in a time period. A directed edge connects two time blocks if there exists pathlines passing through them. Each edge is assigned a weight which equals the probability of the seeds moving from one block to the other. The finite-time Lyapunov exponent (FTLE) helps users study the existence of the Lagrangian coherence structures by quantifying the separation of flows. However, in order to get a high-resolution FTLE field, we need to trace the pathline from every point and for every time step. Because of the large data size, an out-of-core approach is required. Therefore, Chen and Shen designed flow map [22] to help users reduce the I/O cost and maximize the data reuse. They first utilized the ADG to estimate the tracing probabilities between neighboring blocks. Then the discrete-time Markov chains were applied on the ADG to predict the probabilities between all the blocks. Finally, a seed scheduling is applied to select and order the seeds for tracing at each round to maximize the usage of data blocks.

2.1.5 Our Solution

Similar to these related works [60, 61, 65, 92], we define the spatiotemporal neighboring regions with similar attributes as a feature. We first partition the volume into blocks, group similar spatiotemporal neighbors using a region growing algorithm, and treat each group as a feature. While the related works [61, 92, 111] defined a transition between two features if they share a voxel in the neighboring two time steps, we count the number of blocks that transit from one feature to another in the subsequent time steps. These numbers are used as the transition frequencies to calculate the corresponding probabilities. Finally, we implement a force-directed graph layout algorithm [44] to visualize the transition graph and provide several interactions to help users understand the evolution in an occlusion-free way. In conjunction with the force-directed layout, we implemented a focus+context method that is different from [69, 96] for showing details of particular nodes.

Different from the related works [35, 107, 144], we develop our approach based on SAX [83] to compress time-varying volume data. Specifically, a voxel's spatial neigh-

bors over time is compacted into a succinct symbolic representation, which is less time consuming than typical transform-based compression solutions. Meanwhile, we use iSAX [109] to provide cost-effective data indexing and querying. In addition, based on iSAX, we organize the compact data in the SAX form into a hierarchical structure to facilitate data indexing and querying. Finally, we use a 2D hyperbolic tree for navigating the hierarchical structure which provides a good means for focus+context visualization.

2.2 Visualization of Image and Text Collections

Visualizing image-text collections has received a great amount of attention recently. Most of the existing works, however, only focus on the organization of one type of data, i.e., either images or texts. Although some works visualize images and texts together, they typically organize images and texts into a single document, where the relationships among images and texts are not explicitly captured. Furthermore, these approaches do not provide users with enough interaction to support detailed exploration. In this section, we organize related work in the following subsections: image collections, text collections, and image-text collections. Then we discuss our solution.

2.2.1 Image Collections

Organizing image collections into various visual forms has been a well-studied topic in the graphics, imaging, and visualization community. Chen et al. [20] organized the images using the Pathfinder networks based on image colors, textures and shapes. Bederson [8] designed PhotoMesa which arranges images using quantum treemaps and bubblemaps for zoomable image browsing. Platt et al. [98] developed PhotoTOC which uses a temporally ordered list of all photographs from a user as the detailed view and utilizes an automatic clustering algorithm to generate the overview for personal photograph browsing. Torres et al. [118] introduced spiral and concentric rings for focus+context visualization in conjunction with content-based image retrieval. Jankun-Kelly and Ma [69] presented MoireGraphs

which leverages a focus+context radial graph to layout nodes and their associated images. MoireGraphs also provides several interaction techniques, such as focus strength changing, radial rotation, level highlighting, secondary foci, and animated transition, to assist graph exploration. Yang et al. [148] proposed the semantic image browser (SIB) that organizes images using an image layout based on multidimensional scaling. SIB includes the value and relationship display (which allows effective high-dimensional visualization without dimension reduction) and several interaction tools (e.g., searching by sample images and content relationship detection). Brivio et al. [16] presented dynamic image browsing which partitions the screen with weighted Voronoi diagrams to visualize images of different sizes, orientations, and aspect ratios. Wang et al. [138] designed iMap, a treemap based representation for visualizing and navigating image clustering and search results. Their layout places the query image at the center of the display area and arranges other images based on image similarity along a spiral shape from inside out. Zhang et al. [149] studied the effectiveness of animated transitions in a tiled image layout by comparing different animation schemes. They concluded that users completed the tasks faster and more accurately using multi-step animated transition compared with no-animation and one-step animation solutions.

2.2.2 Text Collections

Besides images, text analytics and visualization has received lots of attention recently. Several survey papers [3, 46, 75, 123, 141] discuss text visualization and its related research problems and challenges. There are notable text visualization examples. For instance, Wattenberg and Viegas [142] designed the word tree, a graphical version of the traditional "key word in context" method which allows users to explore the text body, find interesting words, and view additional phrases. Clarkson et al. [26] presented ResultMaps which utilizes the treemap representation to enhance query string-driven digital library search engines. It maps each repository document into a treemap and highlights query results.

ResultMaps avoids users from misled by the rank of the search results, provides better preview, and makes useful connections between documents.

Word cloud, also known as tag cloud, is a widely used visual representation for text visualization. Words are usually single words and their importance values are mapped to font sizes or font colors. Straightforward word clouds organize words lines by lines. This allows quick understanding of the poplar or important words. Since different font sizes are used to arrange words lines by lines, there are gaps between the lines. In addition, their typefaces are usually limited to the standard browser fonts. Wordle [125] was designed to address these problems. The Wordle layout algorithm allows word rotation in order to reduce gaps between words. Plenty of typefaces and color themes are included to enrich the visual experience of Wordle. ManiWordle [74] further allows users to adjust the typefaces, color themes, and word angles, not only for the layout as a whole, but also individual words. Context preserving dynamic word cloud [29] utilizes a force model to adjust word placement in order to reduce and even avoid overlap. Originated from word cloud, tree cloud [45] utilizes a tree structure to show the semantic proximity of the words. SparkClouds [81] combines sparklines with word cloud where a sparkline is shown right beneath each word to show its trend. The classic ThemeRiver [56] utilizes a river metaphor to convey the evolution of thematic contents. In the 2D plot, the horizontal direction represents the time and the vertical direction represents the strength of selected topics. Different colors are used to represent different topics. Built on the ThemeRiver metaphor, TIARA [143] is a text analytic system that shows the content evolution of multiple topics. Users are allowed to expand each topic to study its underlying keywords and evolution over time.

2.2.3 Image and Text Collections

Despite the abundant research on visualizing images and on visualizing texts, considering both images and texts in the same visualization has not been thoroughly investigated. Several techniques tried to reorganize texts and images of a document to fit into a small

screen display, such as paper-to-PDA [15] and SmartNails [9]. Document Card [115], on the other hand, visualizes a document with the extracted keywords and images. It adopts the idea of trumps game cards to provide a combined overview of an object. Each card consists of images and associated keywords which describe a document.

2.2.4 Our Solution

Most approaches for image-text collection visualization have fixed layouts for only one type of objects (images or texts). They provide limited interactions to organize that type of objects according to a selected one. Therefore, they are not able to capture the relationships among different types of objects. Even for the same type of objects, they cannot reveal the relationships between any two objects. Instead, only the relationships between the selected object and other displayed objects are illustrated. Although some works display both images and texts simultaneously, they focus on organizing them into a small screen, where no relationships among objects can be inferred. Our solution, iGraph, uses a force-directed graph layout algorithm [44] so that the positions of displayed objects are updated dynamically to indicate their relationships. Initially, a backbone graph is presented, where the representatives of images and texts are displayed. iGraph clusters the images to pick the representative ones, and chooses those keywords with the highest frequencies of appearance to be the representative texts. In this way, the backbone graph provides an overview of the entire collection and serves as the starting point for navigation. iGraph incorporates several functions to assist the navigation: a recommendation function to suggest similar objects; a node comparison function to present the similarities and differences of several objects in the context of their neighborhoods; a text outlier detection function to identify events of interest. Finally, a parallel implementation is integrated to speed up the computation and visualization performance.

2.3 Visualization of Eye-Tracking Data

Eye-tracking has become a widely used technique to analyze user behaviors in many disciplines. Rayner [100] synthesized over 100 years of eye-tracking research and conducted an excellent survey of eye-tracking applications in reading and other information processing tasks. Duchowski [36] presented a breadth-first survey of eye-tracking applications in the following domains: neuroscience, psychology, industrial engineering and human factors, marketing or advertising, and computer science. Recently, Blascheck et al. [11] presented a comprehensive state-of-the-art report on techniques for visualizing eye-tracking data. They classified the visualization techniques into different categories based on properties of eye-tracking data and properties of visualization techniques. Following their classification, we organize related work into two subsections: studying static stimuli and analyzing dynamic stimuli. In addition, several works that use eye-tracking analysis to evaluate visualization tools are presented as well.

2.3.1 Studying Static Stimuli

To visualize the spatiotemporal behaviors of eye movement data, one can use heat maps or gaze plots. But these visual representations suffer from high aggregation (heat maps) and overplotting (gaze plots). New visual mappings and representations are needed to assist with visual analytics of vast amounts of spatiotemporal eye gaze trajectories. Goldberg and Helfman [48] proposed an interface to identify scanning strategies by automatically aggregating groups of matching scanpaths. First, they converted each scanpath into a sequence of AOIs visited in order. Sequences of AOIs were concatenated into one sequence and plotted with a dotplot. Then they used linear repeated scanpaths to find matching sequences in the dotplot for clustering the scanpaths hierarchically. Tsang et al. [121] presented eSeeTrack, an eye-tracking visualization prototype to facilitate the exploration and comparison of sequential gaze orderings in a static or dynamic scene. Their work inte-

grates a timeline and a tree-structured representation to encode multiple aspects (duration, frequency, and fixation ordering) of eye-tracking data. Burch et al. [18] transformed eye movement data into a dynamic graph and achieved a fair tradeoff between aggregation and details. Their dynamic graph is a sequence of static graphs where nodes represent AOIs and directed edges show transitions between source and target AOIs. Burch et al. [17] designed AOI Rivers for investigating time-varying fixation frequencies, transitions between AOIs, and the sequential order of gaze visits to AOIs. Based on the ThemeRiver technique, they showed the trajectory data as time-varying river-like structures enhanced by influents, effluents, and AOIs transitions, similar to Sankey diagrams.

2.3.2 Analyzing Dynamic Stimuli

Compared with static stimuli, analyzing eye-tracking data from dynamic stimuli poses additional challenges as the stimuli themselves are moving or changing over time. Kurzhals and Weiskopf [77] presented a visual analytics approach for analyzing eye movement data recorded for dynamic stimuli such as video or animated graphics. They utilized a space-time cube visualization along with clustering to analyze the dynamic stimuli and associated eye gaze in a static 3D representation. Kurzhals et al. [78] designed ISeeCube which combines methods for the spatiotemporal analysis of gaze data recorded from unlabeled videos and the possibility to annotate and investigate dynamic AOIs. Kurzhals and Weiskopf [76] introduced AOI transition trees to investigate eye-tracking data of multiple participants recorded from video stimuli. Their visualization employs icicle plots to represent AOI sequences modeled as trees. This approach scales well to large numbers of participants and allows easy detection of common transition patterns. Kurzhals et al. [79] designed gaze stripes for displaying eye-tracking data from multiple participants. Similar to thumbnails for images, their technique directly uses the image data around the gaze points and arranges a sequence of such gaze point images along a horizontal timeline to form gaze stripes. The visualization shows the spatiotemporal data of the gaze points in the context

of the underlying image or video stimulus without occlusion.

2.3.3 Evaluating Visualization Research

Beyond analyzing eye-tracking data, eye movement analysis has gained its popularity as a tool for evaluating visualization research. Andrienko et al. [4] proposed a visual analytics methodology originated from analysis of geographic data for analyzing large amounts of eye-tracking data. They focused on deriving common task solution strategies for a given static stimulus shown to participants. Their work presents a systematic evaluation of movement analysis methods for the applicability of eye-tracking data and provides the guidelines for choosing appropriate methods given the analysis goals. Blascheck et al. [12] presented a visual analytics approach for an integrated analysis of multiple concurrent evaluation procedures such as measures of task performance, think-aloud protocols, analysis of interaction logs, and eye-tracking. An efficient exploratory search and reasoning process is supported through automatic pattern finding to derive common eye-interaction-thinking patterns between participants.

2.3.4 Our Solution

Unlike the previous approaches, which deal with image and video data, our ETGraph focuses on studying reading behaviors through analyzing the scanpaths of dozens of participants reading a book. Due to the distinctive characteristics of this kind of data, ETGraph differs from the previous approaches in two aspects. First, different images and videos may have quite different structures in terms of their contents, but texts on different pages share a similar pattern since they are usually read in similar orders (i.e., left to right and top to bottom). This common pattern allows a more meaningful comparison between the scanpaths of different participants and on different pages. ETGraph studies the reading behaviors in four different settings: single participant single page (SPSP), single participant multiple pages (SPMP), multiple participants single page (MPSP), and multiple partici-

pants multiple pages (MPMP). By comparing multiple participants, the different reading styles can be extracted (e.g., fast readers and slow readers). By comparing the reading behaviors on different pages, we can learn whether or not a participant has consistent reading patterns, and therefore, better recognize the outlier behaviors. Second, AOIs are often used in studying eye-tracking data on images and videos to group the fixations and simplify the eye movement patterns. In those data, AOIs are usually meaningful objects, such as an apple. However, they are not available in texts. ETGraph utilizes the mean-shift algorithm to group fixations into clusters and constructs a transition graph based on these clusters. This simplifies the eye movement patterns to a controllable level, and produces a concise graph representation.

CHAPTER 3

TRANSGRAPH FOR VISUALIZING HIERARCHICAL TRANSITION RELATIONSHIPS OF TIME-VARYING VOLUMETRIC DATA

3.1 Overview

In this chapter, we present TransGraph that encodes the transition relationships extracted from a 4D spatiotemporal data set to a 2D graph view. This is achieved by first organizing a given time-varying data set into a hierarchy of states. By deriving transition probabilities among states, we then construct a global map that captures the essential transition relationships in the time-varying data. The resulting TransGraph not only provides a visual mapping that abstracts data evolution over time in different levels of detail, but also serves as a navigation tool that guides data exploration and tracking. The users interact with the TransGraph and make connection to the volumetric data through brushing and linking. A set of intuitive queries is provided to enable knowledge extraction from the underlying time-varying data.

3.2 Hierarchical State Transition

Figure 3.1 illustrates the input and output for each step of our approach to derive hierarchical state transition relationships. We first partition the volume data at each time step into blocks and identify representative blocks in a hierarchical manner. Representative blocks are used to construct a distance matrix to expedite the subsequent clustering. Next, we define states from data blocks and cluster states into a hierarchy. Finally, we derive

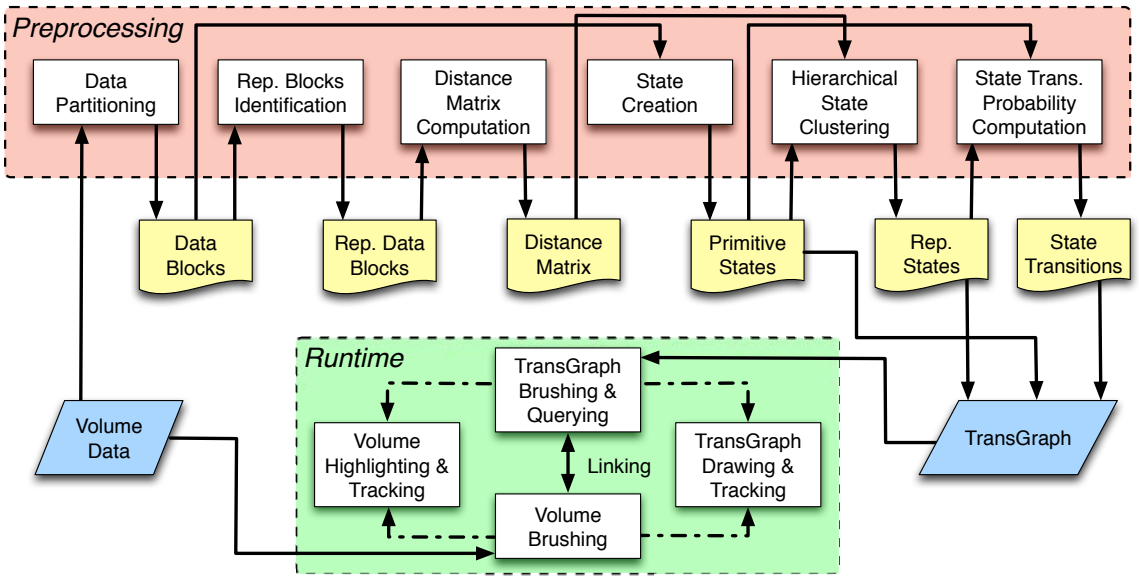


Figure 3.1. The overview of our approach. We derive hierarchical state transition relationships from a given time-varying data set. Such relationships are visualized using a graph-based representation and integrated with volume view for visual data exploration.

directional transition probabilities among states to construct a hierarchical state transition graph. In the following, we describe each step in detail.

3.2.1 Data Blocks and Distance Measure

Given a time-varying volumetric data set, we perform uniform subdivision of each time step separately into a list of data blocks with an equal size. We will use the histograms of blocks to compute their distances, define states based on blocks, and cluster states hierarchically to extract their transition relationships. The size of block is chosen in a way so that each block is large enough for meaningful histogram computation. For instance, assuming a 256-level histogram, the block should contain at least a few thousands of voxels for stable computation. On the other hand, we should generate a sufficient number of blocks for effective hierarchical state clustering. At least a few hundreds of blocks for each time step

is appropriate. In practice, we set the block sizes around 16^3 to 32^3 for the time-varying data sets we experiment with (see Table 3.1). For a data set with larger spatial and temporal dimensions, the block size should increase proportionally.

Given two data blocks b_i and b_j, we can compute their distance in many different ways, for example, using Kullback-Leibler divergence (KLD) for distribution distance computation [128], and defining features and extracting them explicitly for comparison. Instead, we leverage the Jensen-Shannon divergence (JSD) to compute the distance between their probability distributions P_i and P_j. In probability theory and statistics, the JSD is a popular method of measuring the similarity between two probability distributions. As a symmetrized and smoothed version of the KLD, the JSD has its minimum of 0.0 when the two distributions are identical, and its maximum of 1.0 when they are least similar. All these properties nicely fit our requirement for distance measuring. Specifically, we define

$$d_{\mathrm{JS}}(P_i, P_j) = \big(d_{\mathrm{KL}}(P_i||\hat{P}) + d_{\mathrm{KL}}(P_j||\hat{P})\big)/2, \tag{3.1}$$

where $\hat{P}$ is the average of the two distributions, $\hat{P} = (P_i + P_j)/2$, and d_{KL} is the KLD, i.e.,

$$d_{\mathrm{KL}}(P_i||P_j) = \sum_{k=1}^{m} p_i(k) \log \frac{p_i(k)}{p_j(k)}. \tag{3.2}$$

In the discrete sense, we use a histogram to approximate the underlying probability distribution. Therefore, in Equation 3.2, m denotes the total number of bins in the histogram and $p_i(k)$ and $p_j(k)$ are the probabilities (normalized frequencies) for b_i and b_j corresponding to the kth bin. Given a data block, the histogram could be constructed simply as a one-dimensional histogram which records the distribution of the original scalar values, or as a multi-dimensional histogram which records the distribution of a combination of features. Since typical scientific data imply a continuous model, continuous histograms proposed by Bachthaler and Weiskopf [6, 7] can be used to build a better basis for the following computation and analysis. Unlike discrete histograms, continuous histograms are structurally

Algorithm 1 REPBLOCKSIDENTIFICATION($|w|$: int; ε: float)

```
for each pass p do
    for each sliding time window w do
        if |w| = total # of time steps then
            b_w ← # blocks remaining in w
            b_max ← b_w
        else
            b_w ← # blocks remaining in w
            b_max ← # blocks in the first time step of w
        end if
        for b_i ← b_0 to b_max − 1 do
            if b_i has not been clustered then
                for b_j ← b_i to b_w − 1 do
                    if b_j has not been clustered then
                        if d_JS(b_i, b_j) ≤ ε then
                            Cluster b_i and b_j into the same group
                        end if
                    end if
                end for
            end if
        end for
    end for
    for each cluster c identified at pass p do
        Identify the rep. block as the one that has the smallest average distance to the rest of blocks in c
    end for
    Increase the size of sliding time window |w|
    Increase the distance threshold ε
    Gather the rep. blocks identified at pass p as the input for the next pass
end for
Compute the distance matrix M for all rep. blocks identified in the last pass
```

independent of the resolution of a data set and could avoid misleading structures which are not part of the data but due to the low sampling resolution.

3.2.2 Representative Blocks and Distance Matrix

From partitioned blocks, we identify representative blocks and compute a distance matrix recording the distance between any two of them. Later on, for any two blocks in the volume, we will simply look up the distance between their representative blocks as the approximation instead of the actual distance computation. Since the number of representative blocks derived will be substantially less than the number of initial blocks acquired from volume partitioning, using such a small-size matrix for distance lookup will effectively reduce the cost of distance computation in the following hierarchical state classification. For

a large input data set, computing the representatives from all blocks across all time steps would be very time consuming due to the large number of blocks considered simultaneously. We instead take a multi-pass process to efficiently identify representative blocks.

Our solution adopts a moving time window technique that considers a subset of data blocks at a time. Algorithm 1 summarizes our solution in pseudocode. In the first pass, the size of the time window is relatively small. We start b_i from the first block b_0 in the first time step of the initial window w_0 and cluster b_i with other blocks b_j in w_0 if their pairwise distance $d_{\mathrm{JS}}(b_i, b_j)$ (Equation 3.1) is less than a given distance threshold ε. Then, for the remaining blocks that have not been clustered, we locate the first block in the first time step of w_0 as b_i and perform the same clustering process again for these remaining blocks. This process for w_0 stops until we have attempted to cluster all its blocks where b_i loops through all the blocks in the first time step of w_0. Note that at this moment, it is likely that some blocks in w_0 might not be clustered. Next, we slide the window w_0 one time step further to create the window w_1 and perform the same clustering process for all blocks in w_1 that have not been previously clustered. This entire pass stops after the moving time window has swiped through the last time step. For each of the cluster we identify in the pass, we define its *representative block* as the one that has the smallest average distance to the rest of blocks in the cluster.

In the following passes, we repeat the same process as described above. The main differences are: (1) the size of time window will increase in each new pass, so does the value for the distance threshold ε; and (2) the input to the current pass is all representative blocks identified from the previous pass. In the last pass, the time window spans the entire time sequence. Only the representative blocks identified in this final pass will be used to build the distance matrix $\mathbf{M}$. We point out that it could be very time-consuming to cluster blocks and identify representatives even though a multi-pass (essentially hierarchical) process is taken. In our work, we speed up the bottleneck computation (i.e., JSD calculation) using GPU with a CUDA implementation. The performance and speedup we achieve are

Algorithm 2 HIERARCHICALSTATECLUSTERING($|w|$: int; ε: float)

```
for each level l do
  for each sliding time window w do
    if |w| = total # of time steps then
      s_w ← # states remaining in w;  s_max ← s_w
    else
      s_w ← # states remaining in w;  s_max ← # states corresponding to the first time step of w
    end if
    for s_i ← s_0 to s_max − 1 do
      if s_i has not been clustered then
        for s_j ← s_i to s_w − 1 do
          if s_j has not been clustered then
            if s_i and s_j are overlap or neighboring and are in the same or neighboring time steps then
              if p is the first pass then
                d ← the average of the distances between their corresponding blocks in s_i and s_j
              else
                d ← the distance between their primitive states in s_i and s_j
              end if
              if d ≤ ε then
                Cluster s_i and s_j into the same group
              end if
            end if
          end if
        end for
      end if
    end for
  end for
  for each cluster c identified at level l do
    Identify the rep. state as the one that has the smallest average distance to the rest of states in c
  end for
  Increase the size of sliding time window |w|;  Increase the distance threshold ε
  Compute trans. probabilities p_{s_i→s_j} and p_{s_j→s_i} among all states at level l
  Gather the rep. states identified at level l as the input for the next level
end for
```

reported in Section 3.4.1.

3.2.3 States and Transition Probabilities

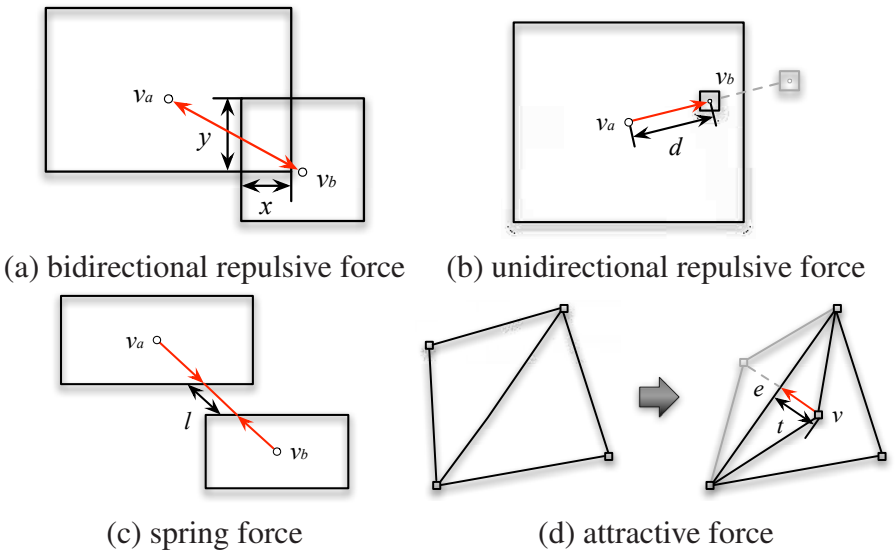
In this work, we define *states* and their *transitions* as follows:

- A *state* is a configuration of neighboring spatiotemporal blocks characterized by their value distributions. A *primitive state* is a configuration of neighboring spatial blocks centered at each block at a single time step. In hierarchical state clustering, we select a *representative state* from a group of states. The group is formed by

merging similar states that are spatially overlapping or neighboring and are in the same or neighboring time steps.

- A *transition* occurs between two states, from one state at time step t to another state at time step $t+1$, if and only if their corresponding central blocks are the same in the spatial domain. Given two states s_i and s_j at the same clustering level, the directional *transition probability* $p_{s_i \to s_j}$ is the ratio between the total number of transitions from s_i to s_j and the total number of transitions from s_i to all states (including itself). This definition indicates how frequently a state is transferred to another state.

To create primitive states, we use an arrangement of $3 \times 3 \times 3$ blocks for spatial configuration. The number of primitive states is the same as the number of data blocks. As shown in Algorithm 2, for all primitive states in the data set, we perform hierarchical clustering and select a representative state from each cluster. We use a method similar to region growing in our clustering mainly due to the efficiency concern. The spatial and temporal constraints are to ensure that clustered states are space-time continuous which makes the subsequent tracking process meaningful. For distance computation between two primitive states s_i and s_j, if any of them is along the volume boundary, special care is taken so that their partial block correspondence is used to derive the distance. Unlike state clustering, we only consider the central blocks in primitive states for transition probability computation. If both states s_i and s_j are clusters of states rather than primitive states, then the transition probability is computed based on the total number of transitions from any primitive state in s_i to any primitive state in s_j. As needed, we may consider the transition probability for all states either in each individual time step (local normalization) or across all time steps (global normalization). In the first case, it would lead to a time-varying graph highlighting the dynamics transitional relationship changes over time. In the second case, it would lead to a static graph summarizing all transitions across all time steps.



(a) bidirectional repulsive force (b) unidirectional repulsive force

(c) spring force (d) attractive force

Figure 3.2. The four different forces we consider for the TransGraph layout adjustment during the level-of-detail exploration. (a) the bidirectional repulsive force pulls apart two nodes that overlap each other. (b) the unidirectional repulsive force pushes a node away from an expanded node if it is inside the expanded node. (c) the spring force is introduced to keep the balance with respect to the two repulsive forces. (d) the attractive force handles the topology change of the underlying triangle mesh, i.e., when a triangle is flipped.

3.3 TransGraph

We create a hierarchical state transition graph, i.e., TransGraph, to record the transition relationships among states at various levels of detail. In this way, we convert a 4D space-time volume data set to a graph which can be visualized in 2D. In the TransGraph, a node represents a state (leaf) or a cluster of states (non-leaf) and the edge between two nodes represents their transition probabilities. For simplicity, we draw a single undirected edge instead of double directed edges in the TransGraph. The edge weight is the summation of the two directional transition probabilities associated with the two incident nodes. Note that the summed edge weight is used only for the purpose of graph drawing while the underlying graph representation is still a directed graph. This allows the user to per-

form separate transition queries on incoming or outgoing transition probabilities. In the following, we describe how we draw the TransGraph to account for the level-of-detail adjustment, and how we brush, query, and track in the graph view in conjunction with the volume view to enable visual data analysis and understanding.

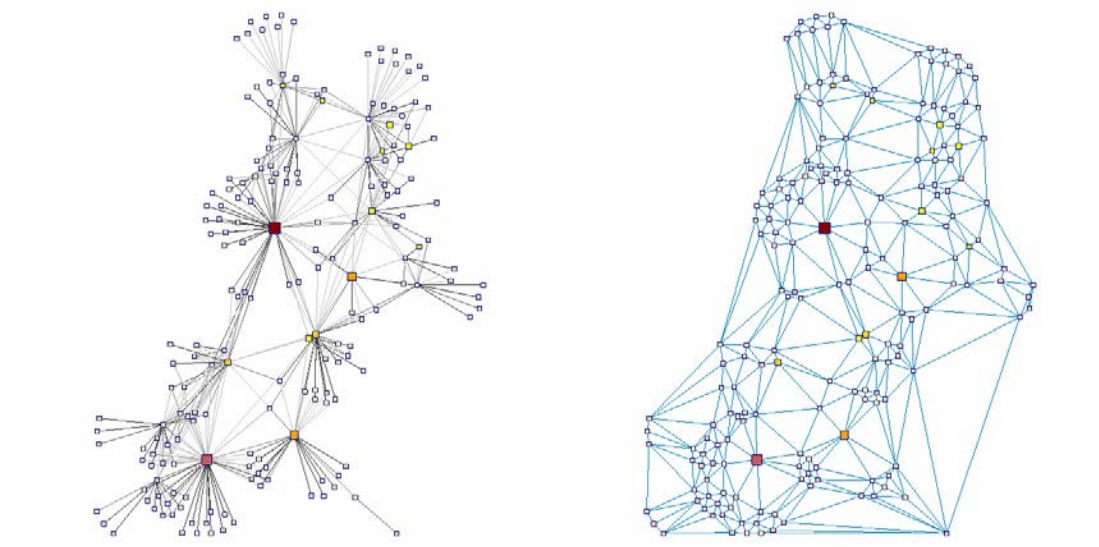
3.3.1 TransGraph Drawing

The TransGraph has two places that introduce dynamics into graph drawing. First, the nodes and edges in the TransGraph change for every time step. Second, the level-of-detail exploration leads to graph changes during the interaction, which requires adjusting node positions to reduce overlap or occlusion. A good layout for TransGraph drawing should maintain a good balance between preserving the mental map (i.e., the abstract structural information a user forms by looking at the graph layout) and revealing the dynamics.

In this work, we adopt the idea of the supergraph [31] by computing a global layout which induces a layout for each time step. This allows us to observe all states and their transitions in a single visualization. In the meanwhile, each graph in the time sequence becomes a subset of the supergraph and is visualized sequentially. To support effective graph viewing, we propose a constrained layout adjustment algorithm to maintain coherent update and minimize node overlap during interactive level-of-detail exploration.

We employ the Fruchterman-Reingold algorithm [44], a popular force-directed layout algorithm to all levels of TransGraph drawing. This algorithm ensures that topologically near nodes are placed near to each other and topologically far nodes are placed far from each other. Furthermore, unlike other algorithms such as the Kamada-Kawai algorithm [70], nodes will not get too close to each other in the drawing. Therefore, the drawing area is effectively utilized as overlap or occlusion among nodes is reduced.

During the graph exploration, the user selects one or multiple nodes for examining their higher levels of detail. As such, selected nodes would be expanded which demands layout adjustment. We generate the initial layout for the coarsest level of the TransGraph. To

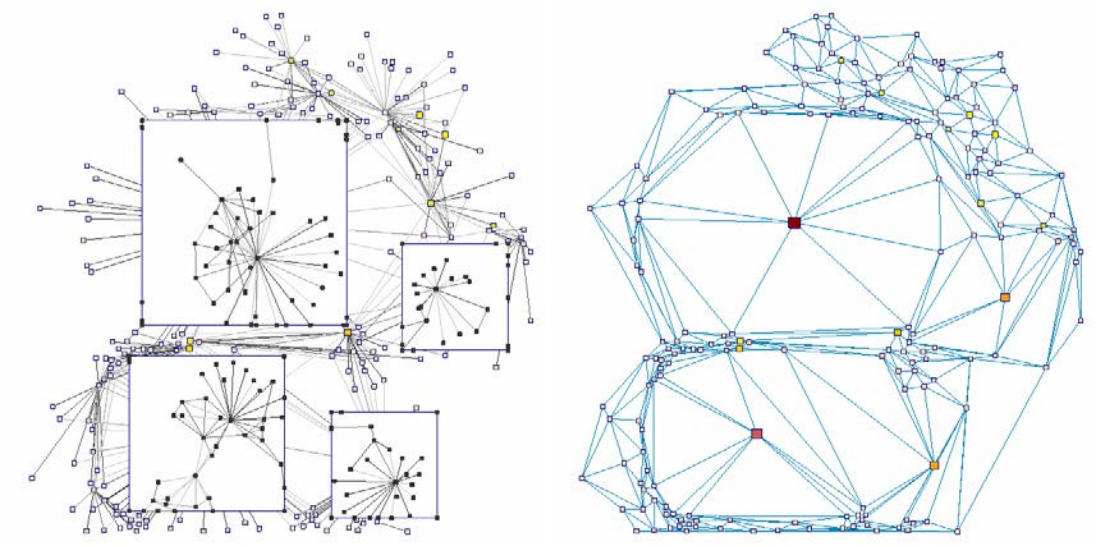


(a) initial layout (b) triangle mesh of (a)

Figure 3.3. (a) the initial layout at the coarsest level is produced using the Fruchterman-Reingold algorithm. The size of each node in the graph is proportional to the number of children within. (b) the triangle mesh produced from the initial node positions.

preserve the relative positions of nodes in the initial layout for coherent update, we apply the triangulation scheme proposed by Shewchuk [108] to the initial graph and use the result of the triangulation to perform constrained layout adjustment when nodes are expanded for further examination. When such a node is expanded, its initial size is proportional to the number of children in its next level of detail. All nodes expanded are assigned the same scaling factor. Similar to the work of dynamic word cloud visualization by Cui et al. [29], we consider the following forces to reposition the nodes to reduce their overlap while maintaining the topology of the coarsest level of the TransGraph.

- **Bidirectional repulsive force:** This force pushes away two nodes v_a and v_b from each other and is effective if and only if v_a and v_b overlap each other. The bidirectional repulsive force is defined as $f_1(v_a, v_b) = k_1 \times \min(x, y)$, where k_1 is a given weight and x and y are the width and height of the overlapping region as shown in



(a) final layout (b) triangle mesh of (a)

Figure 3.4. (a) the layout after four nodes are selected and expanded for detail examination of Figure 3.3. Four different forces (Figure 3.2) are used in the adjustment. (b) the underlying triangle mesh is used to maintain the topology of the graph during layout adjustment.

Figure 3.2 (a). This force is applied to every pair of nodes in the TransGraph.

- **Unidirectional repulsive force:** This force pushes away a node v_b without detail shown from a node v_a with detail shown and is effective if and only if v_b is inside v_a. The unidirectional repulsive force is defined as $f_2(v_a, v_b) = k_2/d$, where k_2 is a given weight and d is the distance between the centers of v_a and v_b, as shown in Figure 3.2 (b). If d is close to zero, then an upper-bound force $f_{2\max}$ is used instead which guarantees that v_b will be moved outside of v_a.

- **Spring force:** This force is used to balance the graph by offsetting the two repulsive forces introduced. Given two nodes v_a and v_b, the spring force is defined as $f_3(v_a, v_b) = w_a \times w_b \times l$, where w_a and w_b are the weights of v_a and v_b respectively, and l is the length of the edge connecting the centers of v_a and v_b that lies outside

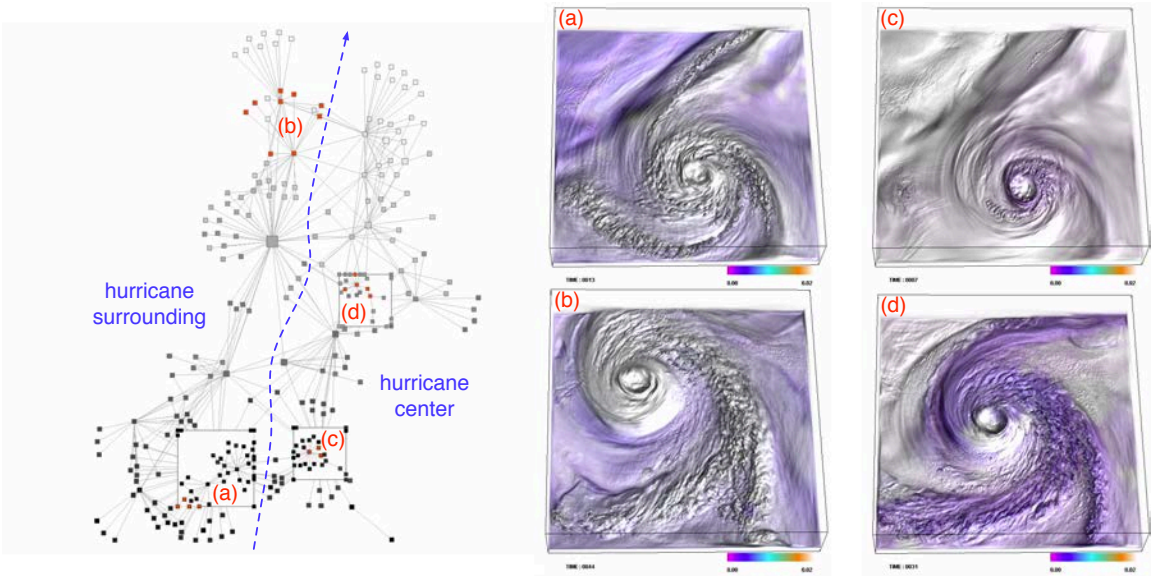


Figure 3.5. Left: the TransGraph of the hurricane data set. Right: (a) to (d) are the corresponding volume highlighting indicating the red nodes selected in the TransGraph at time steps 13, 44, 7, and 31, respectively. The spatiotemporal regions corresponding to the hurricane's surrounding, i.e., (a) and (b), and center, i.e., (c) and (d), lie on the two clear-cut parts of the TransGraph, respectively.

of their boundaries as shown in Figure 3.2 (c). We compute w_a and w_b based on the number of children in v_a and v_b, respectively. The larger the number, the higher the weight. This force is applied to every pair of nodes in the TransGraph.

- **Attractive force:** This force is used to maintain the underlying triangle mesh we construct for the coarsest level of the TransGraph. During the layout adjustment, if a mesh triangle is flipped, as shown in Figure 3.2 (d), then the topology of the triangle mesh changes. Our goal is to maintain a stable update of the graph by introducing an attractive force to flip the triangle back. The attractive force is define as $f_4(v_a) = k_4 \times t$, where k_4 is a given weight and t is the distance from node v to edge e. We also consider virtual triangle edges connecting extreme nodes in the graph to the four corners of the drawing area. This is to ensure that all graph nodes do not go

out of bound.

Figures 3.3 and 3.4 show an example of layout adjustment during the level-of-detail exploration. As we can see, the expanded nodes expel other nodes outside of their regions while the global structure of the TransGraph is still preserved. In this example, we give a fairly large size for drawing each of the higher levels-of-detail to show an extreme case. In practice, the size for drawing a higher level-of-detail could be smaller as long as the nodes within can be clearly seen without much clutter, as shown in Figure 3.5. Our layout adjustment strategy is applied recursively to different hierarchical levels in the same manner.

3.3.2 TransGraph Query

The TransGraph stores rich information about the nodes (states) and edges (transitions) over time. To make the best use of the TransGraph, we propose a set of intuitive queries to enable knowledge extraction from the underlying time-varying data.

- **State query:** We derive how active a state is by computing the number of time steps the state remains active. By summarizing its connecting states and their probabilities, we can tell how active a node is in terms of transferring to other states or how inactive it is in terms of staying in the same state. From the hierarchy of the TransGraph, we also derive the active volume extent a state corresponds to by summarizing the information of its descendants. These queries would allow the user to automatically locate important or dominant states with respect to the spatial extent, transition relationship, or time span. Such queries shares some similarity with recognizing node centrality in a graph, while our query provides quantified search that is unavailable by simply observing the TransGraph.

- **Transition query:** From the edges of the TransGraph, we derive which transitions are the most dominant ones over time and what are the corresponding states. We

can tell which transitions are the most or least balanced ones by comparing the two directional transition probabilities associated with each edge. From the hierarchy of the TransGraph, we also perform transition queries recursively for finer-grained examination as needed.

- **Time step query:** We identify important time steps in the time sequence by summarizing and ranking time steps based on the number of states active, the number of transitions active, or the number of states involved in self- or non-self transition only. Such temporal queries can be used to automatically highlight important or interesting time steps where state transitions are much more frequent or widespread than other time steps.

3.3.3 Brushing and Tracking

We dynamically link together the two views of the time-varying data, namely, the volume view and the graph view. The user interacts with the data in one view and the result is automatically reflected in the other view. This dual-domain interaction is quite standard for the visualization of data from different perspectives. The brushing and linking between the two views compliments each other. The volume view is intuitive for understanding while data occlusion is inevitable and data selection could be tricky. On the contrary, the graph view is an abstract representation while occlusion-free data selection is fairly straightforward. Combining both views together allows easy interaction and comprehensive understanding of the time-varying data.

The user selects data blocks either directly from the volume (by bounding the ranges in the x, y, and z directions) or from the TransGraph (by direct clicking or range selecting the states of interest). For block selection from the volume, the user can clip the volume or adjust the transfer function to better identify interesting 3D data blocks that are occluded. We allow tracking of these data blocks over the space and time by highlighting the influenced region in the volume and the corresponding states in the TransGraph.

TABLE 3.1

THE TIME RESULTS AND PARAMETERS OF DATA SIZES AND BLOCK SETTINGS USED IN HISTOGRAM AND JSD COMPUTATION

data set	variable	volume dimension	block size	data read	hist comp	# bins	JSD (CPU)	JSD (GPU)
climate	salinity	$360 \times 66 \times 27 \times 120$	$15 \times 11 \times 9$	0.23	0.60	64	58	19.2
			$15 \times 6 \times 9$	0.26	0.73	64	155	7.2
			$30 \times 11 \times 9$	0.24	0.99	128	41	4.8
combustion	YOH	$480 \times 720 \times 120 \times 122$	$30 \times 30 \times 30$	167	78	256	12384	31
earthquake	amplitude	$256 \times 256 \times 96 \times 599$	$16 \times 16 \times 16$	125	53	256	64883	95
			$16 \times 16 \times 8$	119	23	128	154212	233
			$32 \times 32 \times 16$	119	41	512	3982	17.9
hurricane	vapor	$500 \times 500 \times 100 \times 48$	$20 \times 20 \times 20$	34	18.2	256	23058	16.3
ionization	gas temp.	$600 \times 248 \times 248 \times 200$	$30 \times 31 \times 31$	164	114	256	73156	156

We present two different ways of tracking: *static tracking* and *dynamic tracking*. Static tracking is to fix the spatial blocks while tracking their state changes over time. Tracking over the TransGraph only is straightforward as all transition information is recorded beforehand. Unlike static tracking, dynamic tracking also conveys the impression of data transition in the volume while tracking the corresponding state changes over time. To enable volumetric tracking, we modulate the color saturation of data blocks based on their transition probabilities. For simplicity, we assume a constant propagation speed of the data. The user has the flexibility to adjust the speed at runtime. Adding the propagation speed enables us to fine tune data transition at the voxel-wise level, going beyond the block-wise granularity for state transition computation. In addition, we consider the distances of a voxel to the centers of initial blocks selected for tracking to further adjust the color saturation of the voxel (refer to Section 3.4.4 for more detail). This allows us to differentiate voxels within a block with different color saturations even though the transition probability stays in the block-wise level. In practice, we perform all these color modulations in the fragment program to ensure the interactivity and realtime tracking.

3.4 Results and Discussion

In this section, we first report the data sets we experimented with and their timing performance. Then, we use different data sets to demonstrate how brushing and linking, query and tracking are performed between the volume view and the graph view. We also describe what are the knowledge and benefit gained from such a dual-domain interaction. In addition, we show the TransGraph layouts with different parameter settings to study their influence on the graph layout.

TABLE 3.2

THE TIMING RESULTS AND PARAMETERS USED FOR HIERARCHICAL CLUSTERING

data set	two-pass (CPU)	two-pass time win	clustering threshold ε	three-level time win	three-level threshold ε
climate	3, 120	0.05, 0.1	52	7, 120, 120	0.15, 0.3, 0.45
	6, 120	0.1, 0.15	118	6, 120, 120	0.275, 0.425, 0.575
	5, 120	0.1, 0.15	7.6	3, 120, 120	0.275, 0.375, 0.45
combustion	4, 122	0.05, 0.1	628	5, 122, 122	0.2, 0.35, 0.4
earthquake	3, 599	0.25, 0.3	1060	5, 599, 599	0.2, 0.4, 0.6
	4, 599	0.075, 0.085	17059	2, 599, 599	0.2, 0.3, 0.4
	4, 599	0.075, 0.09	222	3, 599, 599	0.2, 0.25, 0.4
hurricane	4, 48	0.05, 0.15	5069	7, 48, 48	0.175, 0.3, 0.45
ionization	4, 200	0.0135, 0.02	1027	5, 200, 200	0.2, 0.45, 0.7

3.4.1 Data Sets and Timing Performance

We experimented our approach with five time-varying volumetric data sets. All these data sets are produced from scientific simulations. From top to bottom in Table 3.1, the data sets are from a simulation of the equatorial upper-ocean, a simulation of turbulent combustion, a simulation of Northridge earthquake in 1994, a simulation of Hurricane Isabel in 2003, and a simulation of ionization front instabilities. We picked one variable from each data set in our study.

The timing was collected on a PC with an Intel 2.4GHz CPU and an nVidia GeForce GTX 465 GPU. There are three major tasks in our computation: histogram computation,

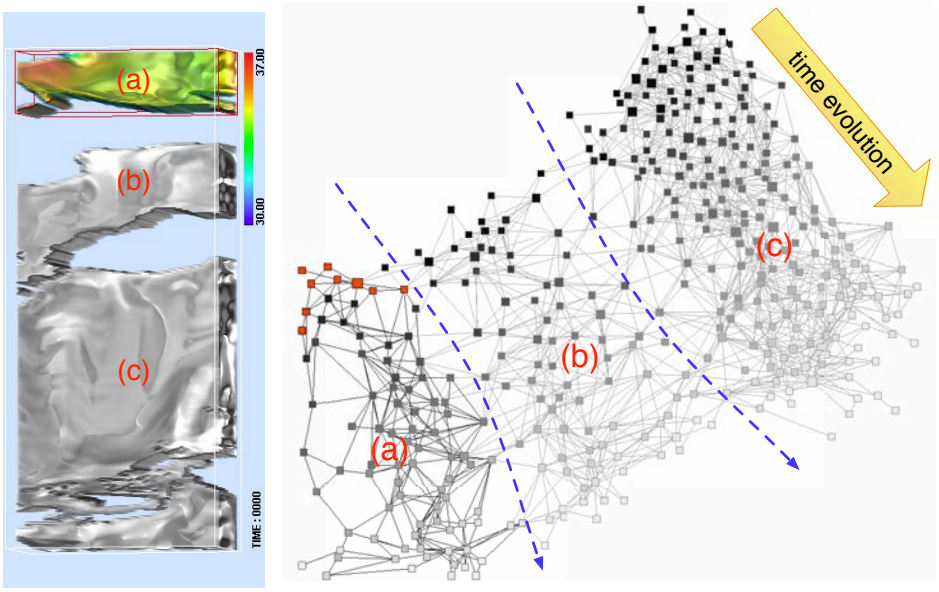


Figure 3.6. Top: a spatial region at the first time step of the climate data set is selected and highlighted in its original color saturation. Bottom: the corresponding nodes are highlighted in red in the TransGraph with the subsequent transitions (edges) of this region highlighted in black. The rest of nodes are displayed with decreasing shading as the time step increases to indicate the direction of time evolution in the graph. Three well-isolated spatial regions marked with (a), (b), and (c) form three clear-cut clusters in the TransGraph.

JSD computation, and hierarchical clustering. Tables 3.1 and 3.2 show the timing results. Histogram computation is to calculate the histograms for all blocks in the data. JSD computation is to compute the distance among blocks to identify representative blocks, and to compute the distance matrix for distance lookup in the subsequent clustering. Hierarchical clustering is to cluster states into a hierarchy for building the TransGraph. For JSD computation, we opted for two passes for simplicity. For hierarchical clustering, we opted for three levels (not including the leaf level consisting of all primitive states). After the initial clustering, we did not impose the time window anymore so that more neighboring states can be merged together into clusters. Among the three tasks, the bottleneck is JSD computation because of the large numbers of log operations (Equation 3.2) involved. As

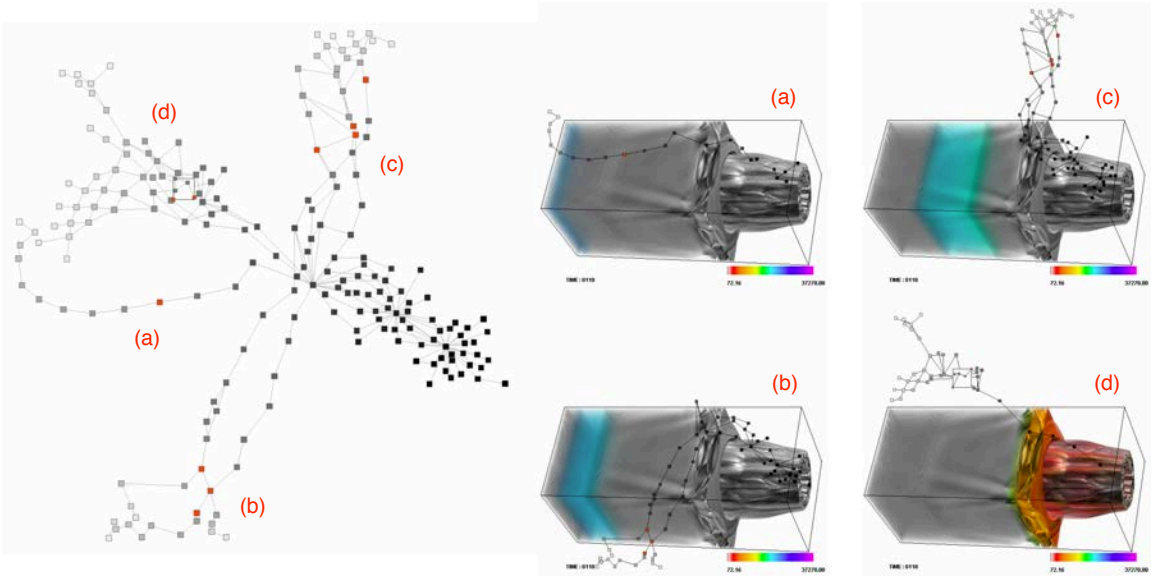
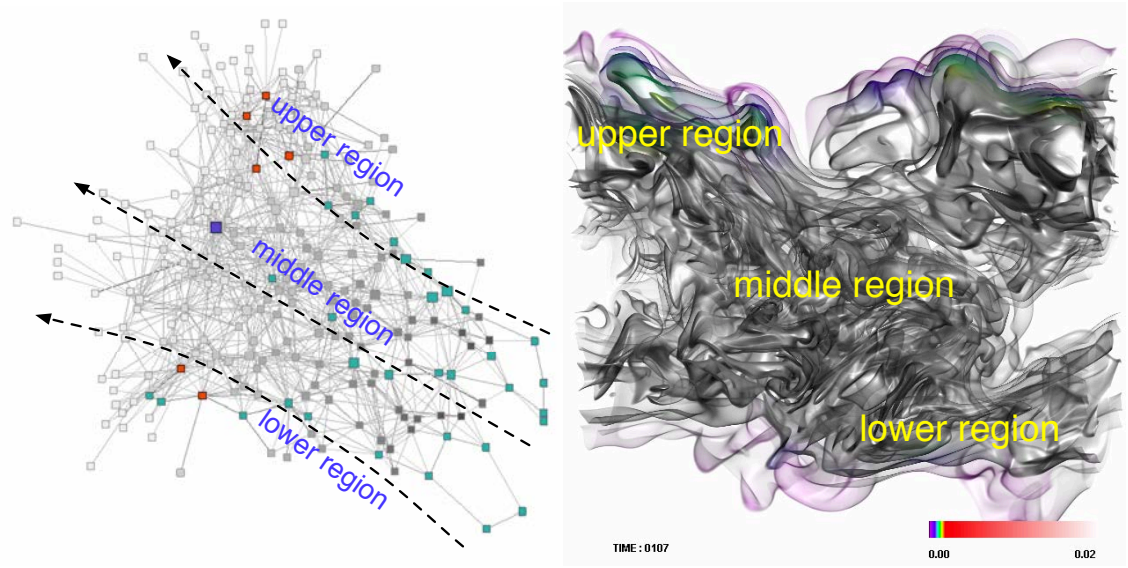


Figure 3.7. Left: the TransGraph of the ionization data set. Right: (a) to (d) are the corresponding volume highlighting indicating the respective red nodes selected in the TransGraph at time step 118. Each spatial region corresponding to the states in red is tracked over time and the state evolution is extracted from the full graph. These separate spatial regions highlighted correspond to different branches in the TransGraph.

such, we also implemented a GPU version to speed it up by calculating all pairs of JSDs for a group of blocks in parallel. All other tasks were computed in the CPU.

For the TransGraph drawing, the initial graph layout and triangulation only needs to be computed once for each data set, which can be completed within a few seconds. At runtime, we allow the user to interactively explore the TransGraph at various levels of detail and perform layout adjustment in real time with up to thousands of nodes. Beyond that, the performance may not be interactive and the drawing area may not be sufficient. Therefore, in our current implementation, we do not allow the user to explore too deep down the TransGraph, such as the leaf level.

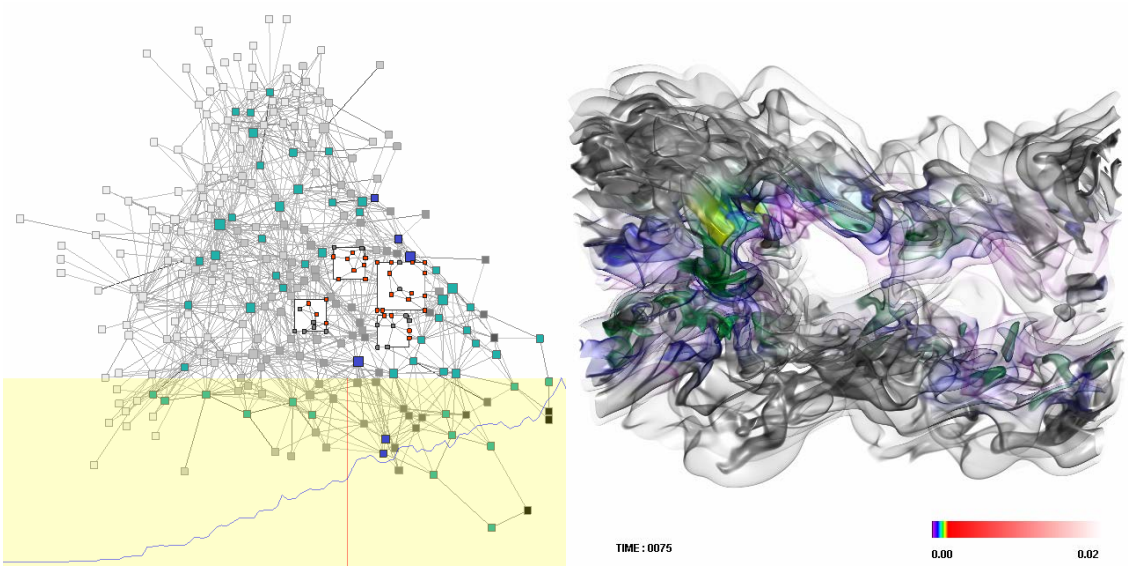


(a) (b)

Figure 3.8. The TransGraph of the combustion data set with state and time step queries. (a) the query of the first-level states having their outgoing transition frequency larger than 0.108. These nodes are in red, blue, and green. (b) the volume highlighting corresponding to the red nodes brushed at time step 107. The node that is not selected is in blue and the rest in other time steps are in green.

3.4.2 Brushing and Linking

The brushing and linking between the volume view and the graph view provides the user with an intuitive way to understand both data and make connections. For example, Figure 3.6 shows an example where we brush a spatial region corresponding to the equatorial Indian Ocean of the climate data set. In the TransGraph, the edges (transitions) associated across all time steps are highlighted in black and the nodes (states) corresponding to the first time step are highlighted in red. The rest of nodes are displayed with decreasing shading as the time step increases. In this way, the direction of time evolution is easily discernible from the graph. As we can see, the well-isolated spatial region we brush forms a clear-cut cluster in the TransGraph. As a matter of fact, three distinct node clusters are



(a) (b)

Figure 3.9. The TransGraph of the combustion data set with state and time step queries. (a) the query of the second-level states having their incoming transition frequency larger than 0.057. (b) the volume highlighting corresponding to the red nodes brushed at time step 75. The time step query, displayed in the lower part in (a), shows the trend of the increasing number of active nodes in the second level over time.

fairly well separated in the TransGraph which correspond to the three ocean regions separated by the continents (i.e., void regions in the volume view). Thanks to the hierarchical state clustering which imposes that the clustered states must be spatiotemporal neighbors in any level of clustering, we can preserve spatiotemporal data neighborhood relationships in the TransGraph. The well-organized nodes in the TransGraph make it convenient for the user to judge the direction of time evolution and the correspondence between volume regions and graph nodes.

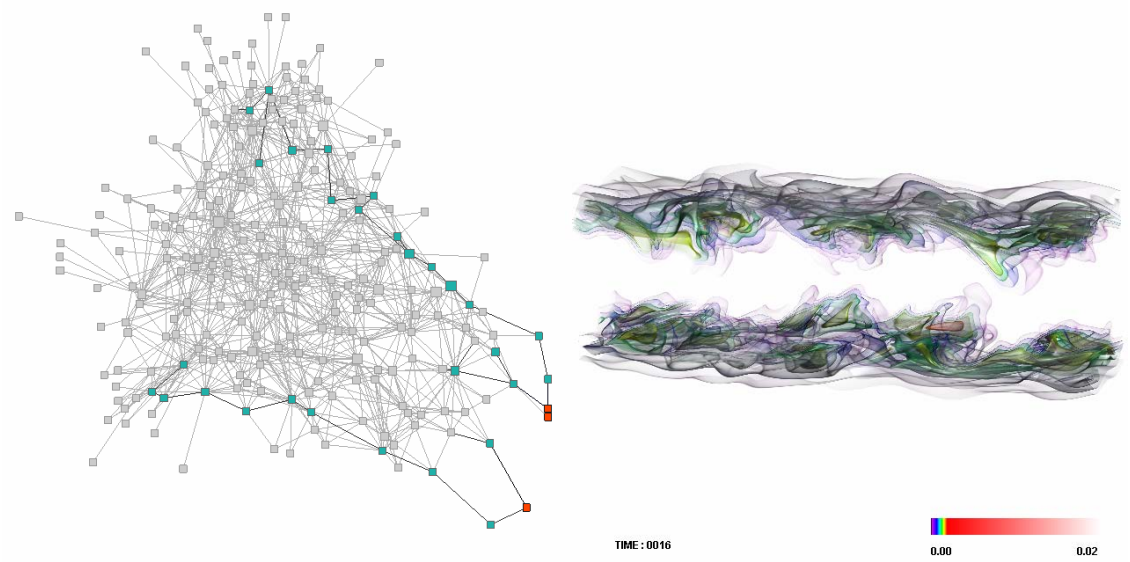
From Figure 3.5, we can observe that the TransGraph of the hurricane data set can be separated into two parts as indicted by the dashed line. These two parts of nodes actually correspond to the spatiotemporal regions of the hurricane's center and surrounding, respectively. This is verified by the rendering of four time steps corresponding to the

nodes (states) selected in the TransGraph. We highlight the volume regions in their original color saturation to indicate their locations. As we can see, these two visually distinct spatiotemporal regions in the hurricane data are well separated in the TransGraph. Since our TransGraph organizes states in a hierarchical manner, the user can easily explore a finer level-of-detail and make finer state selection to narrow down the spatial regions of interest. Figure 3.5 (a), (c), and (d) are examples that demonstrate this capability of hierarchical exploration.

In Figure 3.7, we can observe that multiple branches in the TransGraph of the ionization data set spread out along different directions after a certain time step. To find out where these branches correspond to in the volume, we select nodes (states) at time step 118 and highlight the volume regions in their original color saturation. We also extract the corresponding state transition for each branch and overlay with the volume rendering for better observation. As we can see, these four branches correspond to different spatial regions and express their respective state evolutions.

3.4.3 TransGraph Query

Leveraging the TransGraph, we can perform state, transition, and time step queries to further understand the underlying data. In Figures 3.8 and 3.9, we show two examples of state and time step queries with the combustion data set. In Figure 3.8 (a), we query the first-level nodes (states) with the outgoing transition frequency larger than 0.108 and the query result is shown with colored nodes in the graph (nodes in the current time step are shown in red and blue, others are in green). We can see three branches of colored nodes along the direction of time evolution. They correspond to the upper, middle, and lower regions in the volume, as shown in Figure 3.8 (b). In Figure 3.9 (a), we perform a similar state query on the second-level nodes (states) and the brushed result is highlighted in Figure 3.9 (b). Comparing Figures 3.8 (a) and 3.9 (a) as well as the rendering results in Figures 3.8 (b) and 3.9 (b), we can infer that the middle part of the TransGraph (and the volume)

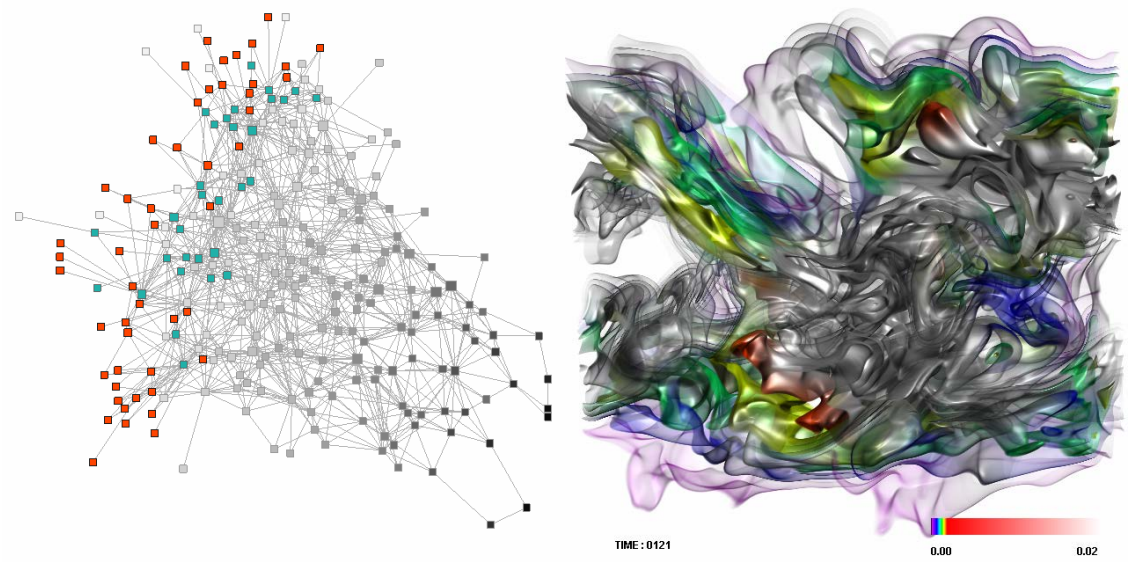


(a) (b)

Figure 3.10. (a) the query of the first-level states that are only involved in self-transition at time step 121. These nodes are in red and the rest of states at the same time step are in green. (b) the volume highlighting corresponding to the red nodes shown in (a).

has more nodes with larger incoming transition frequency than the upper and lower parts of the TransGraph (and the volume). In Figures 3.10 and 3.11, we show two examples of transition query. The first query in Figure 3.10 (a) shows transitions with frequency larger than 0.12. This is to bring out transitions that are frequently happening through the time series. The volume highlighting in Figure 3.10 (b) indicates that at time step 16, these transitions correspond to the middle region in the volume—the place where the two layers interact with each other. The second query in Figure 3.11 (a) shows nodes that are only involved in self-transition at time step 121. The volume highlighting in Figure 3.11 (b) indicates that those regions are stable in the sense that they only transfer to the same states in the next time step.

The TransGraphs in Figures 3.8, 3.9, 3.10, and 3.11 also tell us that in early time steps, we have a less number of states available. The number of states increases as the time



(a) (b)

Figure 3.11. Left: the TransGraph of the combustion data set with transition query. (a) the query of the first-level transitions (either outgoing or incoming) having their frequency larger than 0.12. These edges are in black and corresponding nodes are in red and green. (b) the volume highlighting corresponding to the red nodes brushed at time step 16. The rest of nodes in other time steps are in green.

step increases. Meanwhile, an increasing number of edges (transitions) exist among the upper, middle, and lower regions in the volume. This indicates the increasing turbulent nature of the combustion data set as the time evolves. Furthermore, the time step query in Figure 3.9 (a) which shows the number of active nodes in the second level also indicates the increasing number of nodes as the time step increases. It is important to point out that all these findings (except the correspondence between volume regions and graph nodes) can be solely inferred from the TransGraph beforehand without the present of the volume view. The volume view can help us put the graph in the context, make connection to the spatiotemporal data, and confirm our findings derived from the TransGraph.

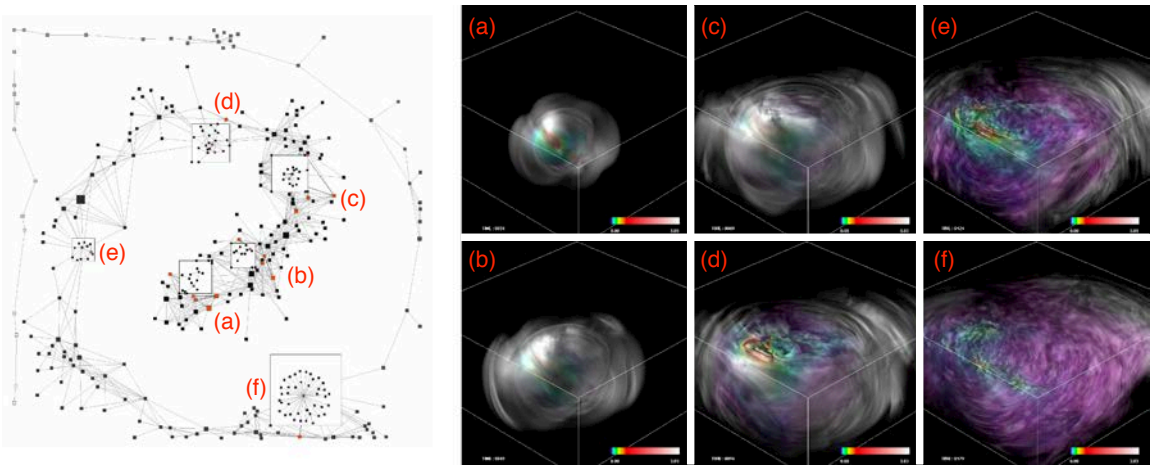
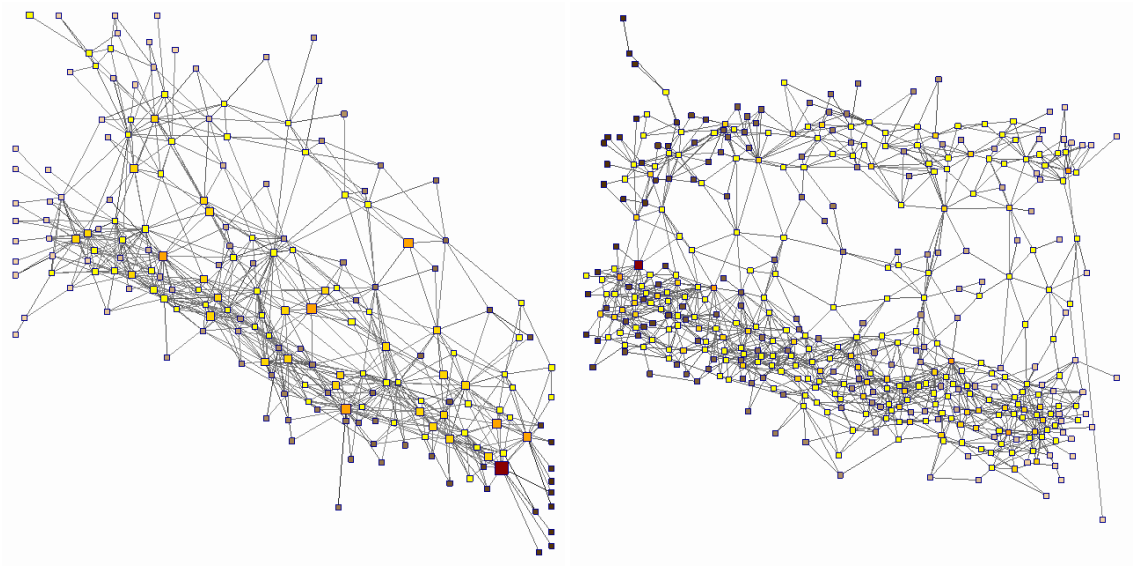


Figure 3.12. Left: the TransGraph of the earthquake data set with dynamic tracking. We select a volume region corresponding to the earthquake's epicenter at time step 34, which is highlighted in its original color saturation in (a). Right: (a) to (f) are the dynamic tracking results in the volume at selected time steps 34, 49, 69, 94, 124, and 179, respectively. These images show the propagation of data transition over time. The corresponding nodes (states) and edges (transitions) at those time steps are highlighted in red and black, respectively, in the TransGraph.

3.4.4 TransGraph Tracking

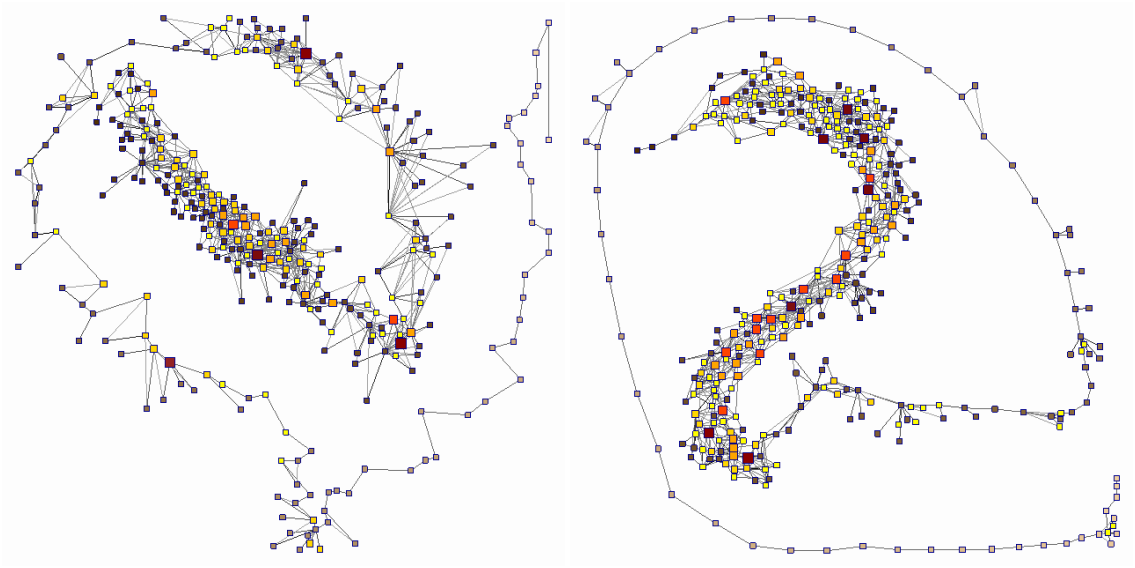
In Figure 3.12, we show an example of dynamic tracking with the earthquake data set. A spiral pattern is formed for the TransGraph due to the large number of time steps (599) involved. The TransGraph reveals that the earthquake data set consists of many more states and transitions in the early time steps than later time steps. This complies with the fact that the earthquake breakout is within the first 200 time steps. After that, the earthquake diminishes gradually for the rest of 400 time steps. To perform dynamic tracking, we first select a volume region at time step 34, which corresponds to the earthquake's epicenter and the region is highlighted in its original color saturation in (a). The tracking results on the volume and graph are shown in the figure. Even though the underlying transitions are computed on a block-wise manner, we allow the user to adjust the propagation speed to



(a) (b)

Figure 3.13. The TransGraph layouts with different parameter settings (refer to Tables 3.1 and 3.2). (a) and (b) are for the climate data set with block sizes of $15 \times 6 \times 9$ and $30 \times 11 \times 9$, respectively.

give the impression of voxel-wise data transition. Specifically, for each of the initial blocks selected for tracking, we propagate its transitions along 27 neighboring blocks (including itself, i.e., self-transition) with a constant speed. The transition probability for each pair of blocks is precomputed and we use the average transition probabilities over a moving time window to ensure smooth transition of color saturation over time. Moreover, we consider the distances of a voxel to the centers of initial blocks selected to further adjust the color saturation of the voxel. The result in Figure 3.12 (d) shows the effect of anisotropic transition as the color saturations of purple voxels are not the same even though they are equally close to the initial blocks' centers.



(a) (b)

Figure 3.14. The TransGraph layouts with different parameter settings (refer to Tables 3.1 and 3.2). (a) and (b) are for the earthquake data set with block sizes of $16 \times 16 \times 8$ and $32 \times 32 \times 16$, respectively.

3.4.5 Parameter Choices

Figures 3.13 and 3.14 show the TransGraph layouts of two data sets under different parameter settings. Tables 3.1 and 3.2 list the detail of parameter values used. In Figures 3.13 and 3.14, the nodes of the TransGraph are colored according to the number of children in the next level of the hierarchy, modulated by the order of the time steps (early time steps darker, later time steps brighter). Comparing Figures 3.6 with 3.13, and 3.12 with 3.14, we can see that although different sets of parameters give different graph layouts, the topology of the TransGraph remains almost the same and is insensitive to parameter changes and the randomness of the Fruchterman-Reingold algorithm. Therefore, the TransGraph is largely determined by the nature of the underlying time-varying data, while parameter changes only introduce slight variations in the layout.

CHAPTER 4

MINING TRANSITION GRAPHS FOR UNDERSTANDING TIME-VARYING VOLUMETRIC DATA

4.1 Overview

In this chapter, we present a graph analytics approach that enables the gaining of deep meanings from large transition graphs derived from time-varying volumetric data. This is achieved through the utilization of a series of graph analysis techniques including graph simplification, community detection, and visual recommendation. We demonstrate our solution with several time-varying data sets of different sizes and characteristics, showing that our approach represents a significant step forward in applying graph-based techniques for scientific data analysis and visualization. For gaining insights from the data, we show that our solution is more efficient and effective than simply asking users to extract relationships via standard interaction techniques, especially when the data set is large and the relationships are complex.

4.2 Transition Graph Construction

Our work is based on the transition relationships in time-varying volumetric data sets as defined in TransGraph [51]. The ideas and techniques presented are applicable to other graph-based representations. They work most effectively when the graph is large. In the following, we briefly review the construction of a transition graph. For details, please refer to Chapter 3.

We first partition the volume data at each time step into blocks. Each block is, for example, $N \times N \times N$ voxels. Given two blocks, we compute the Jensen-Shannon divergence (JSD) of their histograms as the dissimilarity or distance between them. Then we group these blocks based on their spatial and temporal adjacency within a small time window w. Specifically, given a target block b_t, we first check its spatial or temporal neighbor b_n to see whether it is similar to b_t. If so, then b_n is clustered to b_t. We continue to compare the neighbors of b_n with b_t. This process repeats until no similar blocks could be found in w. Note that if b_n is already clustered to another block, we will not check its neighbors when b_n is reached.

Finally, we derive directional transition probabilities among blocks to construct the transition graph. In the transition graph, a node denotes a *state* which represents a group of spatiotemporally neighboring blocks, and a directed edge between two states indicates their *transition probability*. In extreme cases, a state may represent a single block. A *transition* $i \rightarrow j$ occurs between two blocks i and j, from one block i at time step t to another block j at time step $t+1$, if and only if their spatial locations are the same. Given two groups g_i and g_j, the *directional transition probability* $p_{g_i \rightarrow g_j}$ is the ratio between the number of transitions from g_i to g_j and the total number of transitions from g_i to all the groups (including itself). As such, a transition indicates a chance for one group to transfer to another group, and its probability measures how high the chance is.

To draw the transition graph, we apply a two-step process. First, we use the Fruchterman-Reingold force-directed layout algorithm [44] to create the initial layout. Since we draw nodes with certain sizes, visual occlusion becomes unavoidable for a large graph. We therefore utilize four forces: bidirectional repulsive force, unidirectional repulsive force, spring force, and attractive force, to reduce the overlap while preserving the overall graph structure [51].

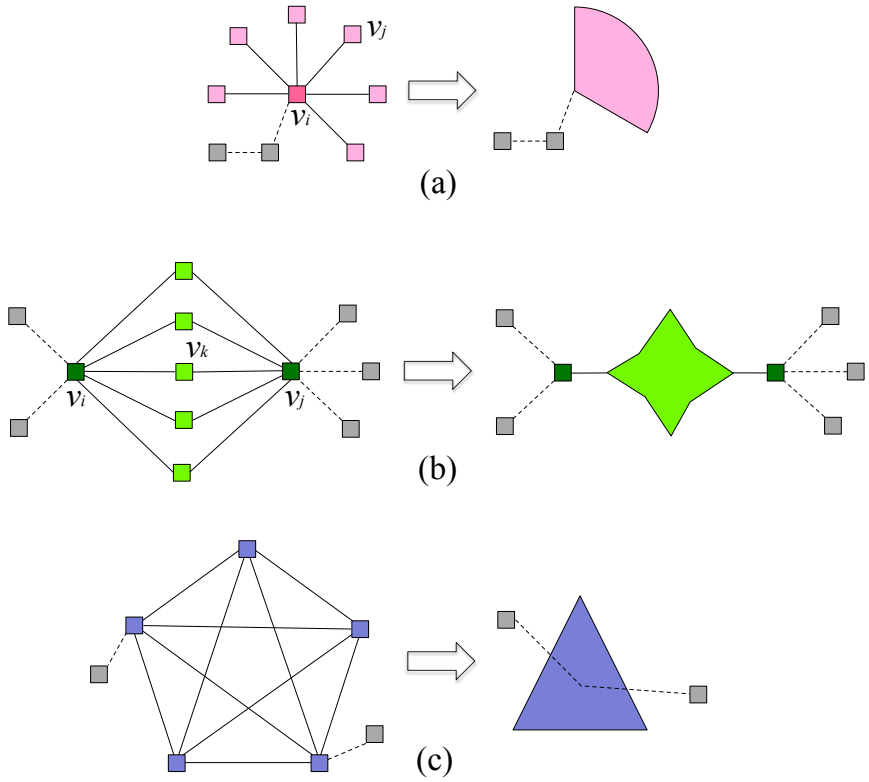
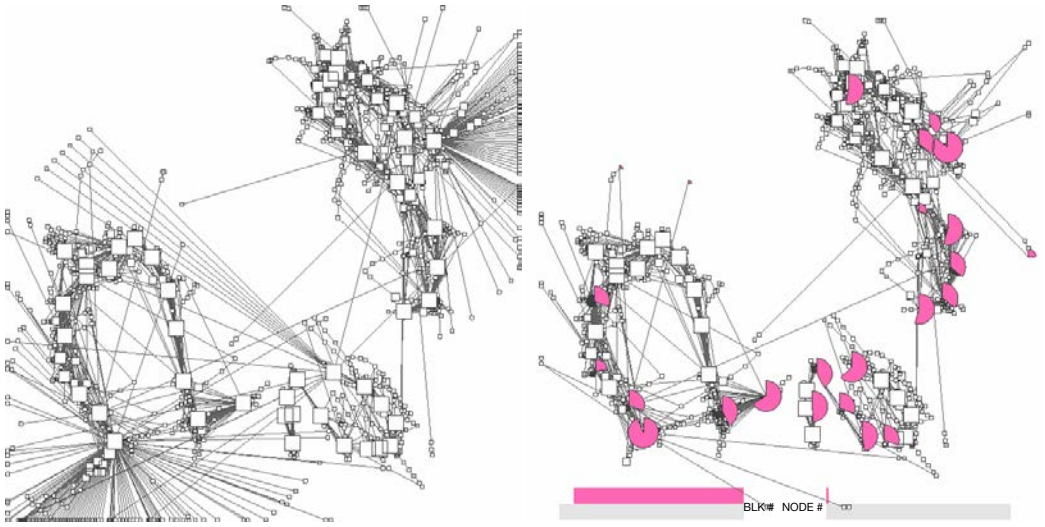


Figure 4.1. Graph features for simplification. (a) to (c) show examples for fan, connector, and clique, respectively. We use the included angle of fan, the size of star, and the size of triangle to indicate the number of nodes simplified for the three graph features, respectively. Regular (i.e., unsimplified) nodes are drawn as squares. In (c), all other nodes connecting to the clique are connected to the center of the triangle.

4.3 Graph Simplification

Visual clutter is common in a transition graph due to the presence of a large number of nodes and edges. We can reduce the clutter through *edge reduction* or *node reduction*. For edge reduction, edge simplification and compression techniques [30, 40, 94, 103, 124] identify the relationships among nodes, group the nodes with similar edge configurations together, and draw a single edge to the group instead of edges to group members. For node reduction, TopoLayout [5] detects topological features such as trees, connected compo-



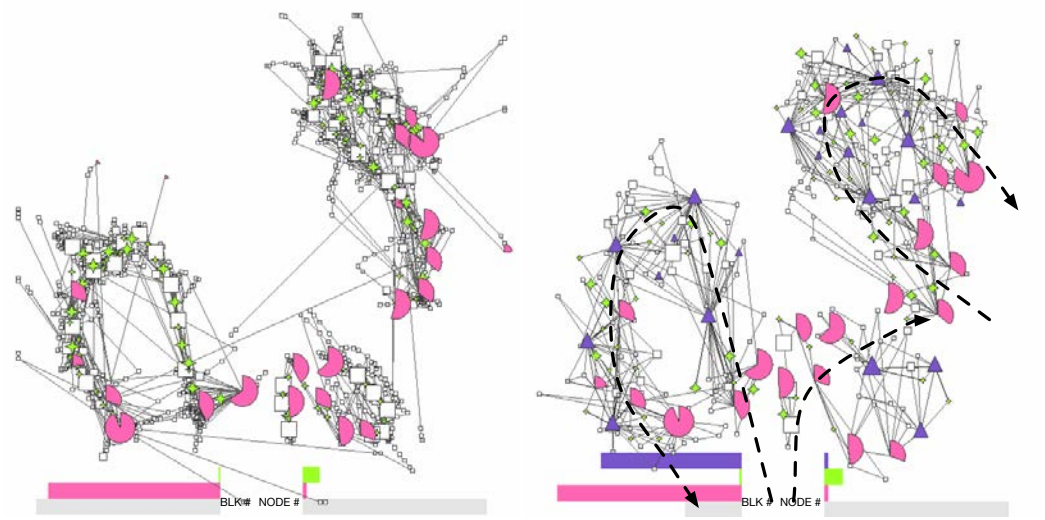
(a) (b)

Figure 4.2. (a) and (b) are the original transition graph and the graph after detecting fans, respectively.

nents and biconnected components, and motif simplification [37] detects fans, connectors and cliques. Both techniques use a single node to replace a graph feature.

In our work, we focus on node reduction and replace certain graph features with symbols in order to highlight important graph structures. Similar to motif simplification [37], we select three graph features for simplification because they represent meaningful transition relationships and do not impose restrictions to the order of simplification. Figure 4.1 illustrates these three graph features and their corresponding symbols for simplification. For graph simplification, we do not consider edge directions in the transition graph. However, the three graph features themselves already imply directional edge information as explained in the following.

- A *fan* is a configuration of a central node with multiple leaf nodes of degree one. The fan is simplified with the fan symbol as shown in Figure 4.1 (a). The undirected edge between nodes v_i and v_j has three kinds of relationships: $v_i \rightarrow v_j$, $v_j \rightarrow v_i$, and

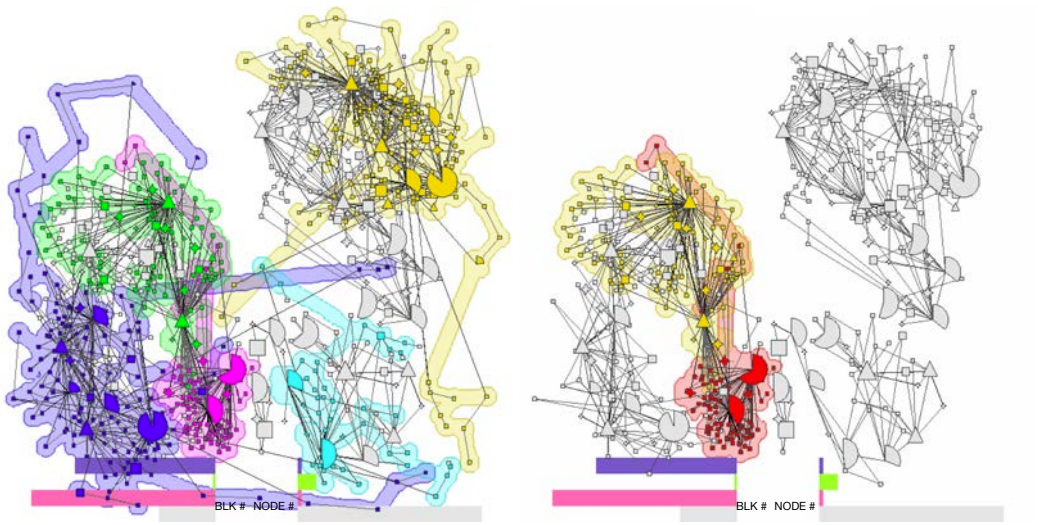


(a) (b)

Figure 4.3. (a) and (b) are the graphs after detecting connectors and cliques, respectively. In (b), we show the graph after layout adjustment which pushes nodes apart for clear observation. There are three branches of nodes in the graph and the evolution of time in each branch is marked with a dashed line. The histograms at the bottom-left and bottom-right corners depict the numbers of volumetric blocks and nodes belonging to the four types of nodes, respectively. Gray for regular nodes, pink for fans, green for connectors, and blue for cliques.

$v_i \leftrightarrow v_j$. Each node encompasses data blocks grouped within a fixed time interval. If a node has only outgoing (incoming) edges, it must be in the first (last) time interval. Therefore, the fan with one directional edge $v_j \rightarrow v_i$ at the first time interval ($v_i \rightarrow v_j$ at the last time interval) indicates the convergence (divergence) of states. In other time intervals, the bidirectional edge $v_i \leftrightarrow v_j$ indicates that v_i derives v_j and v_j will return to v_i within certain time steps. Therefore, v_j can be considered as an *interruption* of v_i.

- A *connector* is a configuration of two ending nodes with multiple intermediate nodes of degree two. The connector is simplified with the star symbol as shown in Figure 4.1 (b). Assume that two ending nodes are v_i and v_j and an intermediate node is v_k,



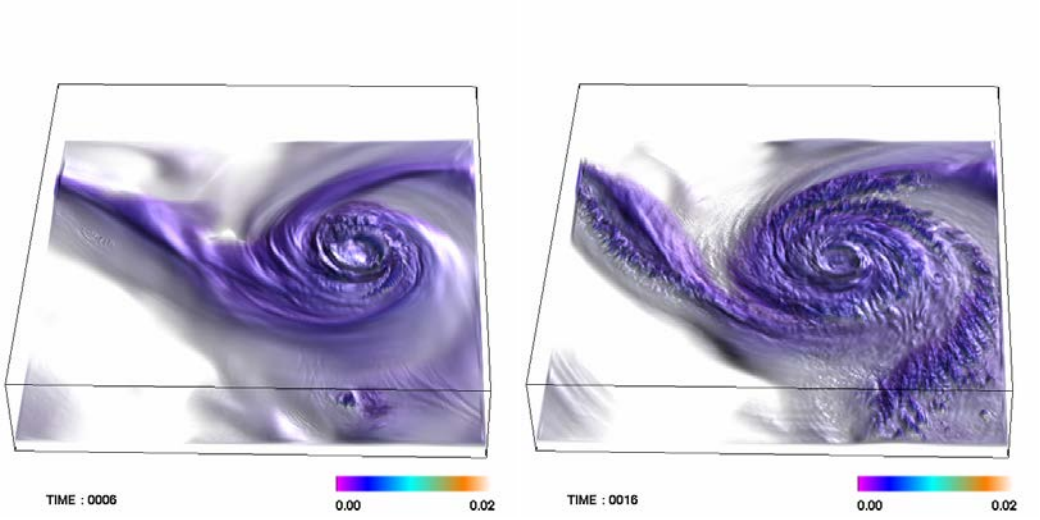
(a) (b)

Figure 4.4. (a) five of the top eight largest communities detected for the transition graph of the hurricane data set and highlighted using Bubble Sets. (b) the user chooses a community with nodes highlighted in red and we automatically recommend another community with nodes highlighted in yellow.

and without loss of generality, there is a directed edge $v_i \rightarrow v_k$, then we have two possible cases. First, v_k derives v_j and we have directed edges $v_i \rightarrow v_k$ and $v_k \rightarrow v_j$. Second, v_k returns to v_i (i.e., now we have $v_i \leftrightarrow v_k$) and we have the directed edge $v_j \rightarrow v_k$. In both cases, v_k can be considered as an *intermediate* state of v_i and v_j. Similar to fans, connectors may also occur in the first and last time intervals. In these cases, we have edges $v_i \rightarrow v_k$ and $v_j \rightarrow v_k$ (for the first time interval), and $v_k \rightarrow v_i$ and $v_k \rightarrow v_j$ (for the last time interval). They correspond to the convergence and divergence of states, respectively.

- A *clique* is a configuration of more than two nodes that are fully connected. As shown in Figure 4.1 (c), cliques represent a set of nodes changing their states among each other. We do not require all edges in a clique to be bidirectional.

Implication. According to the definition of fan, a node will transit to another node at

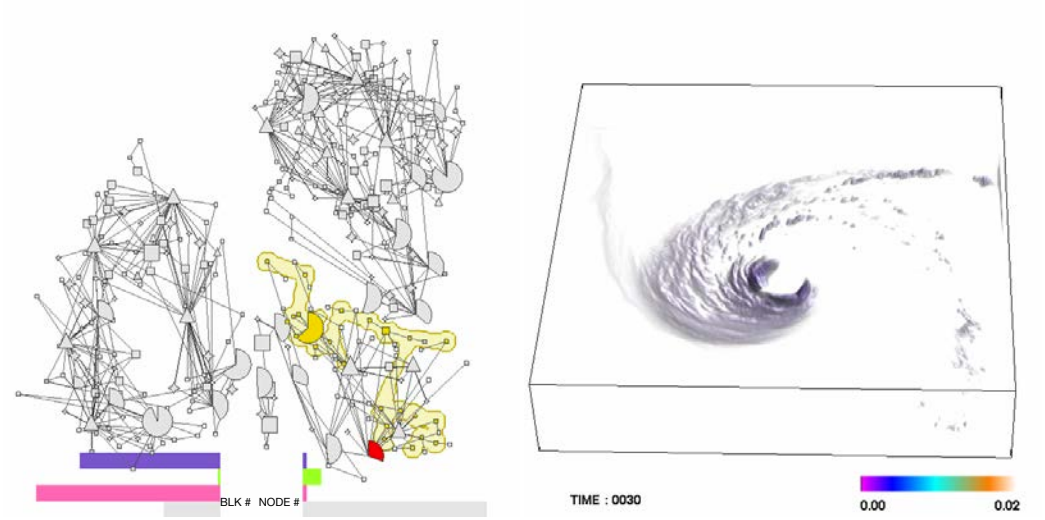


(a) (b)

Figure 4.5. (a) and (b) are the rendering of the corresponding blocks of the chosen and recommended communities in Figure 4.4 (b) at two different time steps, respectively.

later time steps and transit back to itself eventually. As a result, a fan could represent a volume region that has *turbulence* and eventually returns to the original state. In a connector, the nodes in between can be considered as a set of intermediate states between the two ending nodes. Therefore, a connector represents a *transition* from a state to another. A clique is a simplification of several nodes that have strong connections between them. These nodes therefore represent a *locally stable region* during that time period.

Order of Simplification. We note that the order of simplification does not affect the final result. This is because they only apply to regular (i.e., unsimplified) nodes and these three graph features are unique and independent of each other. However, to achieve the best time performance in detecting these graph features, we first simplify fans, followed by connectors and cliques. Fan simplification can reduce the number of nodes dramatically, thus we take this simplification first. Clique simplification is placed last because the time complexity to identify cliques is the highest. Figures 4.2 and 4.3 show an example of the

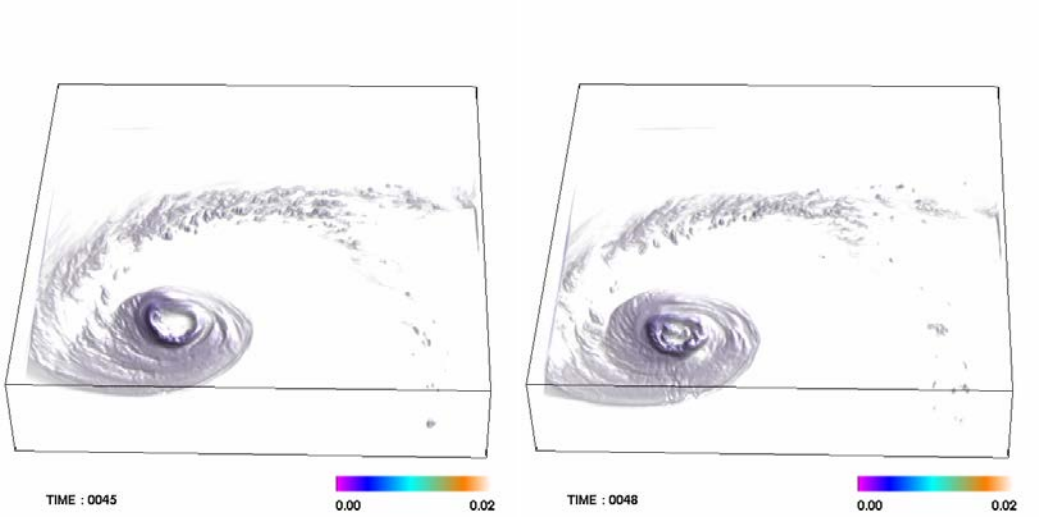


(a) (b)

Figure 4.6. (a) the user chooses a fan shown in red and we automatically recommend nodes shown in yellow. (b) is the rendering of the corresponding blocks of the chosen node.

results of graph simplification. To further reduce visual clutter, we display nodes with small numbers of data blocks only if they belong to query results for highlighting. Based on our experience, those nodes that are not rendered are mainly regular nodes. Some connectors are also omitted while fans and cliques are less likely to be. We report the number of nodes originally created, after simplification, and finally displayed (see Table 4.2). In the following, we describe our algorithms to detect these graph features.

Fan Detection. Our fan detection creates a data container called *map* that stores a set of tuples. Each tuple represents a pair of nodes: a central node and a leaf node. A tuple uses the *central* node as the *key* and the *leaf* node as the *value*. Thus, a set of tuples sharing the same key represents a fan feature. The detection algorithm first goes through all the nodes in the graph, finds the nodes with degree of one, and inserts them to the map. Then, we find those keys with more than one tuple in the map as the central nodes and their corresponding values as leaf nodes.



(a) (b)

Figure 4.7. (a) and (b) are the rendering of the corresponding blocks of the recommended nodes at two later time steps of Figure 4.6 (b).

Algorithm 3 EXPAND($G = (V,E)$)

```
Q ← CANDIDATENODEDETECTION(G = (V,E))
for each node q in Q do
    Build a subgraph G' that consists of all the nodes connecting to q
    if G' does not contain more than one node then
        The nodes expanded along the path, q and its adjacent nodes form a clique
    else
        EXPAND(G' = (V',E'))
    end if
end for
```

Connector Detection. Similar to fan detection, our connector detection also creates a map. To detect a connector, we use the two *ending* nodes as the *key* and an *intermediate* node as the *value*. We first go through all the nodes in the graph, find all the nodes with degree of two, and insert them to the map. If any key has more than two tuples in the map, the keys are the ending nodes and their corresponding values are intermediate nodes.

Clique Detection. To detect cliques, we apply the algorithm introduced by Tomita et al. [117]. This algorithm has a time complexity of $3^{|V|/3}$, where $|V|$ is the number of nodes

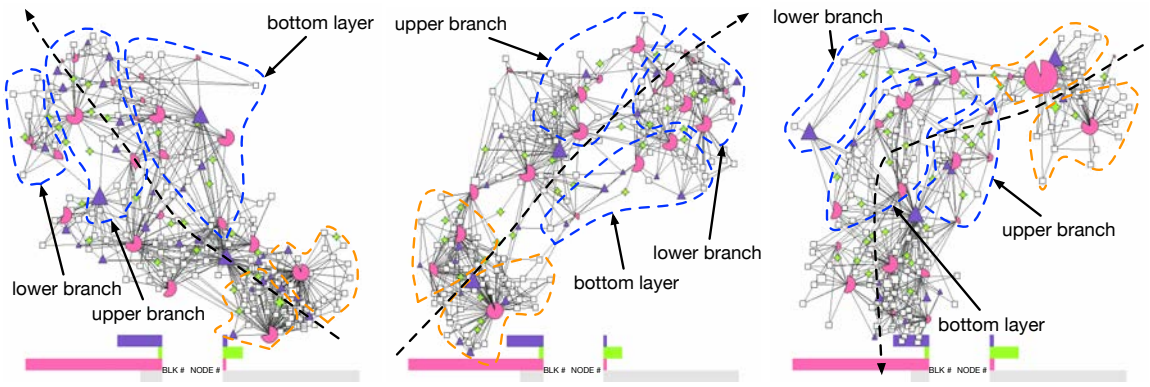
Algorithm 4 CANDIDATENODEDETECTION($G = (V, E)$)

```
Create a FIFO queue Q
Find the node q that has the largest degree in G
Insert q to Q
for each node u in V do
    if u is not adjacent to q then
        Insert u to Q
    end if
end for
return Q
```

in the graph. Since this algorithm finds all the cliques, the cliques may share nodes. To avoid this situation, we utilize a greedy algorithm to select cliques in an iterative manner. In each iteration, we always select the largest clique and rule out all other cliques that share node(s) with any previously selected clique. Algorithm 3 detects all the cliques in the graph. Given a graph (subgraph), we always record the candidate nodes to expand as described in Algorithm 4. We first create a FIFO *queue* Q, and insert the node q with the highest degree to the queue. Then the nodes that are not adjacent to q are also inserted to Q in order. The selection of the nonadjacent nodes is used to avoid duplication. Given a node q in graph G, this algorithm generates a subgraph G' that consists of all the nodes in G that are adjacent to q. Then this algorithm treats G' as a new input graph and repeats the above process. If q does not have any adjacent nodes or has only one adjacent node left, the nodes already expand in full. Thus, q and its adjacent nodes along the path form a clique.

4.4 Community Detection

Generally speaking, techniques for graph partitioning and clustering aim to identify node subsets called *communities* with many internal and few external edges. Unlike a clique which is a subset of nodes inducing a complete subgraph, a community is less restrictive and more practical in many applications because it identifies a highly cohesive structure as a cluster by the mere absence of a few edges. Detecting communities will help us investigate community structures and their evolution. We leverage SLPA [146], an ex-



(a) DCMIP cam-fv (b) DCMIP cam-se (c) DCMIP fim

Figure 4.8. (a) to (c) are the transition graphs of the three DCMIP climate simulation data sets. The evolution of time is marked with the black dashed line.

tended version of the label propagation algorithm (LPA), for community detection. SLPA can handle weighted directed graphs which fits our transition graph. It has an efficient time complexity of $\overline{d}|V|$, where $\overline{d}$ is the average degree of nodes and $|V|$ is the number of nodes. This algorithm also attempts to avoid producing a number of small communities.

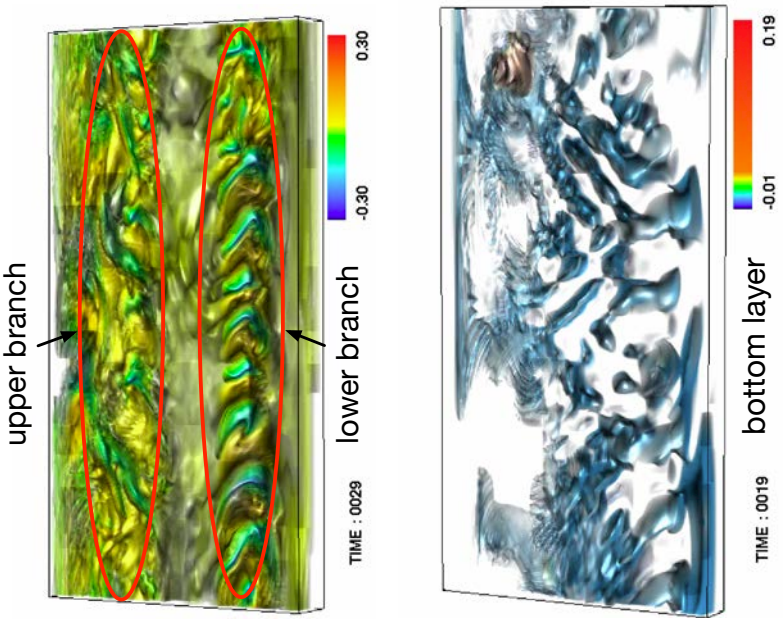
SLPA is outlined in Algorithm 5. In SLPA, a *label* stands for a community identification, and each node has a buffer to store the labels received from neighboring nodes (i.e., having edges pointing to the node). SLPA detects the communities in an iterative manner. In each step, a node serves as a *listener* while its neighbors serve as the *speakers*. At the first iteration, each node adds a unique label to its buffer, which means that every node forms a community. In later iterations, SLPA goes through every node and sets it to be the listener. Its neighbors become the speakers. Each speaker randomly selects a label from its buffer and sends it to the listener. The listener selects the label with the highest rank calculated from its neighbors and adds it to the buffer. The equation to calculate the rank is as follows

$$L_i = \arg\max_{j \in N_i} p_{j \to i} L_j, \tag{4.1}$$

Algorithm 5 COMMUNITYDETECTION($G = (V,E)$)

```
for each node u in V do
    Create a buffer
end for
for each iteration k do
    if k = 1 then
        for each node u in V do
            Insert label u to the buffer
        end for
    else
        for each node u in V do
            Set u as the listener
            for each neighbor node v of u do
                Randomly pick a label l from the buffer
                Send l to node u
            end for
            Gather the labels from neighbors
            Pick the label with the highest frequency and insert it to the buffer
        end for
    end if
end for
for each node u in V do
    Find the label l with the highest frequency
    Use it as the community identification
end for
Nodes with the same identification form a community
```

where L_i is the label with the highest rank received by node i, L_j is the label sent by node j, N_i is the neighborhood of node i, and $p_{j \to i}$ is the transition probability from j to i as defined in our transition graph. After a certain number of iterations, each node uses the label with the highest frequency in its buffer as its community identification. The nodes with the same identification form a community. Figure 4.4 (a) shows an example of communities detected. We implement and draw Bubble Sets [27] to distinguish different communities. Bubble Sets use continuous, often concave, isocontours to delineate group memberships, maintaining the spatial arrangement of the primary data relationship given by the layout algorithm. Using Bubble Sets, we can produce tight capture of group members with less ambiguity compared with using convex hulls. Although the Bubble Sets overlap each other, notice that nodes in communities are disjoint, i.e., any two communities do not share any node in common.



(a) all fans (b) all cliques

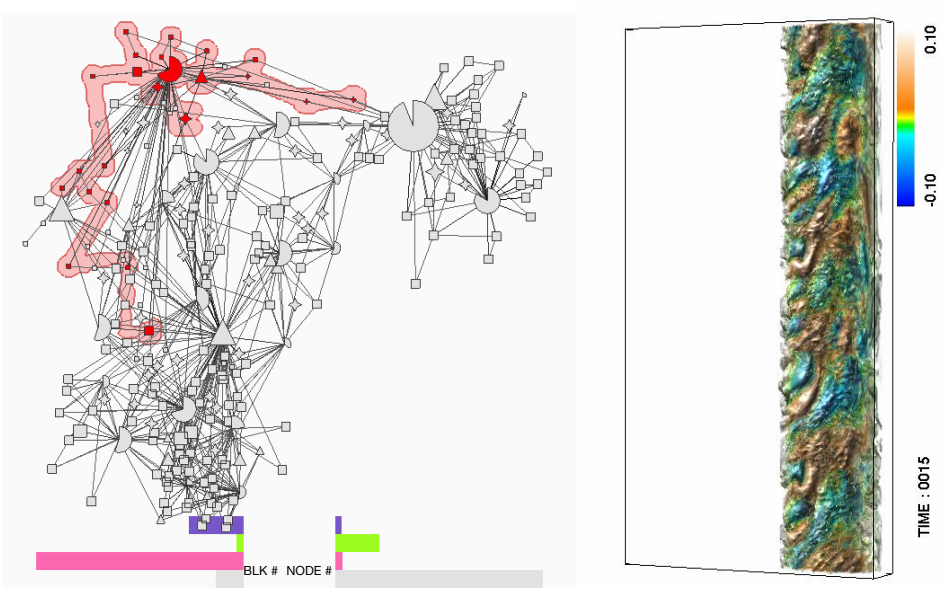
Figure 4.9. (a) and (b) are the rendering of the corresponding blocks of all fans and cliques from DCMIP cam-fv and DCMIP cam-se data sets of Figure 4.8.

4.5 Visual Recommendation

While the focus of this work is on graph interaction, the user can also interact with the volume to highlight relevant aspects of the graph. Details on that are in the TransGraph work [51]. Here we focus on the newly introduced node and community recommendation functions.

4.5.1 Node Recommendation

To further guide the exploration of graph, we recommend similar nodes when a node or multiple nodes are selected. Heer et al. [57] recommended similar nodes with a query relaxation engine. The recommended nodes fulfill the same query requirements but in different scenarios. Different from query relaxation, we recommend similar nodes based



(a) (b)

Figure 4.10. One of the three communities detected from the DCMIP fim data set shown in (a) and the rendering of its corresponding blocks in (b) respectively. (b) corresponds to the lower branch.

on their similarities. We leverage the SimRank algorithm [38] to measure the similarities between nodes. SimRank considers two nodes to be similar if they are related to similar nodes. Given an unweighted directed graph $G = (V, E)$, if the incoming nodes of two nodes v_a and v_b are similar, then v_a and v_b are similar. The similarity between v_a and v_b is calculated as

$$s(v_a, v_b) = \frac{\sum_{i=1}^{|I(v_a)|} \sum_{j=1}^{|I(v_b)|} s(I_i(v_a), I_j(v_b))}{|I(v_a)||I(v_b)|}, \tag{4.2}$$

where $I(v_a)$ is the set of nodes pointing to node v_a, and $I_i(v_a)$ is the i-th node in $I(v_a)$. If neither v_a nor v_b has incoming nodes, $s(v_a, v_b) = 0$ (least similar). SimRank stores a $|V| \times |V|$ matrix $\mathbf{S}$ to record the similarities between nodes, where $|V|$ is the number of nodes in the graph. In the first iteration, all the similarity values along the diagonal of $\mathbf{S}$ are set to 1 (most similar). In later iterations, we update the similarities between nodes

Algorithm 6 NODERECOMMENDATION($G = (V, E)$)

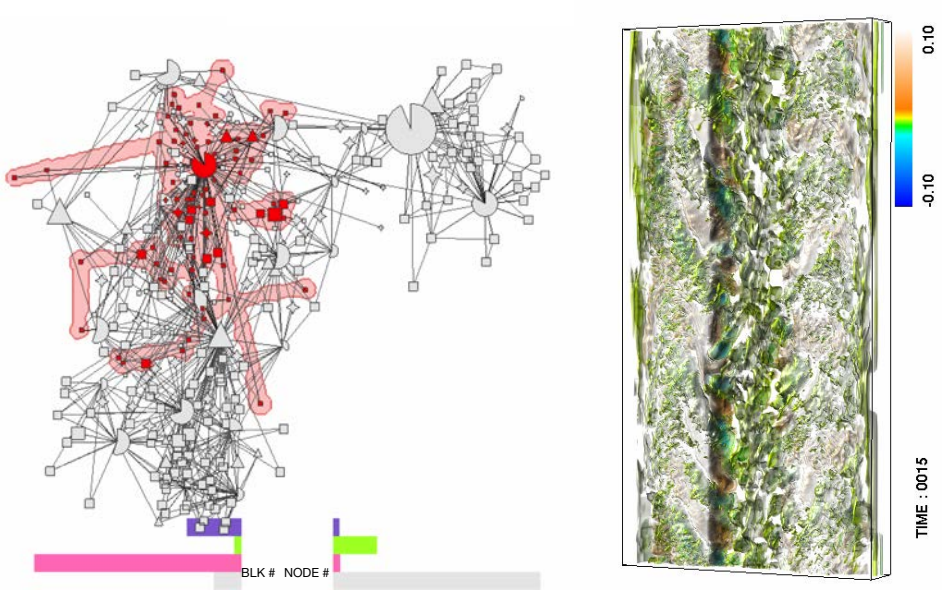
```
Create a |V| × |V| similarity matrix S
Create a |V| × |V| temporary matrix S_t
for each iteration k do
    if k = 1 then
        for each element s_t(u, v) in S_t do
            s_t(u, v) ← 0
        end for
        for each node v in V do
            s_t(v, v) ← 1
        end for
    else
        for each node v_a in V do
            for each node v_b in V do
                if v_a = v_b then
                    s_t(v_a, v_b) ← 1
                else
                    s_t(v_a, v_b) ← 0
                    for each neighbor node I_i(v_a) of v_a do
                        for each neighbor node I_j(v_b) of v_b do
                            s_t(v_a, v_b) += s(I_i(v_a), I_j(v_b)) × p_{i→v_a} × p_{j→v_b}
                        end for
                    end for
                    s_t(v_a, v_b) ← 1/(|I(v_a)||I(v_b)|) s_t(v_a, v_b)
                end if
            end for
        end for
    end if
    S ← S_t
end for
```

according to Equation 4.2. SimRank has the time complexity of $k\overline{d}|V|^2$, where k is the number of iterations and $\overline{d}$ is the average degree of nodes.

In our graph, each edge carries a weight indicating the transition probability between two incident nodes, thus the original SimRank cannot be immediately applied. As described in Algorithm 6, our modified SimRank algorithm takes transition probabilities into account. We modify Equation 4.2 to the following

$$s(v_a, v_b) = \frac{\sum_{i=1}^{|I(v_a)|} \sum_{j=1}^{|I(v_b)|} s(I_i(v_a), I_j(v_b)) \times p_{i \to v_a} \times p_{j \to v_b}}{|I(v_a)||I(v_b)|}, \tag{4.3}$$

where $p_{i \to v_a}$ is the transition probability from node $I_i(v_a)$ to node v_a. We also extend Equation 4.3 to outgoing edges

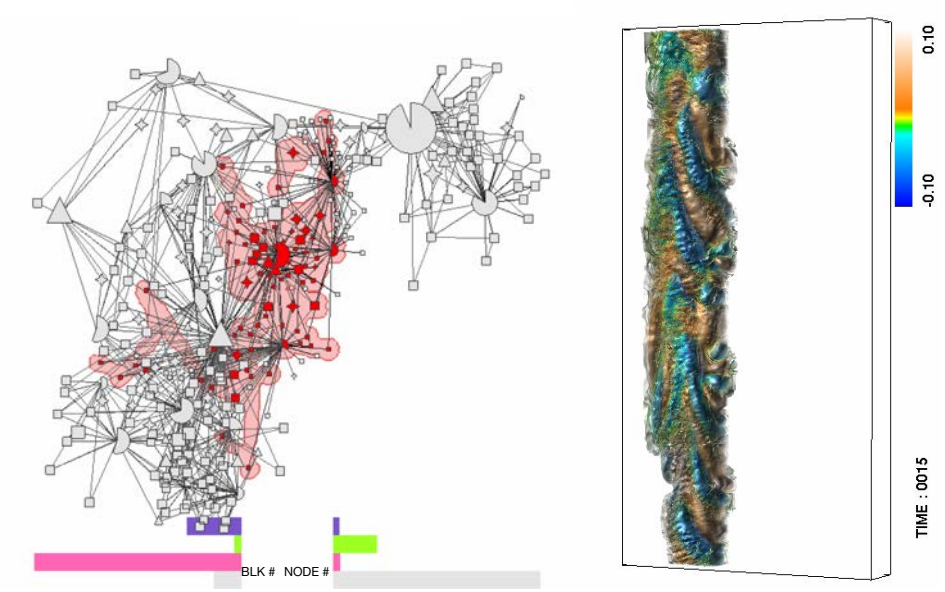


(a) (b)

Figure 4.11. One of the three communities detected from the DCMIP fim data set shown in (a) and the rendering of its corresponding blocks in (b) respectively. (b) corresponds to the surrounding.

$$s(v_a, v_b) = \frac{\sum_{i=1}^{|O(v_a)|} \sum_{j=1}^{|O(v_b)|} s(O_i(v_a), O_j(v_b)) \times p'_{i \to v_a} \times p'_{j \to v_b}}{|O(v_a)||O(v_b)|}, \tag{4.4}$$

where $O(v_a)$ is the set of nodes pointed from node v_a and $p'_{i \to v_a}$ is the transition probability from node v_a to node $O_i(v_a)$. Users can select either Equation 4.3, Equation 4.4, or a combination of both as the final similarity measure. Figures 4.6 and 4.7 show an example of node recommendation based on incoming transition probabilities. The recommended nodes are highlighted with a Bubble Set. As we can see, similar nodes recommended show up in the volume highlighting results, which allow the user to track the evolution of selected node.



(a) (b)

Figure 4.12. One of the three communities detected from the DCMIP fim data set shown in (a) and the rendering of its corresponding blocks in (b) respectively. (b) corresponds to the upper branch.

4.5.2 Community Recommendation

Besides node recommendation, we also recommend similar communities when a community is selected. For simplicity, we first convert each community to an unweighted, undirected graph. Then we treat the similarity between two graphs as the similarity between the two corresponding communities.

To calculate the distance between two unweighted, undirected graphs, we apply the algorithm introduced by Robles-Kelly and Hancock [102]. This algorithm finds a serial ordering of the nodes in a graph and converts *graphs* to *strings*. By comparing the difference between the two strings, we compute the difference between the two graphs.

Algorithm 7 utilizes *random walks* to find the ordering. It first calculates a normalized symmetric random walks probability matrix $\mathbf{P}'$ and computes $\mathbf{P}'$'s leading eigenvector ϕ. Each value in ϕ is related to a node in V indicating its importance. In order to sort the

Algorithm 7 GRAPHTOSTRING($\mathbf{A}$, V)

Create a degree matrix $\mathbf{D}$
for each node i in V **do**
  for each node j in V **do**
    if $i = j$ **then**
      $D(i,i) \leftarrow \frac{1}{\sum_{k=1}^{|V|} A(i,k)}$
    else
      $D(i,j) \leftarrow 0$
    end if
  end for
end for
Create a normalized symmetric random walks probability matrix $\mathbf{P}'$
$\mathbf{P}' \leftarrow \mathbf{D}^{\frac{1}{2}}\mathbf{A}\mathbf{D}^{\frac{1}{2}}$
Find the leading eigenvector ϕ of $\mathbf{P}'$ {A value in ϕ is related to a node in V}
Create an empty string s
Create a set s' consisting of all the nodes in V
Find the node n in s' with the largest value in ϕ
while s' is not empty **do**
  Insert n to s
  Remove n from s'
  if n's neighbors are all in s **then**
    $n \leftarrow$ the node which is the neighbor of a node in s and has the largest value in ϕ among those nodes in s'
  else
    $n \leftarrow$ the neighbor in s' which has the largest value in ϕ
  end if
end while
return s

nodes based on their importance and still preserve edge relationships, the algorithm first finds node n with the largest value in ϕ and puts n in the string s. Then, it looks for node n' with the largest value from n's neighbors and puts n' in s. After that, the algorithm begins to search n''s neighbors. This process repeats until all the nodes are in s. However, if all n's neighbors are in s, the algorithm looks for n' which is a neighbor of any node in s and has the largest value in ϕ among those nodes not in s yet. Then it puts n' in s and keeps searching among n''s neighbors. As a result, this algorithm converts a graph to a string and orders its nodes based on their importance and edge relationships.

After ordering the nodes, Algorithm 8 computes the difference between two strings s_1 and s_2 as the distance between the two corresponding graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. This algorithm creates a lattice $\mathbf{L}$ using s_1 as rows and s_2 as columns. The elements in $\mathbf{L}$ are only linked to their neighbors along the increasing horizontal, vertical

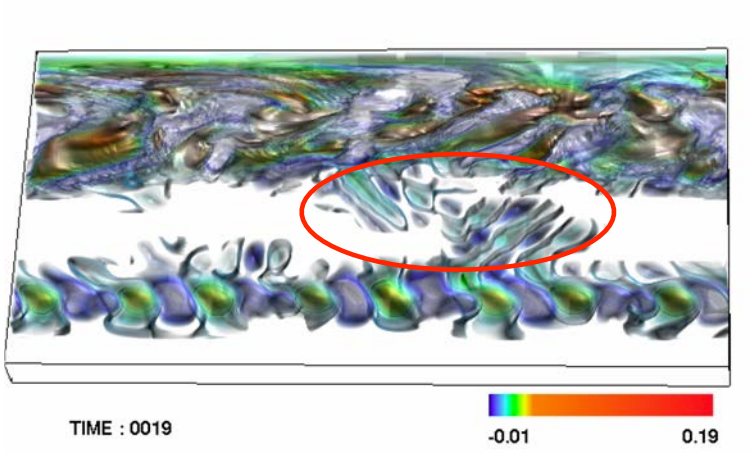


Figure 4.13. The rendering of blocks corresponding to the two branches of the DCMIP cam-se data set, revealing their direct interaction.

Algorithm 8 GRAPHEDITDISTANCE($\mathbf{A}_1$, $\mathbf{A}_2$, $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$)

$s_1 \leftarrow$ GRAPHTOSTRING($\mathbf{A}_1$, V_1)
$s_2 \leftarrow$ GRAPHTOSTRING($\mathbf{A}_2$, V_2)
$\mathbf{P}_1 \leftarrow$ the normalized symmetric random walks probability matrix of $\mathbf{A}_1$
$\mathbf{P}_2 \leftarrow$ the normalized symmetric random walks probability matrix of $\mathbf{A}_2$
Create a lattice $\mathbf{L}$ with size $|V_1| \times |V_2|$
$d(G_1, G_2) \leftarrow$ DIST($\mathbf{L}$, $\mathbf{A}_1$, $\mathbf{A}_2$, V_1, V_2, $\mathbf{P}_1$, $\mathbf{P}_2$)
return $d(G_1, G_2)$

and diagonal directions. A diagonal movement from $L(i, j)$ to $L(i+1, j+1)$ represents a matching of $E_1(s_1(i), s_1(i+1))$ and $E_2(s_2(j), s_2(j+1))$. A horizontal movement from $L(i, j)$ to $L(i+1, j)$ represents a null matching of node $s_1(i)$. Similarly, a vertical movement from $L(i, j)$ to $L(i, j+1)$ represents a null matching of node $s_2(j)$. Therefore, the difference between s_1 and s_2 is measured by the distance between the two elements $L(0,0)$ and $L(|V_1|-1, |V_2|-1)$. In addition, the distance can be treated as finding the *shortest path* from $L(0,0)$ to $L(|V_1|-1, |V_2|-1)$. Finally, we utilize Dijkstra's algorithm [32] to calculate the minimal distance between $L(0,0)$ and $L(|V_1|-1, |V_2|-1)$ and uses it as the difference between s_1 and s_2 (i.e., the distance between G_1 and G_2). In Dijkstra's algorithm, the distance between two neighboring nodes is calculated as

Algorithm 9 DIST($\mathbf{L}$, $\mathbf{A}_1$, $\mathbf{A}_2$, V_1, V_2, $\mathbf{P}_1$, $\mathbf{P}_2$)

Create the degree matrices $\mathbf{D}'_1$ and $\mathbf{D}'_2$
for each node i in V_1 **do**
  for each node j in V_1 **do**
    if $i = j$ **then**
      $D'_1(i,j) \leftarrow \sum_{j=1}^{|V_1|} A_1(i,j)$
    else
      $D'_1(i,j) \leftarrow 0$
    end if
  end for
end for
for each node i in V_2 **do**
  for each node j in V_2 **do**
    if $i = j$ **then**
      $D'_2(i,j) \leftarrow \sum_{j=1}^{|V_2|} A_2(i,j)$
    else
      $D'_2(i,j) \leftarrow 0$
    end if
  end for
end for
Use Dijkstra's algorithm to calculate $d(L(0,0), L(|V_1|-1, |V_2|-1))$
return $d(L(0,0), L(|V_1|-1, |V_2|-1))$

$$d = \beta_{(a,b)} \times \beta_{(c,d)} \times R_1(a,c) \times R_2(b,d), \tag{4.5}$$

where a, b, c and d are the indices of $L(a,b)$ and $L(c,d)$. $\beta_{(a,b)}$ can be calculated as

$$\beta_{(a,b)} = \frac{\max\{D'_1(a), D'_2(b)\} - \min\{D'_1(a), D'_2(b)\}}{\max\{D'_1(a), D'_2(b)\}}, \tag{4.6}$$

where $\mathbf{D}'$ is the degree matrix calculated in Algorithm 9. $\beta_{(c,d)}$ can be calculated similarly. R_1 in Equation 4.5 can be calculated as

$$R_1(a,c) = \begin{cases} P'_1(a,c), & \text{if } A_1(a,c) = 1 \\ \frac{2 \times |V_1| \times |V_2|}{|V_1| + |V_2|}, & \text{otherwise} \end{cases}, \tag{4.7}$$

where $\mathbf{P}'$ is the normalized symmetric random walks probability matrix and $\mathbf{A}$ is the adjacency matrix. R_2 can be calculated similarly.

Figure 4.4 (b) shows an example of community recommendation. The user chooses a community with nodes highlighted in red. The most similar community recommended is

TABLE 4.1

THE DATA SETS, PARAMETERS USED, AND TIMING RESULTS OF CLUSTERING

data set	dimension	block size	state clustering time / w / δ
DCMIP cam-fv	$360 \times 180 \times 30 \times 31$	$20 \times 10 \times 5$	99.62 / 10 / 0.6
DCMIP cam-se	$512 \times 256 \times 30 \times 31$	$32 \times 16 \times 5$	123.32 / 10 / 0.5
DCMIP fim	$360 \times 180 \times 30 \times 31$	$20 \times 10 \times 5$	137.83 / 10 / 0.45
earthquake	$256 \times 256 \times 96 \times 360$	$16 \times 16 \times 16$	43.71 / 10 / 0.4
		$16 \times 16 \times 8$	155.64 / 10 / 0.5
		$32 \times 32 \times 16$	19.49 / 10 / 0.25
hurricane	$500 \times 500 \times 100 \times 48$	$20 \times 20 \times 20$	187.67 / 12 / 0.25
ionization	$600 \times 248 \times 248 \times 200$	$30 \times 31 \times 31$	98.28 / 10 / 0.3
NOAA climate	$360 \times 66 \times 27 \times 120$	$15 \times 11 \times 9$	9.12 / 12 / 0.25
		$15 \times 6 \times 9$	17.16 / 12 / 0.3
		$9 \times 11 \times 9$	11.06 / 12 / 0.35

shown with nodes highlight in yellow. The correspondence of volume highlighting results in Figure 4.5 show the effectiveness of community recommendation.

4.6 Case Study Results

Data Sets, Parameters, and Timing. We evaluated our approach with several time-varying data sets listed in Tables 4.1 and 4.2. The timing results were collected using a workstation with an Intel Core i7-960 3.5GHz CPU and 24GB memory. We used MPI to

TABLE 4.2

THE DATA SETS AND TIMING RESULTS OF GRAPH SIMPLIFICATION AND VISUAL RECOMMENDATIONS

	layout create / adjust	# nodes original / simplified / displayed	community / node recommendation
DCMIP cam-fv	62.07 / 1.43	1909 / 1037 / 247	7.72 / 0.99
DCMIP cam-se	62.25 / 1.5	1959 / 1063 / 223	7.03 / 0.99
DCMIP fim	61.07 / 0.88	1868 / 786 / 212	1.18 / 0.54
earthquake	86.74 / 0.94	2567 / 806 / 356	0.31 / 0.4
	241.37 / 2.52	3783 / 1353 / 406	2.46 / 1.31
	32.00 / 0.89	1390 / 780 / 558	0.48 / 0.37
hurricane	69.88 / 1.43	2051 / 1143 / 313	3.89 / 0.43
ionization	87.56 / 2.06	2320 / 1151 / 714	0.46 / 0.61
NOAA climate	62.67 / 2.56	1926 / 1373 / 685	0.57 / 1.26
	126.57 / 1.90	2800 / 1947 / 909	2.44 / 3.1
	57.24 / 2.43	1849 / 1308 / 632	0.89 / 1.43

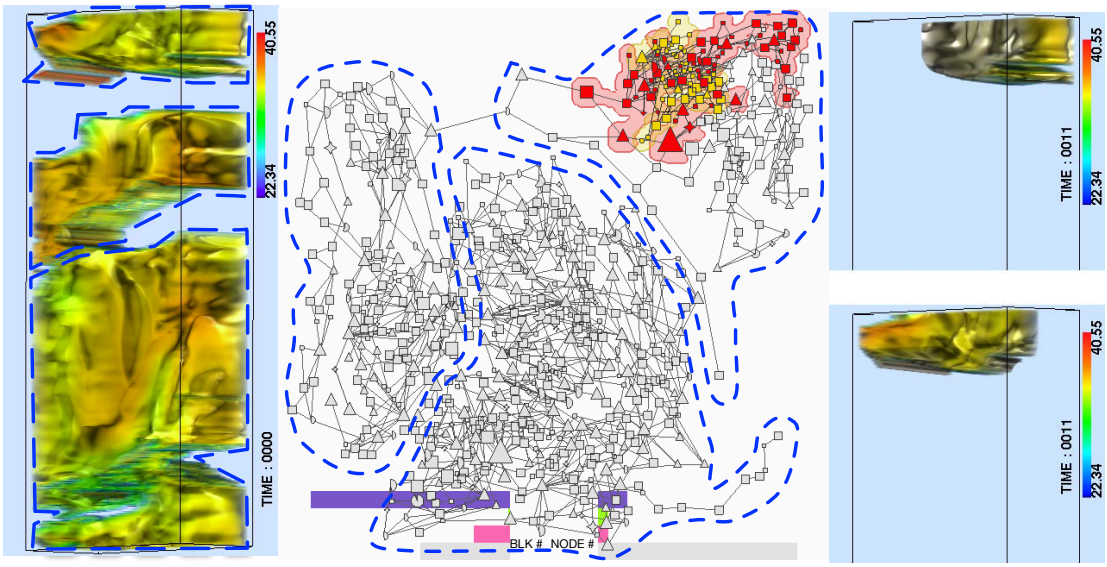
accelerate the performance of state clustering using the quad-core CPU. State clustering and layout computation are the two major tasks, which need to be computed only once per data set. Graph simplification, community detection, and node recommendation were calculated only once and each step took less than one second. Community recommendation was normally completed within a few seconds. The iterations of layout creation, layout adjustment, and node recommendation were set to 500, 150, and 30, respectively, at which point the iterative solutions converge.

4.6.1 DCMIP Climate

The DCMIP climate data sets were produced from the simulations of the Earth's climate in the dynamical core model intercomparison project (DCMIP) [1]. We acquired three simulation data sets generated by different models (cam-fv, cam-se, and fim). In these three models, the volumes are fairly static in the early time steps and later on two turbulent branches appear. We performed cross-comparison of these three data sets.

As expected, the graphs derived from these three data sets are similar, although not entirely the same, as shown in Figure 4.8 (a), (b), and (c). We can see that fans and cliques occupy the central locations in the graphs and they are surrounded by many connectors and regular nodes. Furthermore, at the early time steps, there are two large fans surrounded by many small nodes to form two groups. In the middle or later time steps, the nodes begin to form three groups. The proximity between the three groups in the graph indicates the close interaction among their corresponding volume regions. Since a fan represents the states with interruption and a clique represents a set of nodes that transit among themselves, the presence of a large number of fans and cliques implies that the data sets have a mix of dynamic and static regions. In addition, a large number of connectors indicates that their corresponding regions shift between different states frequently. Figure 4.9 (a) and (b) show the corresponding data blocks of all the fans and cliques of DCMIP cam-fv and DCMIP cam-se data sets, respectively. Fans correspond to the two fast-moving turbulent branches while cliques correspond to the bottom layer with slow movement.

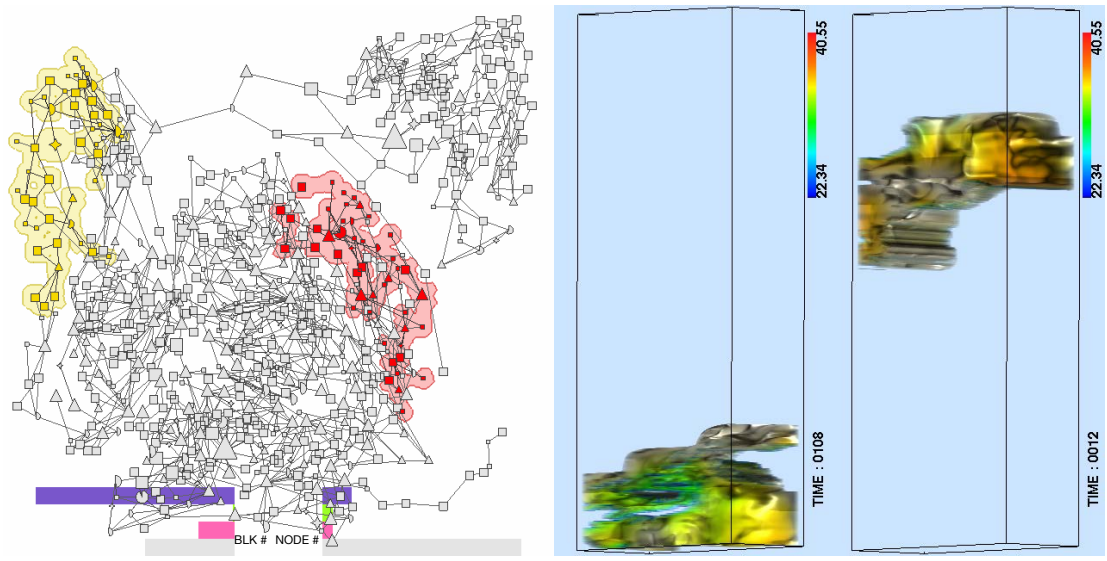
For DCMIP cam-fv and DCMIP cam-se, nodes in the early time steps are fairly cluttered. Although two fans are the centers of the two groups, there are many nodes and edges connecting to them, showing no clear separation. They start to form groups in the middle time steps and node groups get separated more in the later time steps. In addition, a node group that lasts the longest corresponds to the bottom layer and the other two groups corresponding to the two branches appear at different time steps. For DCMIP fim, nodes can be clearly partitioned into two groups in the early time steps and three groups in the



(a) (b) (c)

Figure 4.14. (a) a rendering of the entire NOAA climate data set. (b) and (c): recommending the community (yellow) that is the spatial neighbor of the selected community (red) at the same time step.

middle time steps. Nodes become cluttered in the later time steps. Note that these three node groups appear at the same time step. We first analyze the transition graphs of DCMIP cam-fv and DCMIP cam-se. In Figure 4.8 (a) and (b), cluttered nodes in the early time steps imply that these nodes form separate groups but their differences are not significant. In addition, the nodes in Figure 4.8 (a) begin to form groups in the later time steps. In fact, the lower branch highlighted in Figure 4.9 (a) begins to appear at time step 22. The upper branch does not appear so late, with frequent interaction with the bottom layer. Therefore, the separation between the upper branch and the bottom layer is not clear. In Figure 4.8 (b), the nodes in the early time steps are also cluttered. In the middle time steps, the nodes form two groups. Similar to the graph of DCMIP cam-fv (Figure 4.8 (a)), the lower branch has not appeared yet and the two groups correspond to the upper branch and the rest of regions. In the later time steps, the nodes in the upper branch begin to diverge which leads



(a) (b) (c)

Figure 4.15. (a) to (c): recommending the community (yellow) that is not the spatial neighbor of the selected community (red) at a different time step.

to the nodes corresponding to the lower branch. Therefore, the connections between the upper and lower branches can be noticed. Unlike DCMIP cam-fv, its lower layer becomes distinguishable from the two branches. Therefore, the nodes in the later time steps can form clear groups.

Next, we study the transition graph of DCMIP fim in order to figure out why it is different from the graphs of the other two data sets. In Figure 4.8 (c), there are two fans in the early time steps which naturally form the centers of the two groups. This indicates that the corresponding blocks in early time steps are separated into two groups. One group corresponds to the middle region, and the other group corresponds to the two boundary regions. In the middle time steps, we can see three clear groups as shown in Figures 4.10, 4.11, and 4.12. They correspond to the two turbulent branches and their surrounding. Different from DCMIP cam-fv and DCMIP cam-se, both branches appear at time step 7. That is why the clear separation appears so early in the graph. However, in the later

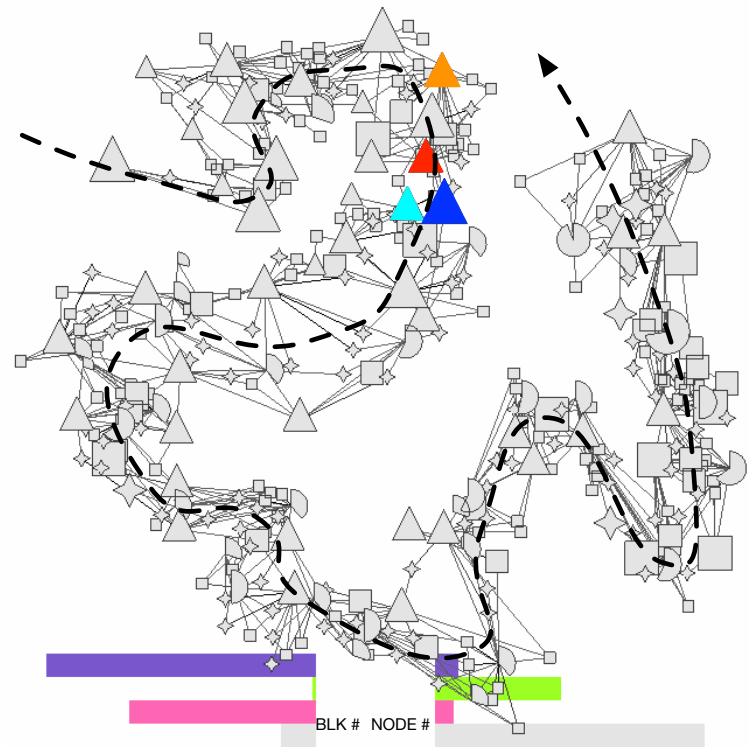
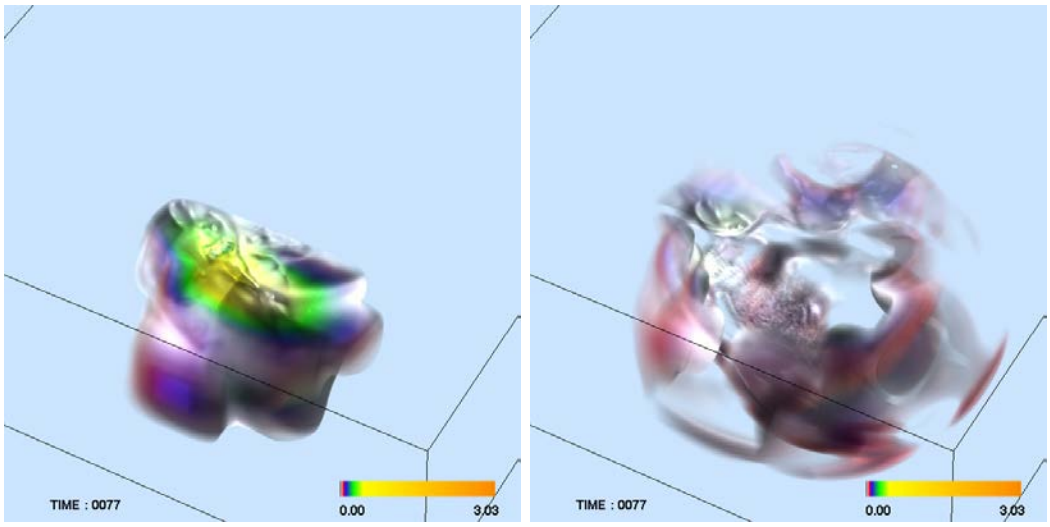


Figure 4.16. four cliques are selected in the transition graph of the earthquake data set. The evolution of time is marked with the dashed line.

time steps, the nodes become cluttered since the two branches are smeared into the rest of regions.

For the DCMIP fim data set, we can differentiate the nodes by examining the communities as shown in Figures 4.10, 4.11, and 4.12. In addition, communities help us explain unusual events. In Figures 4.10 (a), 4.11 (a), and 4.12(a), the communities in the graph correspond to the lower branch, the surrounding, and the upper branch of the volume data, respectively, as shown in Figures 4.10 (b), 4.11 (b), and 4.12 (b). This indicates that the two branches only interact with the surrounding region. This is different in DCMIP cam-fv and DCMIP cam-se data sets, where the two branches have direct interaction with each other. By tracking the communities, we can observe that the two branches in the DCMIP cam-se data set connect to each other. This is highlighted in Figure 4.13. The same conclusion can



(a) (b)

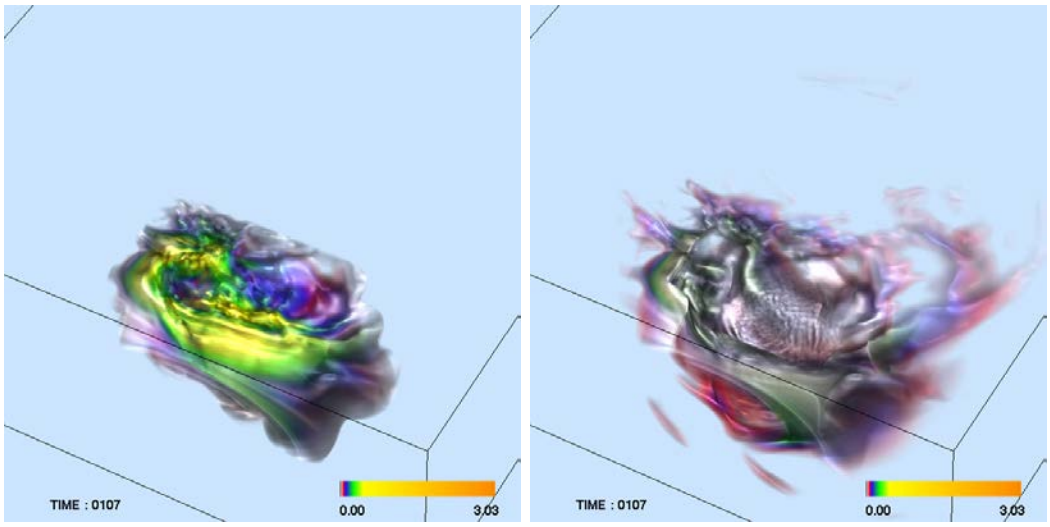
Figure 4.17. (a) and (b) are the rendering of the corresponding data blocks of Figure 4.16: red clique for (a), yellow clique for (b).

be drawn for the DCMIP cam-fv data set.

4.6.2 NOAA Climate

The NOAA climate data set was produced from the NOAA's Geophysical Fluid Dynamics Laboratory (GFDL) CM2.1 global coupled general circulation model. We studied the salinity variable for this data set. A rendering of the volume is shown in Figure 4.14 (a) where the empty regions correspond to the continents. In Figure 4.14 (b), we can see clear separation among three node groups which are highlighted with the dashed boundaries. These three node groups correspond to the three highlighted separate ocean regions in (a). Furthermore, the presence of a large number of cliques implies that data blocks only interact with other blocks in their local neighborhood.

In Figure 4.14 (b), we select a community (shown in red) and the community shown in yellow is recommended. Their corresponding blocks are highlighted in the top and bottom images in Figure 4.14 (c), respectively. We can see that the recommended community



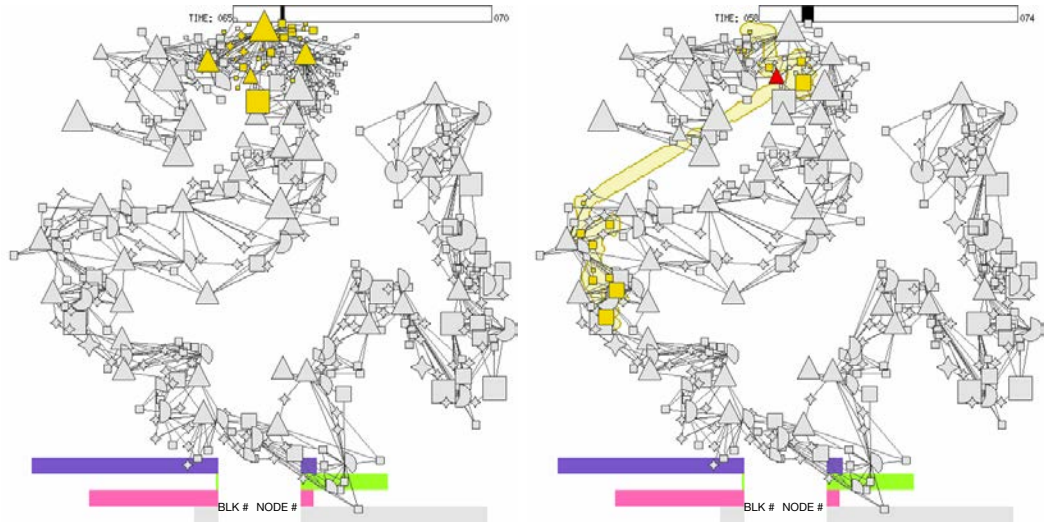
(a) (b)

Figure 4.18. (a) and (b) are the rendering of the corresponding data blocks of Figure 4.16: green clique for (a), and blue clique for (b)

corresponds to the spatial neighbors that are similar to the selected one at the same time step. In Figure 4.15 (a), the selected and recommended communities are in different time steps. Their corresponding blocks are highlighted in Figure 4.15 (b) and (c), respectively. Since the graph structures of two communities are similar, the corresponding data blocks may behave similarly at their respective time step. This provides an interesting cue for scientists to further examine their corresponding regions.

4.6.3 Earthquake

The earthquake data set was produced from a simulation of the 3D seismic wave propagation of the 1994 Northridge earthquake. For this data set, we explored node recommendation. The nodes in Figure 4.16 form a fairly long pattern due to the large number of time steps in the data set. Even after graph simplification, the nodes in the early time steps are denser than the nodes in the later time steps. This indicates more data variation in early time steps, thus more states are created. Furthermore, many cliques occupy the

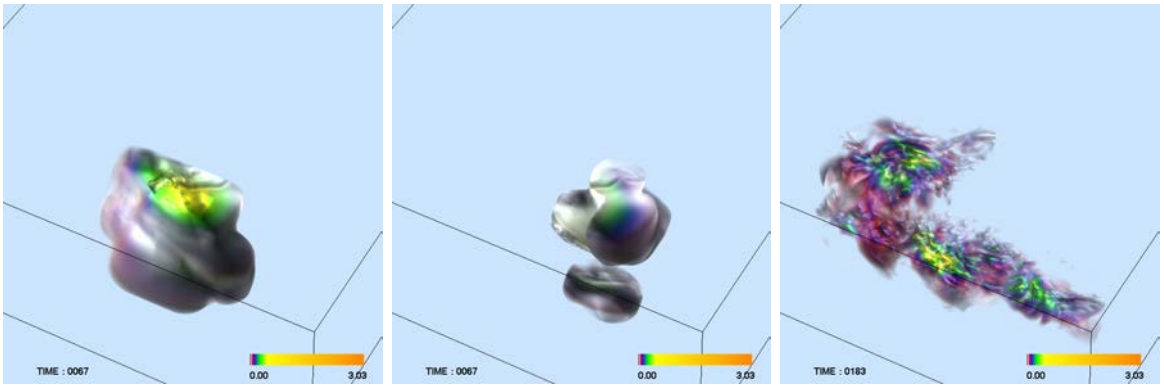


(a) (b)

Figure 4.19. (a) Selecting nodes (highlighted in yellow) overlapping the time range of [65, 70]. (b) a selected clique with the time range of [58, 74] is shown in red and the recommended nodes are shown in yellow.

central locations in the graph. The four cliques highlighted in Figure 4.16 correspond to different volumetric regions in Figures 4.17 and 4.18. Specifically, Figures 4.17 (b) and 4.18 (b) correspond to the inner layers of the earthquake's center, and Figures 4.17 (a) and 4.18 (a) correspond to the outer layers of the earthquake's center.

Since the later growing period is important, we first selected a time range of [65, 70] using the time bar. As shown in Figure 4.19 (a), the nodes in that time span are highlighted in yellow to help users narrow down to the nodes of interest. Then, among these nodes, we select a clique which is highlighted in red, as shown in Figure 4.19 (b). This clique corresponds to the earthquake's center and its immediate surrounding. We considered outgoing transition probabilities for node recommendation. The recommended nodes nearby the selected clique correspond to data blocks at the same time step. These blocks are nearby the data blocks corresponding to the selected clique. The recommended nodes in a later time step still correspond well to the central region of the earthquake.

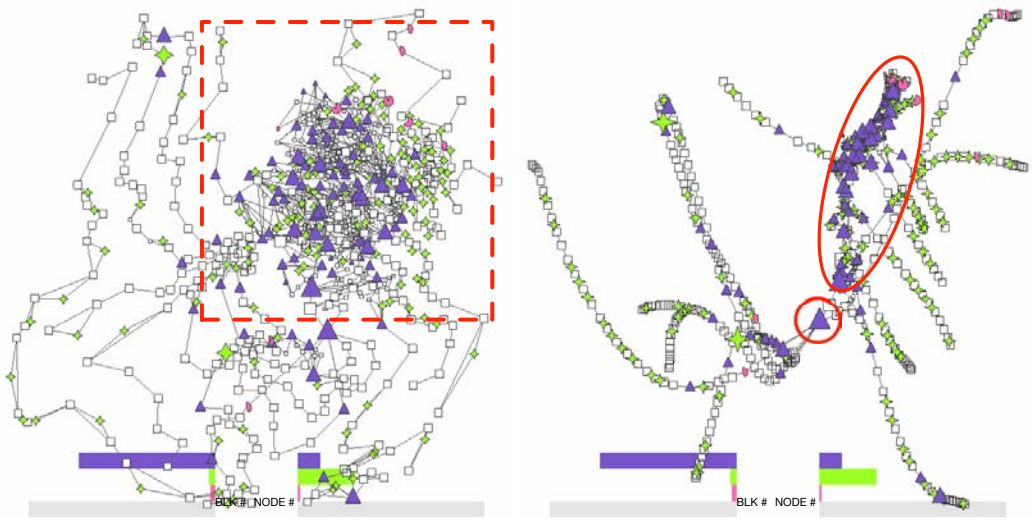


(a) (b) (c)

Figure 4.20. The clique in Figure 4.19 (b) corresponds to the rendering in (a). (b) shows the corresponding rendering of the recommended nodes at the same time step as (a). (c) shows the corresponding rendering of the recommended nodes at a later time step, essentially a tracking result for (b).

4.6.4 Ionization

The ionization data set was produced from a 3D radiation hydrodynamical simulation of ionization front instabilities for studying a variety of phenomena in interstellar medium such as the formation of stars. A small front appears at the beginning time steps. The front moves and grows along the x direction, followed by a cuboid with similar values. Figure 4.21 (b) shows the transition graph. Starting from the clique highlighted at the center of the graph, nodes expand along several directions, each corresponding to a layer of the volume along the x direction. Since there is no interaction between neighboring layers, the nodes in one layer do not have edges connecting to other nodes in another layer. That is why the nodes corresponding to each layer form a single branch in the graph. Different layers appear at different time steps, thus a branch may split into several smaller branches. In addition, it is clear that connectors dominate the graph except one branch. This is quite different from other data sets we explored. For the ionization data set, each layer may have several groups of nodes and there are slight changes in each layer. Although a node in a



(a) (b)

Figure 4.21. In order to better study the front area, the portion of the transition graph marked with a red rectangle in (a) is selected to zoom in. (b) the transition graph before layout adjustment reveals a large number of connectors lying on different branches.

layer may diverge into several nodes, they will converge eventually. Therefore, each group of diverged nodes forms a connector.

Among all the branches shown in Figure 4.21 (b), one of them has more nodes than any other. Highlighted with the ellipse, this branch corresponds to the front of the ionization. Since the front leads the evolution of ionization, it changes most dramatically. Thus the branch of the front has a larger number of nodes than others. Meanwhile, the nodes corresponding to the front are cluttered even after layout adjustment, as shown in Figure 4.22 (a) to (d). This makes it difficult for us to find groups of similar nodes that correspond to different regions of the front. Community detection helps us identify similar nodes through node highlighting. For example, in Figure 4.22 (a) to (d), four communities are selected. Their corresponding volume regions are highlighted in Figure 4.22 (e) to (h). (e) to (g) correspond to three different layers of the front while (h) corresponds to the base.

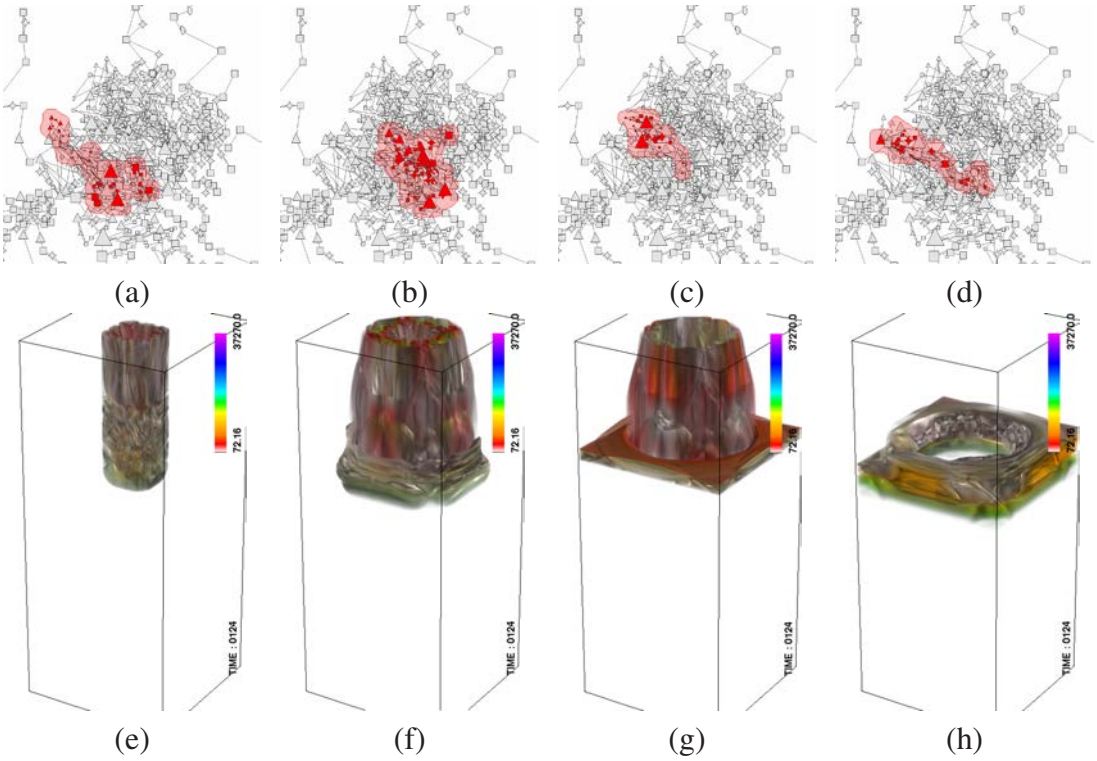
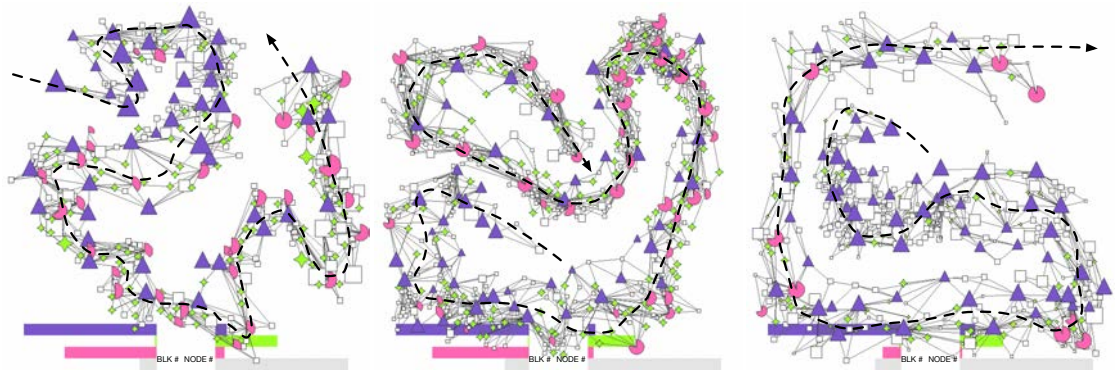


Figure 4.22. (a) to (d) are four communities selected from the transition graph of the ionization data set. In order to better study the front area, the portion of the transition graph marked with a red rectangle in Figure 4.21 (a) is selected to zoom in for (a) to (d). (e) to (h) are their corresponding volume highlighting results.

4.6.5 Parameter Selection

Figures 4.23 and 4.24 show the graph layouts of two data sets under different parameter settings. Comparing Figure 4.23 (a), (b), and (c), we can see that there are many connectors in all three graphs. In addition, there are more cliques and less fans in the early time steps than in the later time steps. Note that the number of fans in Figure 4.23 (c) is less than that in (a) or (b). The large block size set for (c) reduces the differences among blocks, and therefore, the turbulent events become less significant, leading to less fans identified. In Figure 4.24 (a), (b), and (c), we can see clear separation of the nodes which correspond to the three regions in Figure 4.14 (a). Therefore, although different sets of



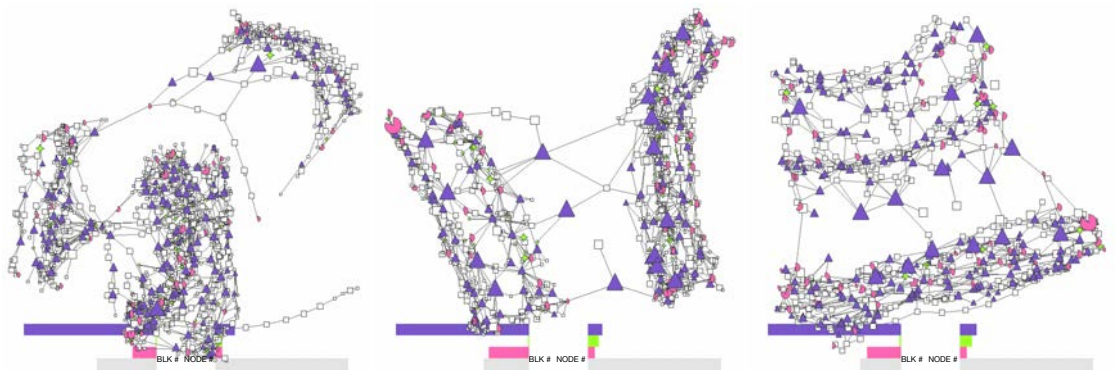
(a) (b) (c)

Figure 4.23. The graph layouts with different parameter settings. (a), (b), and (c) are for the earthquake data set with block sizes $16 \times 16 \times 16$, $16 \times 16 \times 8$, and $32 \times 32 \times 16$, respectively. The evolution of time is marked with the dashed line.

parameter produce different graph layouts, the structure of the graphs after simplification remains almost the same. In other words, the graph structure is insensitive to the change of parameter values and the randomness of initial node placement. We conclude that the graph is largely determined by the nature of the time-varying data, while parameter changes only introduce slight layout variation.

4.7 Feedback from Domain Experts

We also conducted expert evaluation with two domain experts, Drs. Robert Jacob and Seung Hyun Kim. Dr. Jacob's research focuses on ocean and coupled climate models and the computational, mathematical and theoretical issues involved in their construction. Dr. Kim's research focuses on modeling of multiscale and multiphysics problems in relationship to energy science and technology. Dr. Jacob evaluated our work with DCMIP and NOAA climate data sets and mainly focused on model comparison and graph simplification. Dr. Kim evaluated our work with three other data sets and focused on graph simplification, community detection, and visual recommendation. We utilized the think-



(a) (b) (c)

Figure 4.24. The graph layouts with different parameter settings. (a), (b), and (c) are for the NOAA climate data set with block sizes $15 \times 11 \times 9$, $9 \times 11 \times 9$, and $15 \times 6 \times 9$, respectively.

aloud protocol during the evaluation. The experts described their seeing, doing, feelings and comments, and we summarized their comments after the evaluation.

Dr. Jacob commented that our approach is novel and useful. In model comparison, transition graphs abstract the original volumetric data sets and the differences between them allow him to infer the differences between models. For example, he could notice the natural node groups to identify the time steps corresponding to the branching. This is a significant difference between the models being compared. In addition, through operations such as brushing and linking, he could find out that the two branches are similar in one model while they are very different in the other two models. He further pointed out that our work is interesting and useful as it is able to detect the regions where the two branches interact with each other. He also mentioned that graph simplification is beneficial and the level of simplification is appropriate for users to understand the underlying graph structure. Finally, Dr. Jacob commented that it would be helpful if we could provide brushing in the volume view and link the results back to the graph view. Since one of his main interests is studying the influence among oceans, this would allow him to investigate the behavior of

a particular ocean of interest.

Dr. Kim stated that our tool is useful and easy to use. For graph simplification, he mentioned that when the original graphs are complex, it is difficult to identify their structures. The simplification function not only simplifies the view of the graphs, but also helps identify the underlying structures. However, in some cases, even after simplification, the transition graphs are still complex and users could not recognize their structures. He suggested us to leverage focus+context techniques to highlight nodes at the current time step in a less cluttered view. This will make the nodes and their relationships more visible, and thus easier to be selected. Dr. Kim also suggested two ways to further classify the simplified graph. First, we may want to relax cliques to dense subgraphs (quasi-cliques) so that more simplification can be achieved. Second, we may also consider edge weights in the simplification for generating stronger classification results. Dr. Kim studied the cliques in the earthquake data set and the connectors in the ionization data set. He mentioned that a clique represents a cluster of related nodes and it has a *circular* relationship. Therefore, a clique corresponds to a stable state because we would see this structure for a period of time. Unlike cliques, connectors represent a *propagation* of wave-like structures. To better investigate the transition graph after simplification, he suggested us to provide a function to expand fans, connectors and cliques so that users could see more detailed structures.

For community detection, Dr. Kim agreed that it is meaningful because it collects nodes of similar characteristics, which simplifies graphs and helps users explore volume data. He suggested us to consider a maximal time period for each community for better community detection. Since different communities cover different time intervals, we should provide feedback on the time spans of communities. For example, if users are interested in a certain time step, we could highlight all the communities that overlap with this time step. Furthermore, when users select a certain community, we should provide its time duration information so that users can quickly narrow down to time steps of interest. We address this problem by adding a time bar which can be used to filter not only communities, but

also nodes. In addition, we could provide a community importance measure based on their numbers of blocks or time periods to emphasize or deemphasize the communities that are more or less important. Since communities with many blocks usually span large time periods, we address this problem by assigning the importance of each community based on its block number. Dr. Kim mentioned that node and community recommendations help identify similar structures in the data. He suggested us to encode and display the similarity information in nodes or communities when a node or community is selected.

Finally, Dr. Kim commented that our work may be helpful for investigating turbulent combustion data sets. In turbulent combustion, large-scale coherent structures play an important role in determining overall flame characteristics. Such structures are large scale, organized, and propagated downstream, with small-scale turbulent motions being superimposed. Layers of high reaction rates are often associated with these large-scale structures. Cliques, connectors and fans can be used to quickly identify coherent structures. In addition, graphs for turbulent combustion data sets may be very complex due to the chaotic nature of turbulence. Graph simplification is expected to help the exploration of such data sets. Community detection and recommendation will also help explore the evolution of coherent structures or identify similar structures.

Both domain experts suggested that we could support multivariate data sets. Dr. Jacob pointed out that in ocean data sets, salinity and temperature values influence each other. Therefore, it would be helpful if we could investigate these two variables simultaneously. Dr. Kim mentioned that we could support both model comparison and variable comparison. We could generate the transition graphs for different variables and enable users to investigate the differences between graphs.

CHAPTER 5

ITREE: AN INDEXABLE TREE FOR EXPLORING TIME-VARYING DATA

5.1 Overview

In this chapter, we introduce iTree which integrates data compacting and indexing into tree-based visual interface for time-varying data visualization and exploration. We first transform the *time-activity curves* (TACs) representation of a time-varying data set into a hierarchical representation named *symbolic aggregate approximation* (SAX). Then an indexable version of the data hierarchy named iSAX is constructed, from which we create the iTree for visual representation of the time-varying data. Finally, a hyperbolic layout algorithm is employed to draw the iTree with a large number of nodes and provide focus+context visualization for interaction. We achieve effective querying, searching and tracking of time-varying data sets by enabling multiple coordinated views consisting of the iTree, symbolic view, and spatial view.

5.2 SAX and iSAX

The simplest way to construct SAX/iSAX representations from a time-varying data set is to treat each voxel over time as a TAC. However, it may not be cost-effective for a large time-varying data set with a large number of voxels and/or a large number of time steps. Since we will visualize the iSAX hierarchy, it is important to effectively reduce the number of SAX words constructed. One solution is to organize spatially neighboring voxels into a group (such as $2 \times 2 \times 2$) and rearrange their corresponding TACs time step by time step (i.e., going through voxel by voxel for the first time step, then the second etc.), treating

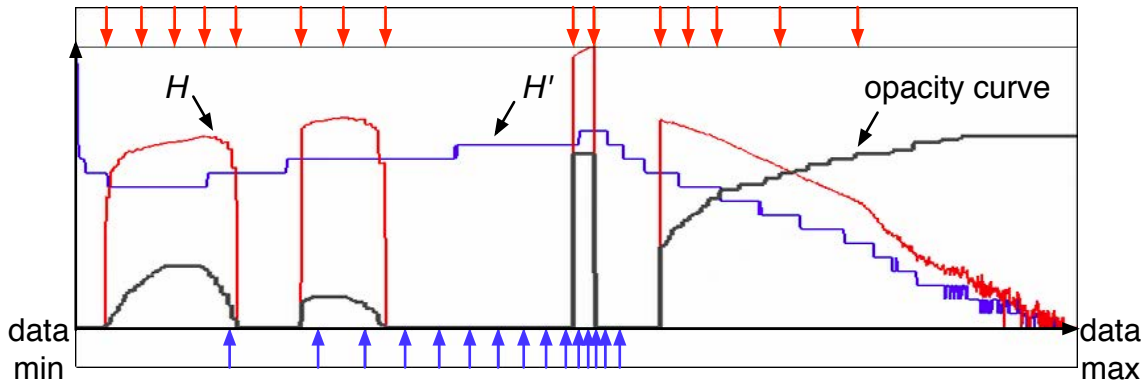


Figure 5.1. H' is the histogram after logarithm and normalization of the original histogram of the earthquake data set. H is the new histogram which results from multiplying H' by the opacity value. Blue and red arrows indicate the breakpoints determined by the original histogram and H, respectively.

it as the TAC for the entire group. Furthermore, breaking the long time steps into time intervals would allow us to cluster data with finer temporal granularity. Therefore, to strike a good balance, we suggest to use group-wise voxels in conjunction with time intervals for generating input TACs. SAX has been applied to detect frequent or outlier patterns for time-varying data [63]. We modify the original algorithms [83, 109] to accommodate the characteristics of time-varying volumetric data in order to better differentiate SAX words, improve computation efficiency, and reduce the number of nodes shown in the iTree.

5.2.1 SAX

A SAX word is a symbolic aggregate approximation of a TAC, which reveals the trend of the TAC and allows indexing in iSAX. A SAX word can be represented by symbols (e.g., a, b, c and d etc.) or bits (e.g., 00, 01, 10 and 11 etc.). SAX achieves saving in storage by dimension reduction of time steps and bit compression.

Before transforming a TAC to a SAX, the user first specifies the value dimension α which indicates how many levels the entire value range will be partitioned into. It is es-

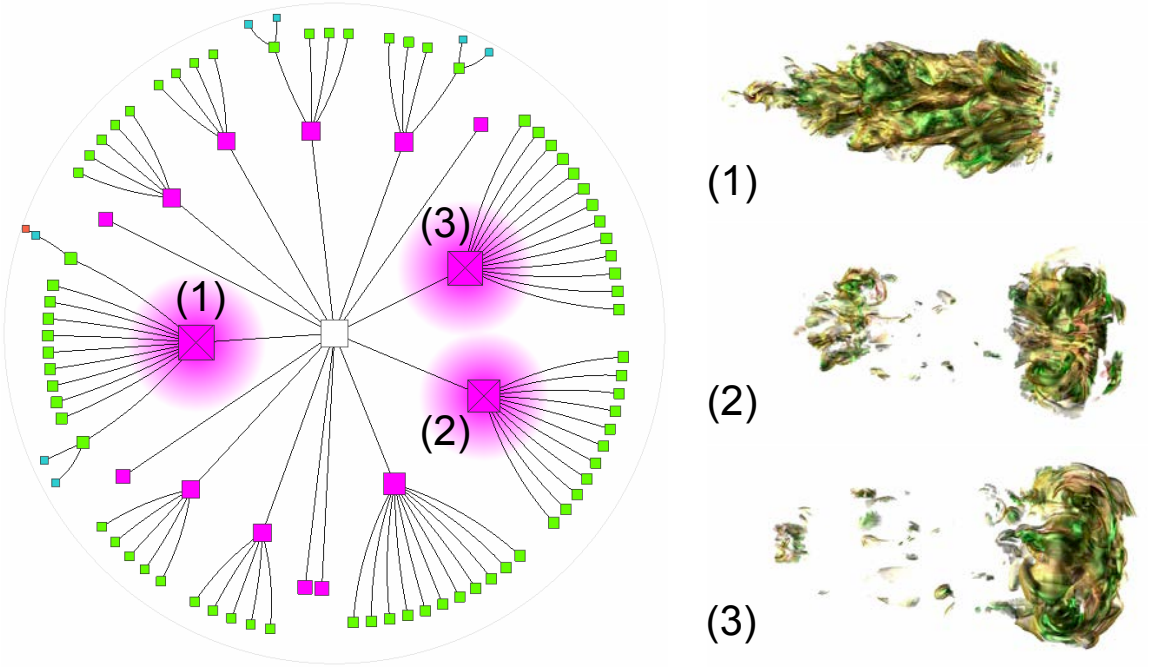


Figure 5.2. The iTree of the argon bubble data set generated using the original algorithm [83, 109]. Three main clusters are highlighted in the volume at time step 160.

sential to appropriately choose the "breakpoints" that determine the partitioning. Lin et al. [83] normalized each TAC to have zero mean and unit variance. Assuming the value distribution of a normalized TAC fulfills the Gaussian distribution, they determined a set of breakpoints by evenly partitioning the area covered by the distribution. Normalizing a TAC may better differentiate the value difference but the information of the original value range is lost. Moreover, solely considering the values may lead us to distinguish the value ranges not even showing up in the transfer function (i.e., the corresponding opacity is zero), which is not desirable either.

To resolve these issues, we present our transfer function based strategy to choose the breakpoints. Specifically, we first convert the TACs to the piecewise aggregate approximation (PAA) representations, then construct a global histogram H' based on all PAA

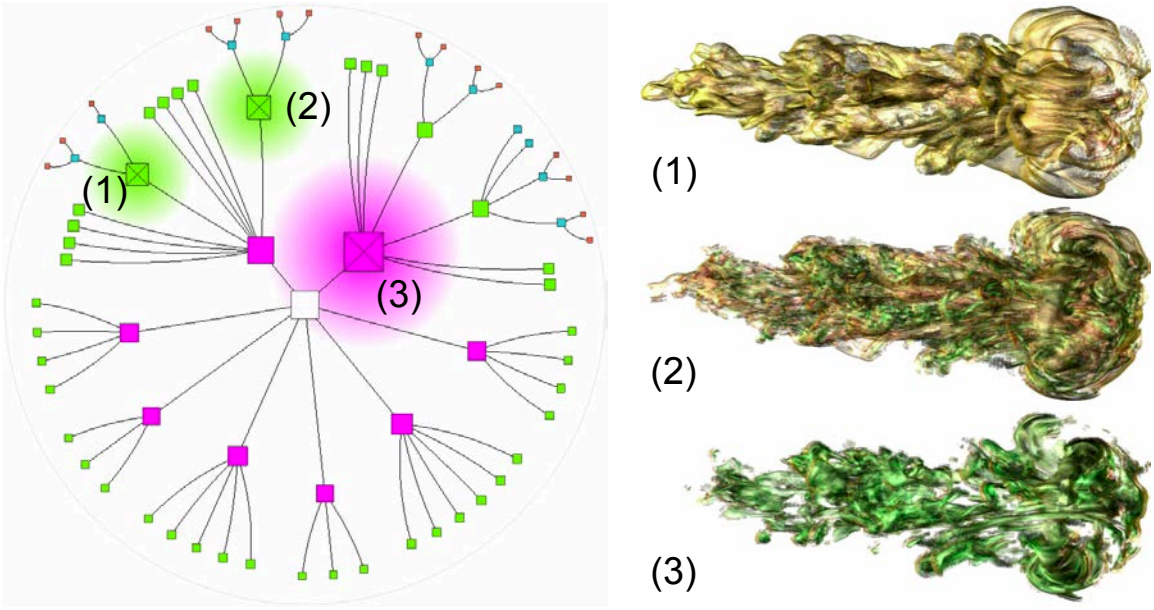


Figure 5.3. The iTree of the argon bubble data set generated using our algorithm with new schemes for breakpoint identification and symbol splitting. Three main clusters are highlighted in the volume at time step 160.

representations. After that, we combine H' with the transfer function to generate a new histogram H. From H, we calculate a set of breakpoints and finally apply a further transformation to obtain discrete symbolic representations.

PAA Conversion. Given a TAC T with length of n, we convert T into a PAA by compressing n values into w dimensions. The ith element in the PAA representation C is calculated as

$$C[i] = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} T[j], \tag{5.1}$$

In practice, n does not need to be an exact multiple of w and we modify the summation to adopt fractional values when w does not evenly divide n.

Breakpoint Identification. From the PAA representations, we construct a fine-grained global histogram H'. Figure 5.1 illustrates such a histogram H' and the corresponding

opacity transfer function. We treat the opacity as the weight for each bin and generate a new histogram H. Given H and the value dimension α, we find a set of breakpoints to mark those ranges with zero opacity and evenly partition the rest of H. We first identify the continuous value ranges with zero opacity. For each of these ranges, we assign two breakpoints to mark the beginning and ending data values. If the beginning (ending) value is the minimum (maximum) value of the data set, we remove this breakpoint. Assuming m and n are the numbers of value ranges with zero and non-zero opacity respectively, we have three cases. If $n+m=\alpha$, the algorithm stops. If $n+m<\alpha$, we successively pick the value range covering the largest area in H and split it into two ranges with equal area. The value for splitting marks the new breakpoint. If $n+m>\alpha$, we successively choose the two neighboring non-zero opacity ranges with the smallest zero opacity value range in between to merge. The beginning and ending values of the new range after merging are the breakpoints left.

In Figure 5.1, the final breakpoints determined by H' and H are marked with blue and red arrows, respectively. As we can see, our transfer function based solution not only avoids focusing on the value ranges with zero opacity, but also better balances the frequencies of all SAX symbols in the final representation. Figures 5.2 and 5.3 show a comparison and it is clear that using our transfer function based solution leads to a more meaningful layer-by-layer data classification as shown in Figure 5.3.

SAX Word Generation. Once the breakpoints are obtained, we construct an alphabet Φ and transform C into an array of symbols $\hat{C}$ to form the SAX word. We assign the symbol $\hat{C}[i]$ by comparing $C[i]$ with the value ranges bounded by breakpoints β_{j-1} and β_j

$$\hat{C}[i]=\Phi_j, \text{ iff } \beta_{j-1}\leq C[i]<\beta_j. \tag{5.2}$$

Given two symbols $\hat{C}[i]$ and $\hat{C}[j]$ with the same value dimension α, the distance between them is defined as

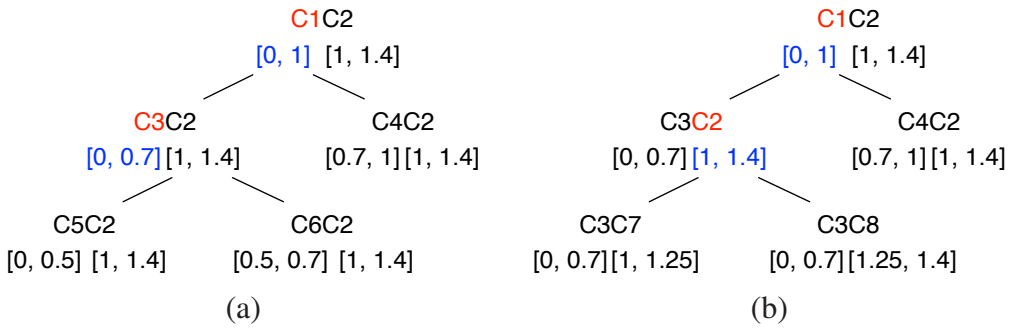


Figure 5.4. Comparing SAX symbol splitting using our algorithm (a) and the original algorithm [83] (b). Red indicates which symbol to split and blue indicates the largest value range.

$$d(\hat{C}[i],\hat{C}[j]) = \begin{cases} 0, & \text{if } \left|\hat{C}[i]-\hat{C}[j]\right| \leq 1 \\ \beta_{\max(\hat{C}[i],\hat{C}[j])-1} - \beta_{\min(\hat{C}[i],\hat{C}[j])}, & \text{otherwise} \end{cases} \tag{5.3}$$

where $\hat{C}[i]$ and $\hat{C}[j]$ are the indices of the corresponding symbols that are between 0 and $\alpha - 1$.

The distance between two SAX words $\hat{C}_1$ and $\hat{C}_2$ is defined as

$$d(\hat{C}_1,\hat{C}_2) = \sqrt{\frac{1}{w}\sum_{i=1}^{w}\left(d(\hat{C}_1[i],\hat{C}_2[i])\right)^2}. \tag{5.4}$$

Note that the distance measure defined based on two SAX words is the lower bound of the Euclidean distance defined based on the PAA representation of the original time series. This allows us to transform TACs into vectors of discrete symbols and work on these symbols directly to speed up the performance.

5.2.2 iSAX

After converting all TACs to SAX words, we build the iSAX hierarchy through clustering. In the hierarchy, a node represents a set of TACs with the same or similar SAX words. When the number of SAX words at a node exceeds a certain threshold δ_n, the node is split into two children with bit promotion of a symbol. This is to keep the number of SAX words corresponding to each terminal node small enough for efficient search.

The key of this process is to find the proper symbol to split. Shieh and Keogh [109] always chose the symbol with the left-most smallest bit cardinality to split, which guarantees that the difference between the largest and smallest bit cardinalities in each SAX word is less or equal to 1. We choose a symbol covering the largest value range to split in order to maximize the difference between SAX words. Figure 5.4 illustrates a comparison using our algorithm and the original algorithm. After two iterations, our algorithm has three SAX words C_5C_2, C_6C_2 and C_4C_2 (i.e., leaf nodes of the hierarchy). The difference between C_5C_2 and C_4C_2 is $\sqrt{((0.7-0.5)+0)/2} = 0.32$. The other two differences are both zero because the corresponding SAX symbols are either the same or have neighboring value ranges. The three SAX words generated by the original algorithm are C_3C_7, C_3C_8 and C_4C_2. The differences among them are all zero. Thus, we can see that our algorithm maximizes the distances of the SAX words and minimizes the largest value range. Figures 5.2 and 5.3 show a comparison and we can see that our symbol splitting scheme leads to a more balanced data classification result. It also produces a less number of nodes in the iTree which we will create from the iSAX hierarchy.

Bit Promotion and Reduction. Let us assume all the symbols s in a SAX word range from 0 to $\alpha - 1$ where α is the value dimension and $|\alpha| = \lceil \log \alpha \rceil$ is the bit cardinality. We write a SAX word with its symbol value and the corresponding bit cardinality in a particular form. For example, $0_2.2_2.6_3.3_2$ is a SAX word consisting of four symbols, 0, 2, 6 and 3, and the bit cardinalities are 2, 2, 3 and 2, respectively. We can also write the symbol values in bits, i.e., 00, 10, 110 and 11 for 0_2, 2_2, 6_3 and 3_2, respectively.

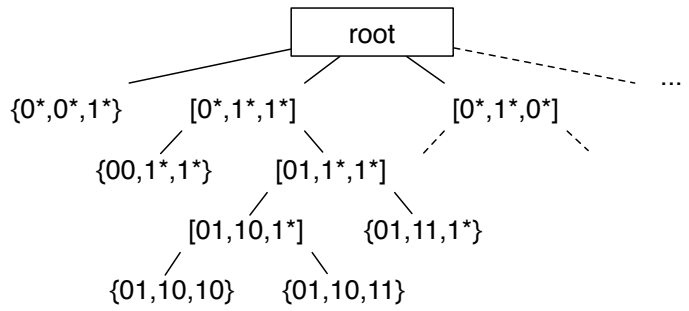


Figure 5.5. An illustration of an iSAX index. Internal nodes and terminal nodes are denoted with [ ] and { }, respectively.

When comparing two SAX words, we actually compare their corresponding symbols in order. If the bit cardinalities of the two symbols are different, we can promote the bits with the smaller cardinality to the same as the larger one. Given two symbols s and t and the bit cardinality of s is less than that of t, we treat s as $s*$. There are three cases to consider. First, if s is the prefix of t, $*$ are the same as the corresponding bits in t for all unknown bits. Second, if s is lexicographically smaller than the corresponding bits in t, $*$ are all 1s for the rest unknown bits. Third, if s is lexicographically larger than the corresponding bits in t, $*$ are all 0s for the rest unknown bits. The guiding principle for these cases is to promote the bits in s so that s and t are closest in the SAX space. After bit promotion, we apply Equation 5.4 to compute the distance between these two SAX words.

Besides bit promotion, we can also reduce the bits of the symbol with the larger cardinality to the same as the smaller one. If the reduced SAX word is the same as the SAX word of the lower one, we consider these two SAX words as a match. In practice, *bit promotion* is used for SAX word comparison in search, while *bit reduction* is used for building the iSAX hierarchy.

iSAX Construction. To construct the iSAX hierarchy, iSAX 2.0 [19] uses the raw TACs as input and loads the TACs into main memory till almost full. In main memory it converts each TAC into a SAX word, then places the SAX words into terminal nodes. At

last it flushes them into disk and clears main memory. After each iteration, it continues to load the rest of TACs. Thus the benefits of iSAX 2.0 are that it reduces disk access, eliminates a second loading for each TAC, and provides the ability of handling large data sets. In our algorithm, we partition this process into two parts. First, we convert all the TACs to SAX words, then insert all SAX words one by one into the hierarchy. Our method not only offers the benefits of iSAX 2.0, but also increases the performance with GPU calculation and prevents system crashes by storing intermediate data into files.

As sketched in Figure 5.5, starting from the root whose children have the same property that the cardinality of each symbol is 1, we compare the input SAX word with these children. If the input SAX word matches one child c that is an internal node, we continue to check c's children in the next level of the hierarchy. Otherwise, we place the SAX word as a member of c. When the member of SAX words placed in a node c is larger than δ_n, we split c into two child nodes. In practice, we choose the symbol that covers the largest value range. For example, given a SAX word $2_2.3_2.3_2.1_2$, let us assume 2_2 covers a range of 0.5 while both 3_2 and 1_2 cover a range of 0.1. We split the SAX word into two: $4_3.3_2.3_2.1_2$ and $5_3.3_2.3_2.1_2$. Let us further assume 4_3 covers a range of 0.3 and 5_3 covers a range of 0.2. If we need to further split $4_3.3_2.3_2.2_2$, the two new SAX words will be $8_4.3_2.3_2.2_2$ and $9_4.3_2.3_2.2_2$. Finally, for each terminal node, we store all its corresponding voxel IDs into a file, using the SAX word itself as the file name to facilitate the subsequent search.

Acceleration Strategy. The above iSAX hierarchy construction algorithm has two limitations. First, each SAX word is inserted into the hierarchy in a sequential order, making it not amenable for parallel processing. Second, the entire iSAX hierarchy along with all voxel IDs needs to be stored in memory, making it not well scalable for handling large-scale data sets. We therefore propose an out-of-core algorithm for iSAX hierarchy construction to address these limitations. Our main idea is to first partition all voxels or groups into at most 2^w buckets, where w is the word length of their SAX words. We save each non-empty bucket into a file, corresponding to all the non-empty nodes in the

first level from the root of the hierarchy. After that, we choose the file with the largest voxel/group count. If the voxel/group count is larger than δ_n, then we split it into two files following our symbol splitting scheme. We continue this process until there is no file which has its voxel/group count larger than δ_n. This algorithm performs file splitting in an out-of-core fashion, thus is more memory efficient. In practice, for each SAX word, we only need to load one symbol involved for splitting. In addition, this improved solution is flexible with the change of δ_n as only the splitting step need to be recomputed, which is not available with the previous solution.

5.2.3 Approximate Search and Exact Search

With the iSAX hierarchy built, we are able to perform approximate search and exact search. Both searches take the PAA representation of a voxel or voxel group and a threshold δ as the input. We find similar voxels or voxel groups with their distance to the input within δ. The approximate search converts the input PAA to a SAX word and compares each of the file names corresponding to the terminal nodes in the hierarchy with this SAX word. If the distance between the file name and the SAX word is larger than δ, we discard that file because the distance between two SAX words is the lower bound of their PAA's Euclidean distance. That is, if the SAX distance is larger than δ, then the Euclidean PAA distance must be larger than δ. Otherwise, we open the corresponding file and return all voxels or voxel groups in the file. The search result is the union of all voxels for all the files opened. The exact search needs an additional step. Instead of simply returning all voxels in the file which has a SAX distance within δ, we actually compute their PAA-based distances to the input PAA and only return those voxels that have a smaller distance less than or equal to δ.

5.3 iTree

The iSAX hierarchy is an *internal* representation of the time-varying data. We construct its corresponding *external* version of iTree for visual presentation and navigation. Unlike

the VizTree [84] which draws the hierarchy as a regular tree, we utilize a hyperbolic graph drawing algorithm to draw the iTree and support interactive F+C visualization. Such a solution nicely organizes a large number of nodes in the iTree within a limited display area and allows the users to query the data in an adaptive manner. In conjunction with the intuitive SAX view, we enable coordinated multiple views to facilitate versatile exploration of time-varying data.

5.3.1 From iSAX Hierarchy to iTree

The original iSAX hierarchy we create is not readily suitable for visualization due to the following reasons. First, the number of non-empty children of the root is fairly large which easily leads to occlusion and clutter when drawing the hierarchy. Second, except for the first level from the root which has a very large fanout, all the internal nodes have exactly two children, which is a fairly small fanout. As a result, the iSAX hierarchy may have a very large number of levels which also makes the drawing of the hierarchy less effective for viewing and exploring. Third, sibling nodes are not arranged in the iSAX hierarchy according to their similarity. Therefore, we propose the following steps: *level promoting*, *sibling grouping* and *sibling reordering* to transform the iSAX hierarchy to the iTree suitable for drawing, navigation and query.

Level Promoting. We first promote each node in the iSAX hierarchy such that its new level in the hierarchy equals the maximal bit cardinality used for any symbol of its SAX word. For example, in Figure 5.5, nodes at the first level under the root remains the same. For the second level, we promote terminal nodes $\{01, 10, 10\}$, $\{01, 10, 11\}$ and $\{01, 11, 1^*\}$ to become the immediate children of node $[0^*, 1^*, 1^*]$. Accordingly, internal nodes $[01, 1^*, 1^*]$ and $[01, 10, 1^*]$ are removed. For this example, since the maximal bit cardinality used for any symbol in a SAX word is two, the height of the iSAX hierarchy after level promoting is three (i.e., three levels corresponding to 0 bit, 1 bit and 2 bits, respectively).

Sibling Grouping. After level promoting, for each non-leaf node, we use a hybrid k-means clustering algorithm [71] to group its children if the number of children n is larger than a certain threshold δ_n. The number of clusters is chosen as $\lfloor\sqrt{n}\rfloor$. We calculate the representative value for each SAX symbol with cardinality $|\alpha|$ as

$$v_i = \begin{cases} (v_{\min} + \beta_i)/2, & \text{if } |i| = 0 \\ (v_{\max} + \beta_{i-1})/2, & \text{if } |i| = \alpha \\ (\beta_i + \beta_{i-1})/2, & \text{otherwise} \end{cases} \tag{5.5}$$

where $v_{\min}$ and $v_{\max}$ are the minimum and maximum values of the data set. β_i is the ith breakpoint. For a SAX symbol $C_{i.a}$ with value i and cardinality a where $a < |\alpha|$, we calculate its representative value $v_{(C_{i.a})}$ as follows

$$v_{(C_{i.a})} = (v_{i \ll (|\alpha|-a)} + v_{(i+1) \ll (|\alpha|-a)-1})/2, \tag{5.6}$$

where $\ll$ is the left shift operation, $|\alpha| - a$ is the difference between the maximum bit cardinality $|\alpha|$ and the current cardinality a. $i \ll (|\alpha| - a)$ and $(i+1) \ll (|\alpha| - a) - 1$ indicate the smallest and largest value ranges that $C_{i.a}$ covers. For example, value 0 with cardinality 1 actually covers 0 (000) to 3 (011) with cardinality 3. Replace i as 0, $|\alpha|$ as 3, a as 1, we get $0 \ll (3-1) = 0$ and $(0+1) \ll (3-1) - 1 = 3$ which are the smallest and largest value ranges that $C_{0.1}$ covers.

We define the distance between two SAX symbols as the distance between their representative values, and the distance between two SAX words as the average distance of their corresponding SAX symbols. For each cluster created, we identify a node that is closest to the centroid of the cluster as its representative. This process is performed in a top-down manner and we replace nodes in each non-root level with their representatives in the actual iTree drawing.

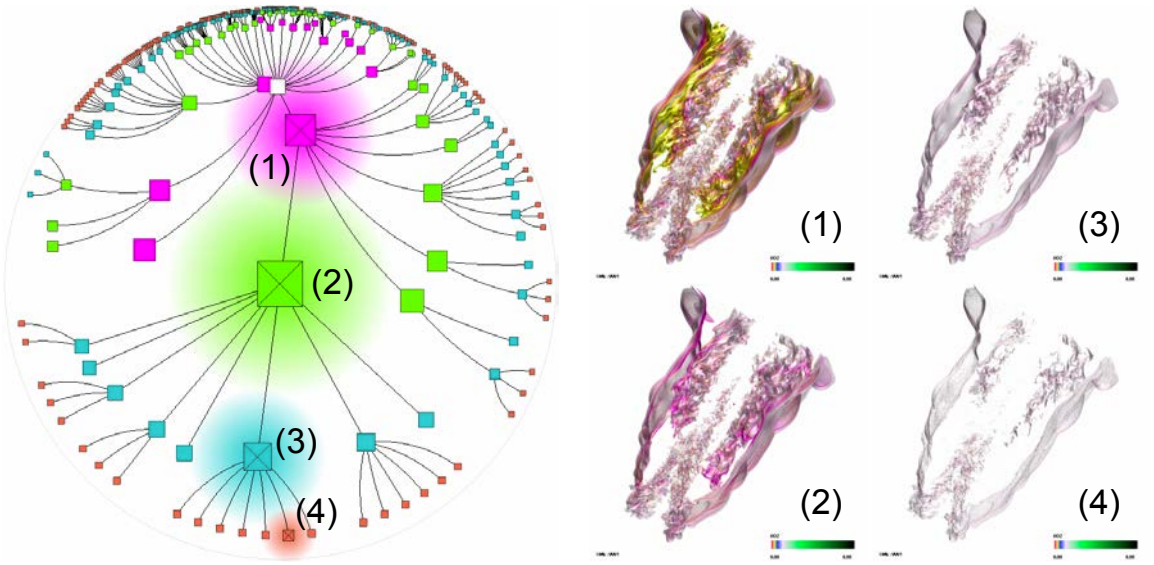


Figure 5.6. Level-of-detail exploration of the iTree of the combustion data set. We mark four nodes (1) to (4) at four different levels of detail in the iTree. The four images to the right show the corresponding clusters highlighted in the volume at the first time step.

After level promoting and sibling grouping, the resulting iTree has the following two nice properties for drawing and querying. First, the height of the iTree is determined by the maximal bit cardinality for representing any symbol in the SAX words. Second, the iTree is balanced in the sense that any node does not have a large fanout. By transforming the iSAX hierarchy to the iTree, we maintain the connection between the internal and external representations (such as node correspondence) so that any further query of the iTree will still follow the underlying iSAX hierarchy correctly.

Sibling Reordering. Finally, we reorder sibling nodes under the same parent in the iTree based on the similarity of their SAX words. When neighboring sibling nodes are selected together, they will form a meaningful group because their corresponding voxels in the volume have a high degree of similarity in terms of spatial closeness and temporal trend. To get the optimal reordering, we need to compute the sum of distances for neighboring sibling nodes and identify the permutation with the smallest sum for every possible

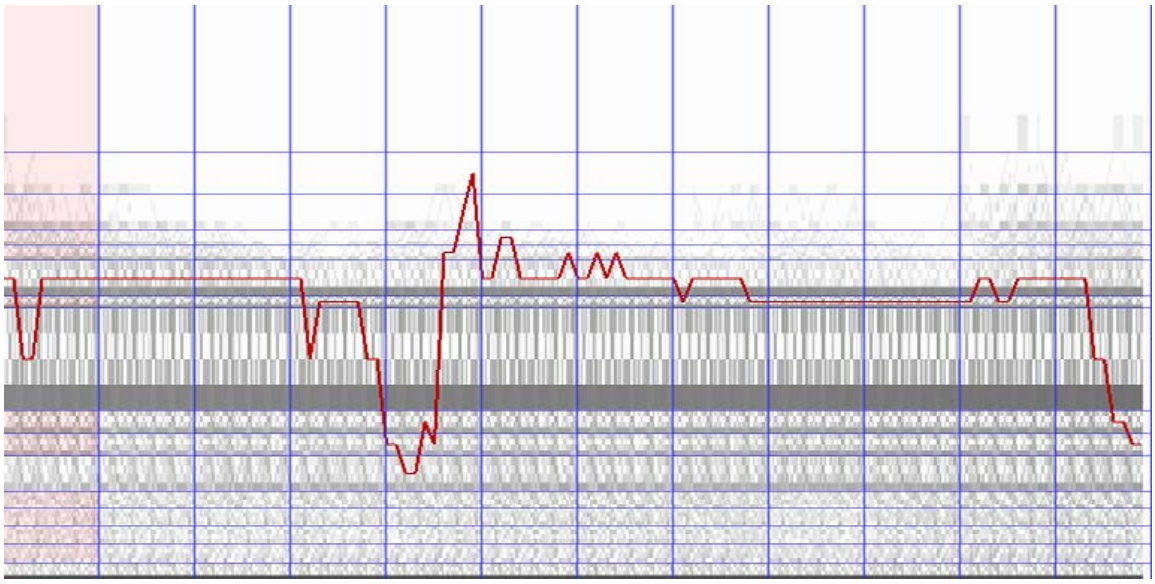


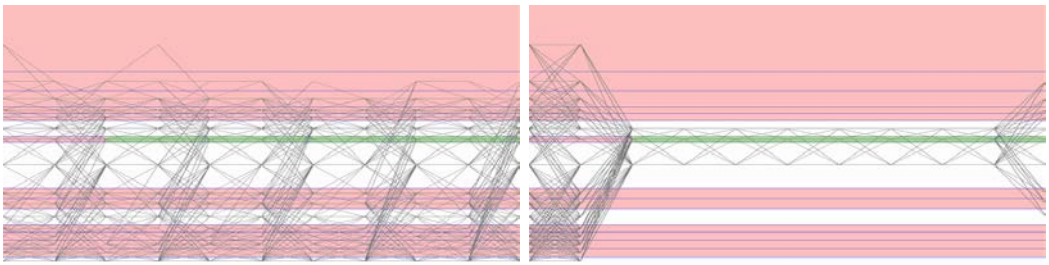
Figure 5.7. Exploring the supernova (entropy) data set using the SAX view. This figure shows the overall density pattern of SAX words with a particular SAX word highlighted.

permutation of sibling nodes.

Since the complexity of optimal reordering follows a factorial growth, we opt for an approximate random swap solution for efficient handling a large number of sibling nodes. Random swap starts with an initial sibling ordering and randomly chooses two nodes to swap their positions. If the new ordering has a smaller sum of distances, then we replace this ordering with the new one; otherwise we keep the old ordering. Then, we randomly select another pair of nodes to swap. We continue this process for at most $2m^2$ iterations or until we have m consecutive random swaps without decreasing the sum of distances, where m is the number of sibling nodes.

5.3.2 iTree Drawing and Focus+Context Visualization

We leverage the hyperbolic layout algorithm introduced by Lamping and Rao [80] to visualize the iTree. The hyperbolic layout organizes a given tree hierarchy within a 2D circle.



(a) zoom into the first time interval (b) SAX filtering result

Figure 5.8. Exploring the supernova (entropy) data set using the SAX view. (a) shows the zoom-in into the first time interval where user selections are highlighted in green. (b) SAX filtering shows the SAX words that are within a distance of 1.0 to the selected green regions.

Since a tree hierarchy tends to expand exponentially with depth, using a circular display provides exponentially more space along its radius to nicely accommodate the increasing number of nodes in successive levels of the hierarchy. To achieve this, the algorithm lays out the tree on the hyperbolic plane and then maps the structure to the Euclidean plane during the display. Compared to the conventional tree layout, the hyperbolic layout can display up to ten times more nodes within the same display region while providing more effective navigation, such as F+C visualization, around the hierarchy. Change of focus is achieved by changing the mapping from the hyperbolic plane to the Euclidean plane, which can be efficiently performed as node positions in the hyperbolic plane remain unchanged. It also allows for smooth blending of focus and context and continuous repositioning of the focus. There are two best known models to map the tree laid out on the hyperbolic plane to the Euclidean plane: the Klein model and the Poincaré model. The former preserves *straightness* of lines while the later preserves *angles* but maps lines in the hyperbolic space into arcs in the Euclidean space. We choose to draw arcs for a more pleasing visualization. Details about implementing the hyperbolic layout and F+C visualization can be found in [80].

TABLE 5.1

PARAMETER VALUES AND TIMING RESULTS FOR GENERATING SAX WORDS

data set	volume dimension	group size	time interval	word length	quant. level	I/O	PAA GPU	BP CPU	SAX GPU
argon bubble	$640 \times 256 \times 256 \times 165$	$2 \times 2 \times 2$	33	10	16	28.27	0.10	9.82	2.09
combustion	$800 \times 686 \times 215 \times 53$	$2 \times 2 \times 5$	18	12	16	23.11	0.35	7.98	1.73
earthquake	$256 \times 256 \times 96 \times 599$	$4 \times 4 \times 3$	50	12	16	8.85	7.25	0.01	0.14
		$2 \times 2 \times 2$	60	10	16	11.00	7.05	8.07	0.67
		$1 \times 1 \times 1$	25	8	16	15.29	7.04	4.05	1.54
hurricane	$500 \times 500 \times 100 \times 48$	$2 \times 2 \times 2$	12	8	16	4.1	2.52	1.99	0.48
supernova (entropy)	$432 \times 432 \times 432 \times 60$	$2 \times 2 \times 2$	12	10	16	25.98	9.50	6.84	1.50
supernova (vel. mag.)	$864 \times 864 \times 864 \times 105$	$2 \times 2 \times 2$	7	10	32	1038.03	407.83	17.78	7.16

TABLE 5.2

PARAMETER VALUES AND TIMING RESULTS FOR CONSTRUCTING ISAX HIERARCHY AND ITREE

data set	δ_n	I/O	bucketing GPU	file splitting GPU	# nodes for iSAX	# nodes for iTree	promoting CPU	grouping CPU
argon bubble	7500	242.55	1.19	0.20	1499	101	0.09	0.07
combustion	50000	596.66	0.98	6.05	4869	357	0.33	0.34
earthquake	5000	1.17	0.14	1.89	441	38	0.03	0.04
	1000	15.49	0.26	0.86	513	46	0.01	0.03
	50000	58.90	1.00	52.23	465	36	0.03	0.04
hurricane	100000	9.42	0.37	0.33	416	76	0.01	0.03
supernova (entropy)	50000	327.54	1.00	4.26	2048	387	0.11	0.08
supernova (vel. mag.)	500000	1381.17	3.92	21.54	1927	600	0.14	0.39

5.3.3 Query in Multiple Coordinated Views

We dynamically link together the three views of the time-varying data, i.e., the volume view, the iTree view and the SAX view. The user interacts with the data in one view and the result is automatically reflected in the other view. To support effective exploration of time-varying data sets, we provide the following queries.

iTree Query. We allow the user to filter out nodes based on their levels in the hierarchy, or the number of voxels they contain. This helps reduce the clutter and facilitates the observation and exploration. Nodes on different levels in the iTree are distinguished using different colors. We map the size of a node to the number of descendants its contains, thus attracting the user's attention to those interesting nodes that are worth exploring. The user can select such a node and press keyboard shortcuts to explore its siblings or ancestors and descendants conveniently. Multiple nodes can be selected for a joint view. The corresponding selected voxels or data regions are highlighted in the spatial volume view. If the TACs are built based on time intervals instead of all time steps, animating over time will reveal how the selected data clusters vary over space and time. In addition, when the user select a node, we draw a time ring to indicate the time intervals it covers.

SAX Query. Complementing the abstract iTree view, the SAX view allows the user to operate in an intuitive manner. We enable F+C visualization so that the user can closely exam the details in a particular value range or time interval. To combat the dense drawing of a large number of SAX curves, we draw quads with the binning of SAX value and time interval combination to show an overview of SAX curve distribution. The density (i.e., color saturation) show the number of curves falling into a bin. The user can brush bins to select SAX curves of interest and make connection to the volume view and the iTree view. We also show SAX curve clustering results so that the user can easily identify the different temporal trends exhibited in the time-varying data.

Volume Query. We allow the user to select a voxel or a group of voxels of interest from the volume at a certain time step (by bounding the ranges in the x, y and z directions)

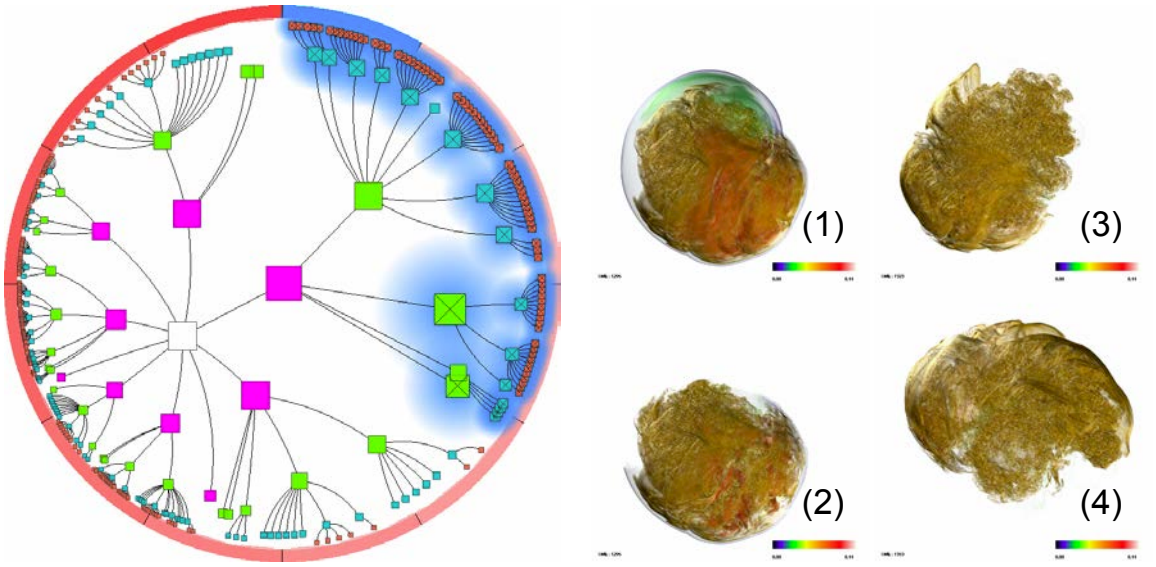
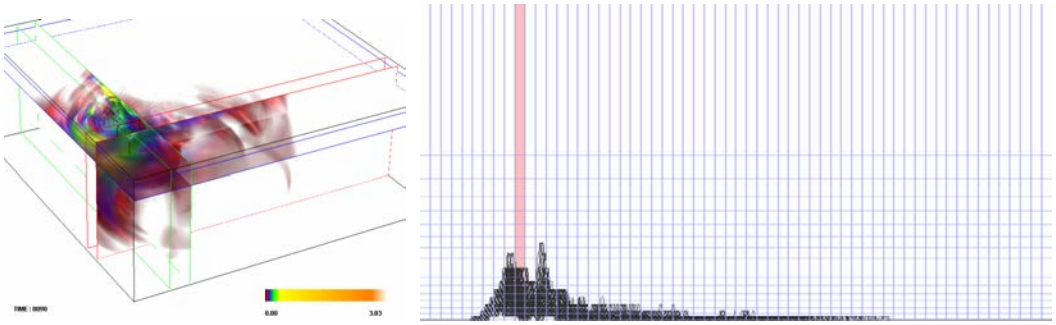


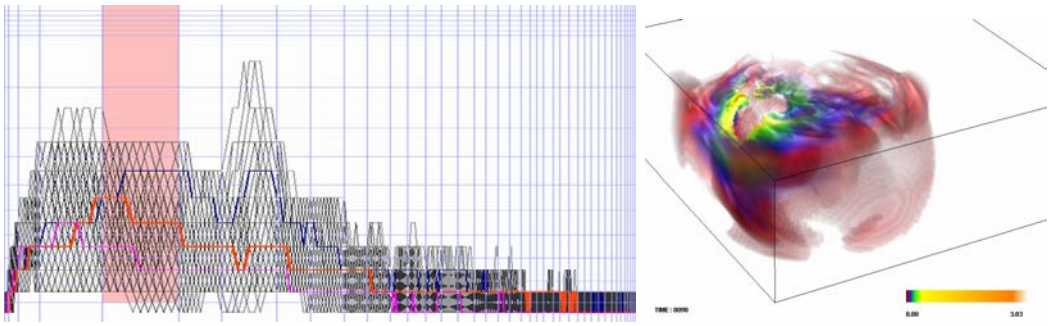
Figure 5.9. Exploring the supernova (entropy) data set using the iTree and volume views. We show the iTree with the corresponding nodes highlighted according to SAX filtering. (1) shows the full volume rendering at time step 1295. (2) to (4) show the corresponding volumetric region and the tracking results at three selected time steps: 1295, 1323 and 1353.

and search for similar voxels in the time-varying data. Two kinds of search using the iSAX hierarchy are supported: approximate search and exact search (Section 5.2.3). The search results can be highlighted in the iTree and in the volume. Again, the user can animate over time to reveal how the similar data distribute over space and time. Leveraging the iSAX hierarchy, indexing and searching the time-varying data is much faster than the brute-force solution without indexing.



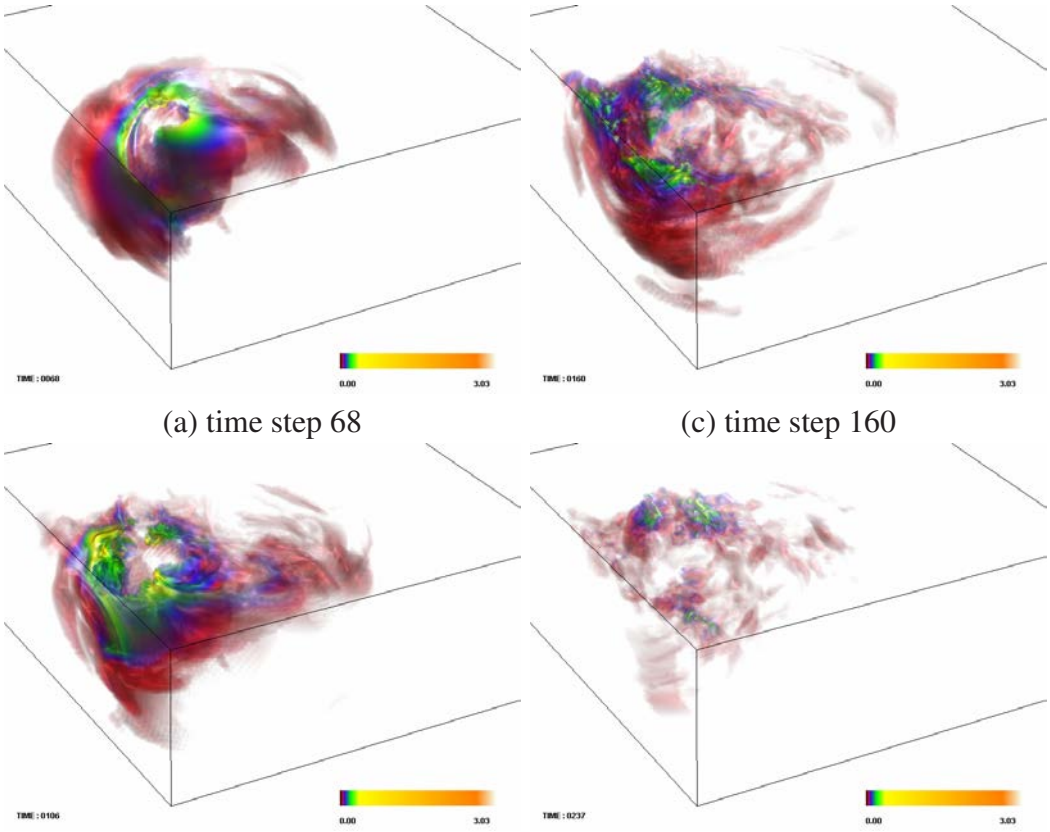
(a) time step 90, group selection (b) initial SAX view of the selected voxel group

Figure 5.10. Querying the earthquake data set. (a) a group of blocks at time step 90 are selected by bounding two slices along each of the x, y and z axes, respectively. (b) the initial SAX view of the group of blocks.



(a) SAX cluster selection with F+C visualization (b) time step 90

Figure 5.11. Querying the earthquake data set. (a) the F+C visualization with three clusters highlighted and the orange cluster selected. (b) is the query result where the SAX words are within a distance of 4.0 to the selected cluster at every SAX symbol.



(a) time step 68 (c) time step 160

(b) time step 106 (d) time step 237

Figure 5.12. Querying the earthquake data set. (a) to (d) are the query results where the SAX words are within a distance of 4.0 to the selected cluster at every SAX symbol.

TABLE 5.3

TIMING PERFORMANCE COMPARISON AMONG BRUTE-FORCE (BF) SEARCH, EXACT SEARCH AND APPROXIMATE SEARCH

data set	query location (x, y, z, t)	δ	BF time	exact time	approx. time	BF time	exact time	approx. time
			current interval			all time steps		
argon bubble	(99, 59, 59, 138)	0.103	0.44	0.23	0.14	12.10	6.81	0.13
combustion	(286, 152, 37, 10)	0.000002	1.14	0.34	0.18	45.77	8.22	0.18
earthquake, $4 \times 4 \times 3$ group size	(26, 52, 29, 66)	0.103	0.13	0.01	0.00	8.40	0.30	0.00
earthquake, $2 \times 2 \times 2$ group size	(51, 104, 45, 70)	0.094	1.92	0.04	0.01	76.54	6.20	0.50
earthquake, $1 \times 1 \times 1$ group size	(113, 204, 86, 68)	0.018	0.27	0.14	0.13	8.06	1.46	0.14
hurricane	(91, 31, 20, 48)	0.00071	0.41	0.29	0.07	13.53	11.01	0.06
supernova (entropy)	(108, 120, 97, 1312)	0.001	1.42	0.41	0.25	44.96	6.68	0.25
supernova (vel. mag.)	(191, 282, 153, 22)	0.015	3.20	2.42	2.08	49.27	22.83	2.23

5.4 Results and Discussion

Data Sets, Parameter Setting and Timing Performance. We experimented our approach with several time-varying data sets, as listed in Tables 5.1 and 5.2. These data sets range from small (128^3) to large (864^3) in spatial extent and tens of time steps to hundreds of time steps in temporal extent. For SAX word generation, we allowed either a voxel-wise or group-wise setting. Splitting the entire time sequence into intervals was useful, leading to time-dependent clustering results. Choosing different word lengths and quantization levels affected the speed performance. A longer word length has a finer representation of temporal trend and a larger quantization levels has a finer representation of value range. This leads to a better discriminating power in the subsequent data clustering, indexing and searching while increasing the cost to process, store and search the reduced SAX/iSAX representations. Our experience shows that choosing 8 to 12 for word length and 16 or 32 for quantization level are appropriate for quality and speed tradeoff. For SAX hierarchy construction, the key parameter is the threshold for voxel/group count as it determines the time cost for file splitting. Normally, we chose tens of thousands as the threshold. A larger threshold leads to a smaller number of nodes produced for the iSAX hierarchy. For iTree construction, we were able to reduce the number of nodes to at least an order of magnitude smaller for the iTree. This process not only groups clusters together but also prepares for a more effective drawing, viewing and interacting with the iTree.

The timing was collected on a PC with an Intel Core i7-960 3.2GHz CPU, 24GB main memory, and an nVidia GeForce GTX 580 GPU with 1.5 GB graphics memory. For SAX word creation, breakpoint identification was performed in the CPU while PAA conversion and SAX word generation were performed in the GPU using CUDA. For SAX hierarchy construction, both bucketing and splitting were conducted in the GPU. Nevertheless, excluding I/O time, SAX hierarchy construction can be completed in less than 10 minutes for all data sets. For iTree construction, all tasks were performed in the CPU. Data I/O and sibling reordering tasks were negligible in terms of timing. The rest of two tasks can

be completed within a second. All these three stages were done during preprocessing. At runtime, the drawing of and interaction with the iTree and the volume are interactive.

Data Classification and Tracking. The iSAX hierarchy essentially classifies a time-varying data into multiple levels of detail based on the similarity of their trends. By exploring the iTree, we select nodes of interest and visualize the classification results in the volume view. Figure 5.3 shows an example with the argon bubble data set. After exploring the child nodes of the root, we identify three nodes that correspond to three distinct regions in the volume. As we can see, these three regions are the outer, middle and inner layers of the bubble, respectively. We used one-dimensional transfer function where the scalar data value was mapped to color. The three regions classified have overlap in their data value ranges. Therefore, our iSAX-based classification actually goes beyond straightforward value-based segmentation. This result shows the nice capability of iTree in terms of classifying spatiotemporal data into meaningful groups for interpretation.

Figure 5.6 shows the level-of-detail exploration of the iTree. In the actual interaction, as we move from the root to the leaf node, the selected node at higher levels of detail is displayed with a larger screen space in a F+C manner. The corresponding volumetric region highlighting shows the coarse-to-fine exploration of the combustion data set, starting from the main flame structure and narrowing down to a feature region at the boundary.

Since our iSAX hierarchy is created from a time-varying data organized into multiple time intervals, a node in the iTree corresponds to spatiotemporal regions that exhibit similar patterns or trends. Therefore, the iTree also enables us to track spatiotemporal regions over time. Figures 5.7 and 5.8 show such an example with the supernova data set. In the SAX view shown in Figure 5.7, we can see two dark value ranges with higher density values which indicates that these two ranges may contain interesting large-scale features. The vertical value ranges with non-zero opacity transfer function content are highlighted in pink in the first time interval. In Figure 5.8 (a), we zoom into the first time interval and brush some regions of interest. The corresponding SAX words that have a distance within

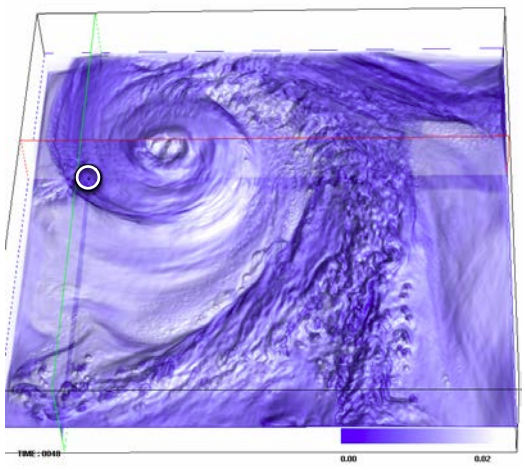
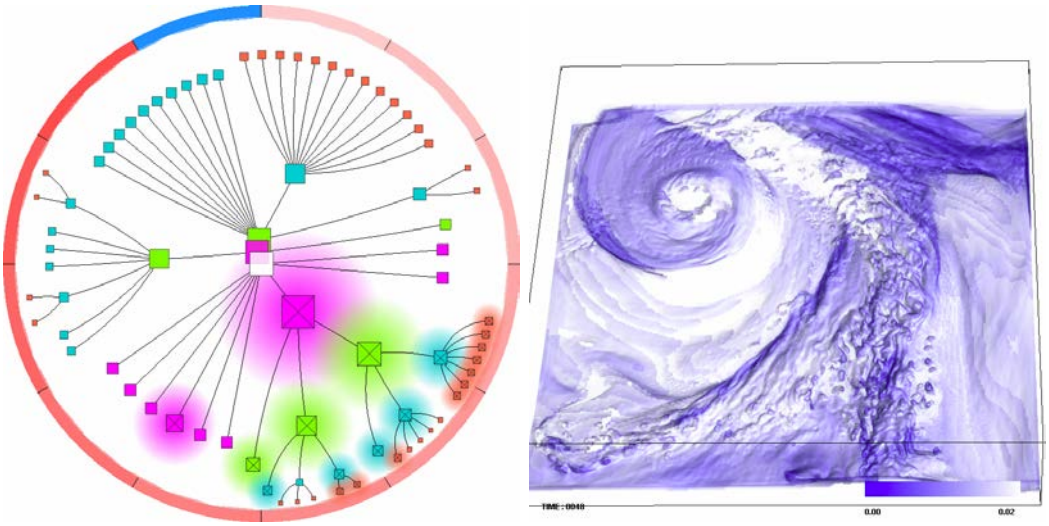


Figure 5.13. Searching the iTree of the hurricane data set. A block at $(x, y, z, t) = (91, 31, 20, 48)$ is selected using three slices along the x, y and z axes, respectively.

1.0 to the selected regions are displayed in Figure 5.8 (b). In Figure 5.9, the corresponding iTree nodes are highlighted with halos. The time ring is also displayed. In the time ring, the first time step starts from the 12 o'clock direction and subsequent time steps follow the clockwise direction. Time intervals are marked in black and the current time interval is highlighted in blue. The corresponding volumetric regions satisfying the query are shown in (2) to (4) at three selected time steps. Compared to (1), we can see that the selected region actually corresponds to the main internal large-scale features of the supernova data set. The highlight results are consistent across different time steps to support meaningful tracking.

Data Indexing and Search. We conducted our data indexing and search by comparing brute-force search, exact search and approximate search. Brute-force search does not use any indexing scheme but simply goes over the PAA representation of data for identifying similar voxels or groups of voxel. We compared the performance for searching the current time interval and searching all time steps. The searching performance results with different

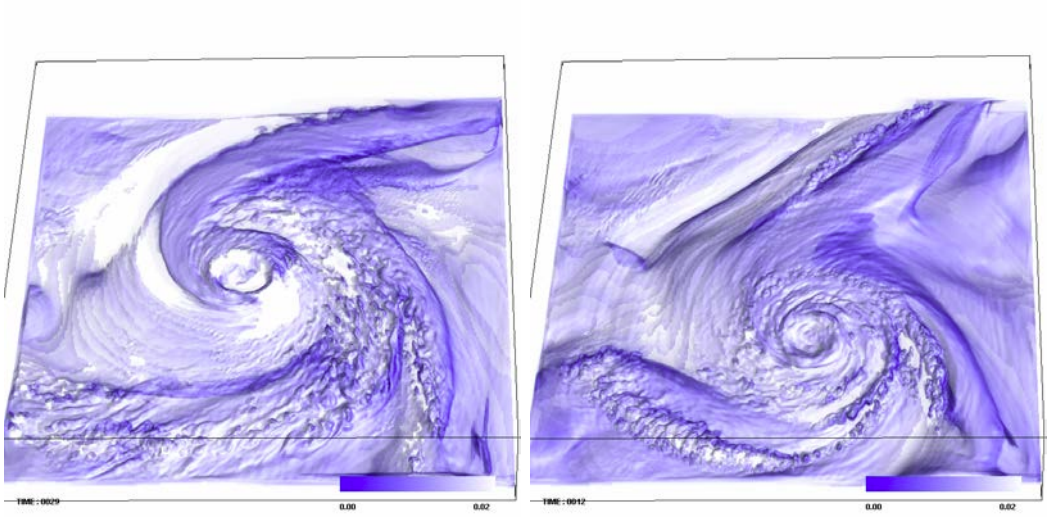


(a) iTree highlighting (b) time step 48

Figure 5.14. Searching the iTree of the hurricane data set. (a) and (b) show the exact search results of Figure 5.13 with $\delta = 0.000711$.

data sets are shown in Table 5.3. The threshold δ selected for each data set is proportional to its value range. We can see that, in most cases, the time for brute-force search is much larger than that of exact search under the same threshold, and the time for exact search is much larger than that of approximate search. Comparing searching the current time interval with searching all time steps, we find that usually the time cost does not increase much for approximate search, but does increase substantially for both brute-force and exact searches. This is because approximate search only involves using the names of index files for distance computation, while the other two searches need to examine the content in these index files.

In Figures 5.10, 5.11, and 5.12, we show the querying of the earthquake data set using the SAX view. As shown in Figure 5.10 (a), a group of blocks are selected at time step 90 and their SAX words are displayed in the SAX view in (b). As shown in Figure 5.11 (a), using automatic F+C visualization based on the relative importance of each time interval (along the horizontal direction) and each value range (along the vertical direction), we are



(a) time setp 29 (b) time step 12

Figure 5.15. Searching the iTree of the hurricane data set. (a) and (b) are the results at other two selective time steps for the exact search of Figure 5.14 with the same threshold.

able to see more clearly the pattern of the corresponding SAX words. Next, we classify the SAX words using a k-mean clustering algorithm. Three clusters are displayed while we select one cluster of interest for further querying. The results of all SAX words that are within a distance of 4.0 to the selected cluster at every SAX symbol are highlighted in Figures 5.11 (b) and 5.12 (a) to (d).

Figures 5.13, 5.14, and 5.15 show an example of the exact search with the hurricane data set. A block at a certain spatial position at time step 48 is selected in Figure 5.13. We perform the exact search with the threshold $\delta = 0.000711$. The result captures the main hurricane structure that is similar to the selected voxel. Our experience shows that compared to approximate search, exact search gives a finer result since after finding all indexed files that match the query criteria, it also requires all voxels in the matched files to be compared with the PAA representation of the query voxel.

CHAPTER 6

IGRAPH FOR SCALABLE VISUAL ANALYTICS OF IMAGE AND TEXT COLLECTIONS

6.1 Overview

In this chapter, we present iGraph, a scalable approach to visual analytics of large image and text collections. Given such a collection, we computed the similarity between images, the distances between texts, and the connections between images and texts to construct iGraph, a compound graph representation which encodes the underlying relationships among these images and texts. To enable effective visual navigation and comprehension of iGraph, we present a progressive solution that offers collection overview, node comparison, and visual recommendation. Our solution not only allows users to explore the entire collection with representative images and keywords, but also supports detail comparison for understanding and intuitive guidance for navigation.

We experiment with two well-known collections: the APOD collection and the MIR Flickr collection. The Astronomy Picture of the Day (APOD) [93] is an online astronomy image collection maintained by NASA and Michigan Technological University. Everyday APOD features a picture of our universe, along with a brief explanation written by a professional astronomer. Since its debut in June 1995, APOD has archived thousands of handpicked pictures, which makes it the largest collection of annotated astronomy images on the Internet. The MIR Flickr collection [62] is offered by the LIACS Medialab at Leiden University. The collection was introduced by the ACM MIR Committee in 2008 as an ACM sponsored image retrieval evaluation. We use the MIRFLICKR-25000 collection

which consists of 25,000 annotated images downloaded from the social photography site Flickr through its public API.

6.2 iGraph Definition and Construction

We define iGraph as a compound graph that consists of two kinds of nodes, i.e., I-node and K-node, and three kinds of edges, i.e., I-I edge, I-K edge, and K-K edge, where "I" stands for image and "K" stands for keyword. An I-node (K-node) represents an image (keyword) extracted from the collection. An I-I edge (K-K edge) is formed between two I-nodes (K-nodes) and the edge weight indicates the similarity between these two I-nodes (K-nodes). Finally, an I-K edge is formed between an I-node and an K-node only if there is a connection between them.

6.2.1 I-node and K-node

In iGraph, each I-node corresponds to an image. Each K-node corresponds to a keyword related to more than one image (otherwise, we treat it as a rare keyword and exclude it from iGraph). Keywords are given as tags in the MIR Flickr collection. For the APOD collection, we extract meta-tagged keywords from the HTML header and other keywords from the paragraph of explanation accompanying each image.

In the explanation sections, we first extract the nouns using a tag extraction tool named "Stanford Log-linear Part-Of-Speech Tagger" [120]. Then we combine the keywords extracted from the HTML header and explanation sections. Finally, we manually check the keywords to eliminate the low-frequent or meaningless keywords. We use frequencies of the keywords as their importance instead of "term frequency-inverse document frequency" (TF-IDF) because of the low reappearance of the keywords in the same webpage. TF-IDF considers the overall frequencies and the number of appearing webpages of each keyword. In the APOD collection, since the explanation sections are short, the keywords barely appear twice. Therefore, the frequencies of the keywords are approximately the same as the

number of webpages which they appear in. We thus use the frequencies as their importance.

6.2.2 I-I edge, K-K edge, and I-K edge

An I-I edge exists between any two I-nodes and the edge weight is the similarity between the two images. Different images in the collection come with different dimensions, types, and formats. For simplicity, we convert all images to the same type and format (i.e., portable pixmap format, PPM), and scale them down to a fixed resolution (256×256) for similarity analysis. Such a resolution strikes a good balance between maintaining image content and achieving comparison efficiency. We consider three aspects of images, namely, *grayscale content*, *power spectrum*, and *color histogram*, to calculate the distance between two images [138]. The overall distance between two images is a weighted sum of the three partial distances.

An K-K edge exists between any two K-nodes and the edge weight is the similarity between the two keywords. To analyze the similarity between two keywords, Gomaa and Fahmy [49] categorized the approaches into string-based, corpus-based, knowledge-based, and hybrid similarity measures. A corpus-based similarity measure is a semantic similarity measure. It determines the similarities between keywords according to information gained from large corpora. This kind of measures usually provides better similarity measurement than string-based measures. A knowledge-based similarity measure is also a semantic similarity measure, but it uses information derived from semantic networks. Among corpus-based similarity measures, we choose Google similarity distance (GSD) [25] due to its simplicity.

Given that a keyword K_a appears in a webpage, its GSD to another keyword K_b is computed as

$$GSD(K_a, K_b) = \frac{\log \max\{f_a, f_b\} - \log c_{ab}}{\log MAX - \log \min\{f_a, f_b\}}, \tag{6.1}$$

where f_a and f_b are the frequencies of K_a and K_b, respectively, c_{ab} is their co-occurrence frequency, and MAX is the multiplication of the largest two frequencies in all keywords. As we can see, if K_a and K_b always appear simultaneously, their distance is zero. When the frequencies of K_a and K_b are constant, their distance gets smaller if they share more webpages. When the number of sharing webpages is constant, their distance gets larger if the number of non-sharing webpages increases.

An I-K edge exists between an I-node and an K-node if and only if the image has the keyword in its tags in the MIR Flickr collection or the image and keyword exist in the same webpage of the APOD collection.

6.3 Progressive Drawing

6.3.1 Initial Backbone iGraph

Direct drawing the entire iGraph consisting of over tens of thousands of nodes incurs a heavy computational cost and usually produces a poor visualization, making the subsequent exploration very difficult. We therefore advocate the visual analytics mantra: "*analyze first, show the important, zoom, filter and analyze further, details on demand*" [73] in the drawing. Specifically, we first extract the backbone of iGraph by identifying representative I-nodes and important K-nodes from the graph for overview. We apply *affinity propagation* [42] to the image collection to identify representative I-nodes. Unlike k-means and k-medoids clustering algorithms, affinity propagation simultaneously considers all data points as potential exemplars and automatically determines the number of clusters. For K-nodes, we rank their corresponding keywords based on the frequency to determine their importance.

For an I-I edge (K-K edge), the edge weight is defined as the similarity of the two incident I-nodes (K-nodes). For any I-K edge, we define the edge weight as 1 (0) if there is a (no) connection between the I-node and the K-node. We modify a classical force-

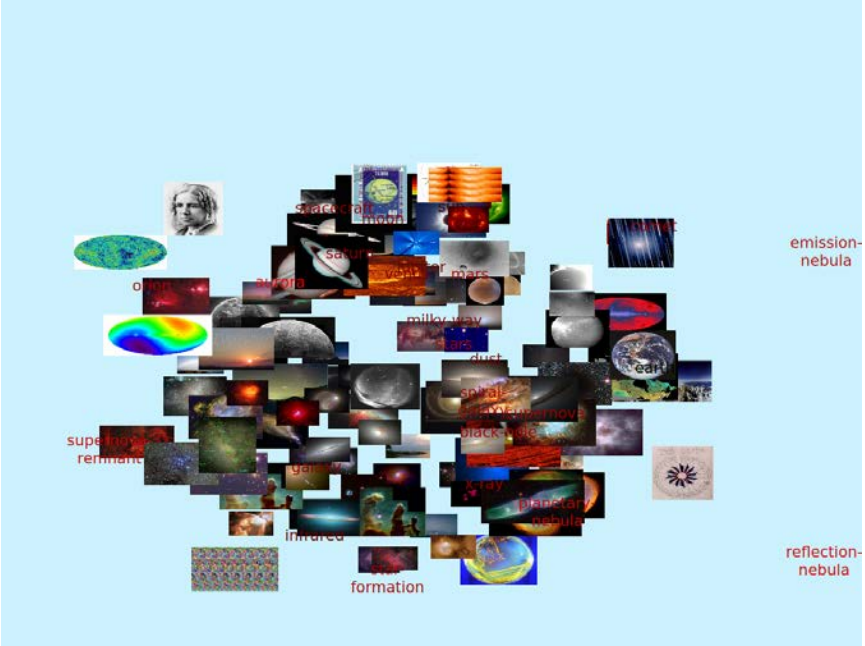


Figure 6.1. The result of the modified FR algorithm.

directed graph layout algorithm, the Fruchterman-Reingold (FR) algorithm [44], to draw the initial iGraph. Specifically, we design two forces, the repulsive force F_r and attractive force F_a. Given two nodes with their graph distance d and similarity σ, the two forces are calculated as $F_r = k_r/d$ and $F_a = k_a \times d^2 \times \sigma^2$, where k_r and k_a are the constant factors for the repulsive and attractive forces, respectively. We add σ^2 in the attractive force to ensure that more similar nodes are closer to each other. This is not included in the repulsive force, because we want to ensure that there is always a distance between any two nodes.

6.3.2 Dynamic Layout Adjustment

As a user explores iGraph, we dynamically adjust it around areas of interest to allow navigation through the graph in a progressive manner. The graph keeps updating as the new images or keywords of focus are selected. Note that each I-node or K-node displayed in the graph occupies a certain rectangular area. We map the size of an I-node to the

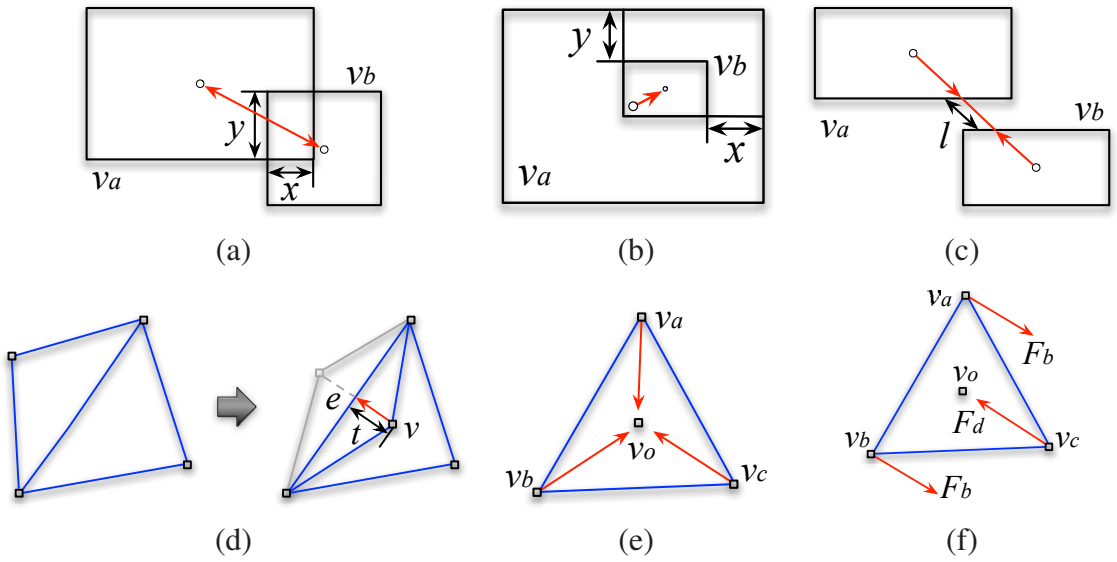


Figure 6.2. Four forces for constrained layout adjustment: (a) bidirectional repulsive force, (b) unidirectional repulsive force, (c) spring force, and (d) attractive force. Two forces for density adjustment: (e) density force and (f) bounding force.

importance of the corresponding image. For K-nodes, we use the same font size, while their frequency is mapped to color saturation (bright red to dark red). The most challenging issue for dynamic layout adjustment is to reduce their overlap or occlusion. In particular, we do not allow an K-node to have any overlap with other nodes, while two I-nodes could overlap each other but we want to reduce such an overlap as much as possible. A good layout adjustment algorithm should also maintain a good balance between preserving the structural information of the graph and revealing the dynamics.

To this end, we generate an initial layout for the backbone iGraph. To achieve stable graph update, good screen utilization, and node overlapping reduction, we introduce four adjustment steps: *constrained layout adjustment*, *density adjustment*, *K-node overlapping adjustment*, and *occluded I-node removal*. For constrained layout adjustment, we apply a triangulation scheme [108] to all the nodes in the initial graph as shown in Figure 6.1 and use the resulting mesh to perform the adjustment. Similar to the algorithm described

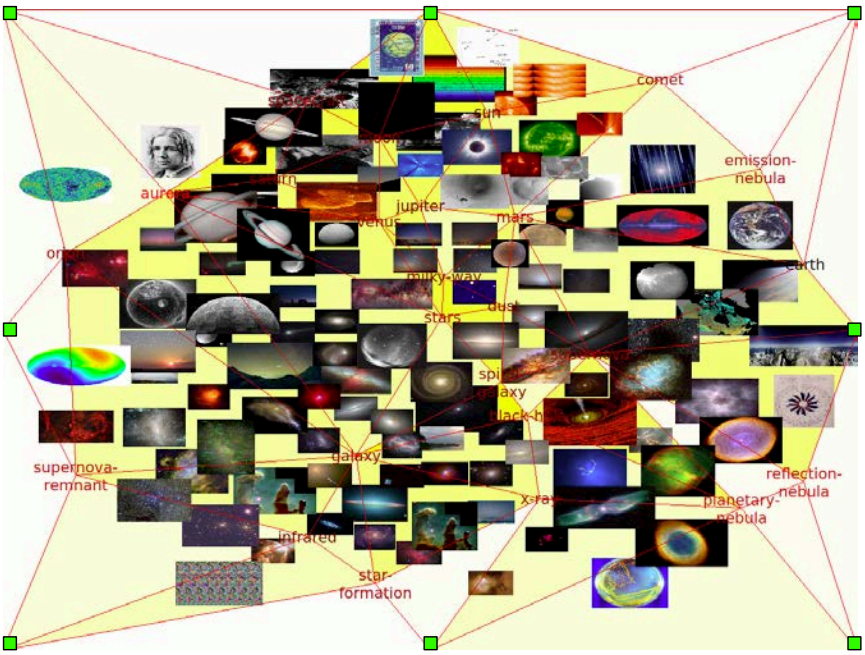


Figure 6.3. The triangle mesh based on K-nodes after constrained layout adjustment of Figure 6.1. The colors of the triangles indicate their density values (higher saturation, higher density). Eight bounding nodes are marked in green.

by Cui et al. [29], we consider four kinds of forces to reposition the nodes to reduce their overlap while maintaining the topology of iGraph. These forces are:

- *Bidirectional repulsive force:* This force pushes away two nodes v_a and v_b from each other and is effective only if v_a and v_b overlap each other. It is defined as $F_1(v_a, v_b) = k_1 \times \min(x, y)$, where k_1 is a given weight and x and y are the width and height of the overlapping (see Figure 6.2 (a)).

- *Unidirectional repulsive force:* This force pushes away a node v_b from a node v_a and is effective only if v_b is inside v_a. It is defined as $F_2(v_a, v_b) = k_2 \times \min(x, y)$, where k_2 is a given weight and x and y are the width and height of the gap region (see Figure 6.2 (b)).

- *Spring force:* This force balances the graph by offsetting the two repulsive forces

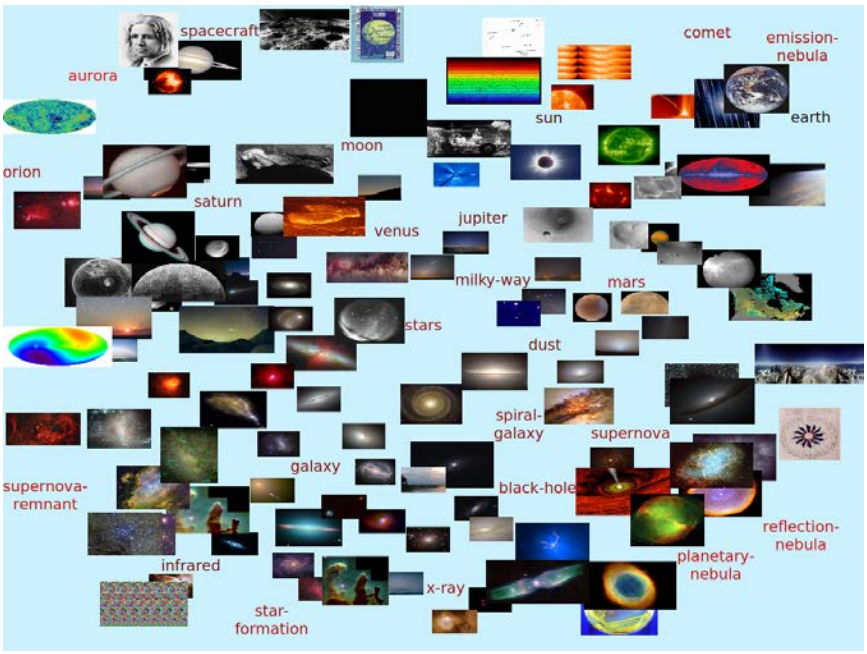


Figure 6.4. The layouts after K-node overlapping adjustment and occluded I-node removal based on Figure 6.3.

introduced. Given two nodes v_a and v_b, the spring force is defined as $F_3(v_a, v_b) = k_3 \times l$, where k_3 is a given weight and l is the length of the line segment connecting the centers of v_a and v_b that lies outside of their boundaries (see Figure 6.2 (c)).

- *Attractive force:* This force maintains the topology of the triangle mesh we construct for the graph. During layout adjustment, a triangle may be flipped (see Figure 6.2 (d)). Our goal is to maintain stable update of the graph by introducing an attractive force to flip the triangle back. The attractive force is define as $F_4(v) = k_4 \times t$, where k_4 is a given weight and t is the distance from node v to edge e. We also consider virtual triangle edges connecting extreme nodes in the graph to the *bounding nodes* (i.e., four corners and the midpoints of four sides of the drawing area, see Figure 6.3). This is to ensure that all graph nodes do not go out of bound.

For density adjustment, we apply the same triangulation scheme to the K-nodes only.

For each triangle in the resulting mesh, we calculate its density value ρ as $\rho = \sum_i A_i/T$, where A_i is the size of the image whose center is in the triangle and T is the size of the triangle. Then we introduce the following two forces:

- *Density force:* The density force F_d keeps the triangle area proportional to its density. As shown in Figure 6.2 (e), for each node, F_d pulls it to the center of the triangle. The force is calculate as $F_d = k_d \times \rho$, where k_d is a given constant.

- *Bounding force:* If we only apply the density force, the eight bounding nodes will be pulled toward to the drawing center. However, since the bounding nodes are fixed, all the rest of points will be pulled to the drawing center. To balance the effect that F_d works on the bounding nodes, we introduce the bounding force F_b. In Figure 6.2 (f), assume v_c is the bounding node and v_a and v_b are not. F_d on v_c is the density force, F_b on v_a and v_b is the bounding force which has the same magnitude as F_d but in the opposite direction.

For K-node overlapping adjustment, we reduce the overlapping with any K-node by adjusting the positions of the nodes which overlap with an K-node. Figures 6.1, 6.3, and 6.4 show an example of our dynamic layout adjustment results. As we can see, constrained layout adjustment nicely reduces node overlap while maintaining the graph topology. Density adjustment is able to pull nodes further apart and further reduces their overlap. Finally, K-node overlapping adjustment dictates that no K-node should overlap any other node. This is to make sure that all keywords are easily readable.

For occluded I-node removal, we calculate for each I-node, the percentage of its pixels overlapped with any other I-node. When the largest percentage of all I-nodes is larger than a given threshold, we simply remove the corresponding I-node from the graph and update the percentages for the remaining I-nodes. We repeat this removal process until the largest percentage is smaller than the threshold.

We did not apply IPSEP-COLA [39] because the FR algorithm usually works faster

than the algorithms using quadratic solvers. IPSEP-COLA even allows users to predefine the minimal distances between nodes. In iGraph, although we do not want nodes overlapping with each other, due to the sizes of images and texts as well as the number of image and text items displayed, overlapping within the given display area often becomes inevitable. Therefore, the predefined minimal distance does not fit our purpose. On the other hand, our force model can reduce the overlaps and better utilize the screen space. In addition, different from the algorithms using quadratic solvers, the placement of each node in iGraph can be calculated individually. Therefore, our algorithm can be further accelerated using the parallel algorithm as described in Section 6.7.

6.3.3 Graph Transition

To preserve the mental map (i.e., the abstract structural information a user forms by looking at the graph layout) during graph exploration, we provide animated transition from one layout to another by linearly interpolating node positions over the duration of animation. Besides the compound graph, we also provide users with the option to observe the image or keyword subgraph only in a less cluttered view through animated transition.

6.4 Filtering, Comparison and Recommendation

6.4.1 Interactive Filtering

We provide interactive filtering to users for sifting through the graph to quickly narrow down to images or keywords of interest. For images, users can scan through the entire list of images to identify the ones they would like to add into iGraph. They can also type a keyword to retrieve related images for selection. For keywords, users can scan through the entire list of keywords to identify the ones or type a keyword prefix to find matched ones for selection. Users can also sift through the graph according to the time information. They can not only retrieve the images and keywords in a particular time period, but also click a

keyword to retrieve its related images in the time period for selection. For keywords that are already in the graph, we highlight them in red while the rest in the list are in black. For images, we draw them with lower opacity in the list if they are already in the graph. In addition, users can also dynamically adjust the number of images or keywords they would like to display in the current view.

6.4.2 Node Comparison

Our node comparison allows users to compare nodes of interest for detail comprehension. Similar to the work of PivotPaths [34], we allow users to select or input two to four nodes for comparison. These nodes will be moved to fixed positions around the drawing center. For each selected I-node (K-node), we retrieve n most similar I-nodes (K-nodes) and m related K-nodes (I-nodes) as its group, where n and m are user-defined parameters. Then, we remove the nodes from the previous layout which are not in any group being compared. After that, we apply the modified FR algorithm while fixing the positions of selected nodes, fade in the nodes that are not in the previous layout, and dynamically adjust the layout while the selected nodes are given the freedom to move. The reason is that the layout will be very cluttered in most cases if we always fix the positions for the selected nodes. Meanwhile, the mental map would still be preserved since the selected nodes will not change their positions dramatically compared to their previously fixed positions during the last step of the adjustment. We assign different colors to the selected nodes to indicate their group memberships.

6.4.3 Visual Recommendation

To allow exploring iGraph with no prior knowledge, it would be desirable for our system to show only the most relevant portions of the graph to users, while suggesting directions for potential exploration. When a node is selected, we apply collaborative filtering to recommend related nodes. These recommendations are highlighted in a focus+context

manner. User interaction history is saved as the input to collaborative filtering.

Used by Amazon.com, the collaborative filtering [85] recommends items to customers based on the item they are currently viewing. We keep an *item-user matrix* **N** where the rows are the items (images and keywords in our scenario) and the columns are the users. This item-user matrix records how many times an item has been selected by a user. Unlike the collaborative filtering which recommends the items based on all the items that the current user has selected, we only recommend based on item I_j that the user just selected. An array A is created, where each entry A_i records the number of users that selected both items I_i and I_j ($i \neq j$). Then we sort A in the decreasing order and recommend the first n items while n is a user-defined number. We refer to this solution as the *user-centric* approach.

Another solution is the *item-centric* approach. Initially, when we either do not have many users or do not have much user exploration history, we mostly make recommendations based on the similarity between items. That is, we recommend nodes that are most similar to the node selected. We provide three similarity metrics for users to select. The first metric is the direct use of the similarity matrix defined in iGraph, which we call the primitive approach. The second and third ones are derived using the rooted PageRank [82] and Katz [72] algorithms, respectively. With an increasing number of users using iGraph leading to richer user exploration history, we are able to gradually shift from the item-centric approach to the user-centric approach and effectively increase the number of items recommended. In the following, we discuss the rooted PageRank and Katz algorithms in detail.

Given a graph, the hitting time $H_{(n_i,n_j)}$ measures the estimated number of steps from one node n_i to another node n_j using random walks. There are two issues with this measure. First, if n_j has a large stationary probability, the hitting time $H_{(n_i,n_j)}$ is very small from any given node n_i. Second, this measure is sensitive to the subgraph far away from n_i and n_j even if they are very close to each other. This is because the random walks may walk to

(a) (b)

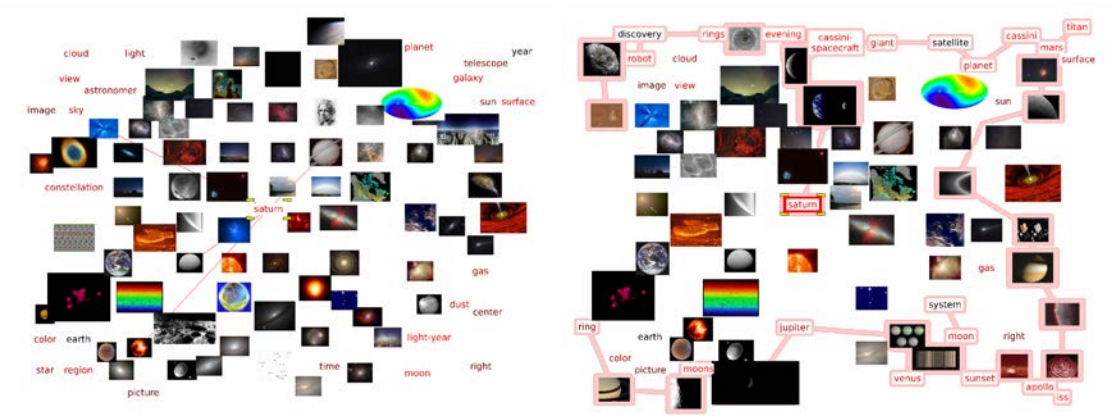
Figure 6.5. Node comparison of (a) the APOD data set and (b) the MIR Flickr data set. (a) Three K-nodes "emission nebula", "planetary nebula", and "reflection nebula" are chosen for comparison. The nodes with green bottom-left corners, red upper-left corners, and blue upper-right corners are related to "emission nebula", "planetary nebula" and "reflection nebula", respectively. (b) One K-node and two I-nodes are chosen for comparison. The K-node is "self-portrait" and the two I-nodes are images of people. One is a picture of a girl wearing a T-shirt and the other is a picture of a pair of feet.

a subgraph far away from them and cannot return to n_i and n_j within a few steps, which increases the hitting time. The rooted PageRank [82] was designed to solve these two issues. To solve the first issue, it considers the stationary probability along with the hitting time in the measurement. To solve the second one, it allows the random walks from n_i to n_j to periodically return to n_i with a probability a.

Assuming a graph is represented using an adjacency matrix $\mathbf{A}$, we compute the degree matrix $\mathbf{D}$ as

$$\mathbf{D}_{i,j} = \begin{cases} \frac{1}{\Sigma_{k=1}^{N} \mathbf{A}_{i,k}}, & i = j \\ 0, & i \neq j \end{cases} \tag{6.2}$$

where N is the number of nodes in the graph. The probability matrix $\mathbf{P}$ measures the



(a) (b)

Figure 6.6. Visual recommendation of the APOD data set. (a) and (b) iGraphs before and after the recommendation (based on K-node "saturn"), respectively. The recommended nodes are highlighted with the bubble set in (b).

probability of walking from any node to its neighbors in a single step with a uniform probability, where

$$\mathbf{P} = \mathbf{DA}. \tag{6.3}$$

Then we compute the similarity matrix $\mathbf{S}$ as

$$\mathbf{S} = (1-a)(\mathbf{I}-a\mathbf{P})^{-1}, \tag{6.4}$$

where a is the probability of returning to the starting node and $\mathbf{I}$ is the identity matrix.

Katz [72] argued that the similarity between two nodes is not only related to how similar they are, but also related to how many others are similar to them. The similarity $S_{i,j}$ between two nodes n_i and n_j is computed as

$$S_{i,j} = \sum_{l=1}^{\infty} a^l \times |\mathbb{P}(n_i, n_j)^l|, \tag{6.5}$$

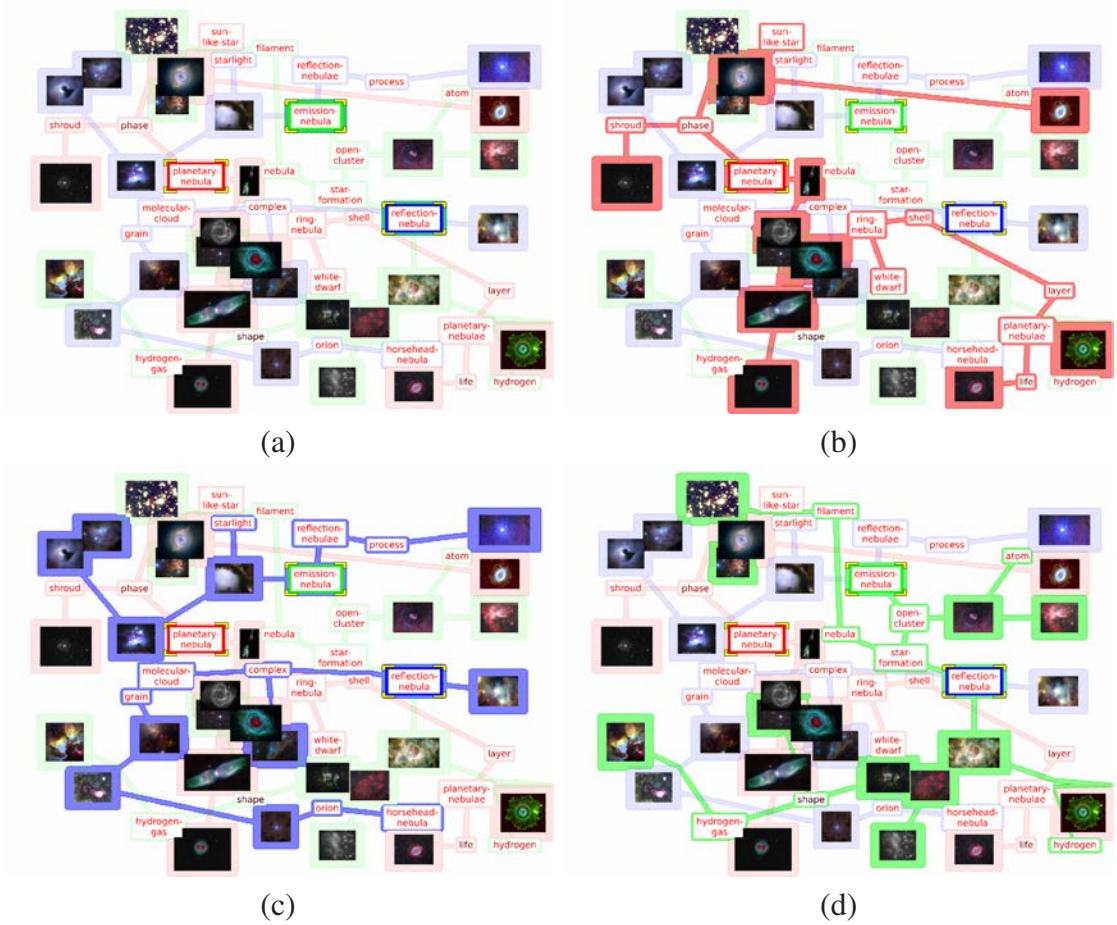


Figure 6.7. Node comparison of Figure 6.5 (a) using the bubble set visualization to highlight the belonging relationships.

where $\mathbb{P}(n_i, n_j)^l$ is the set of paths from n_i to n_j with path length l, and a is a weight between 0 and 1. Assuming a graph is represented as an adjacent matrix $\mathbf{A}$, the similarity matrix $\mathbf{S}$ is computed as

$$\begin{aligned} \mathbf{S} &= a\mathbf{A} + a^2\mathbf{A}^2 + \ldots + a^k\mathbf{A}^k + \ldots \\ &= (\mathbf{I} - a\mathbf{A})^{-1} - \mathbf{I}. \end{aligned} \tag{6.6}$$

In terms of visualization, rather than visualizing recommended items in a separate view

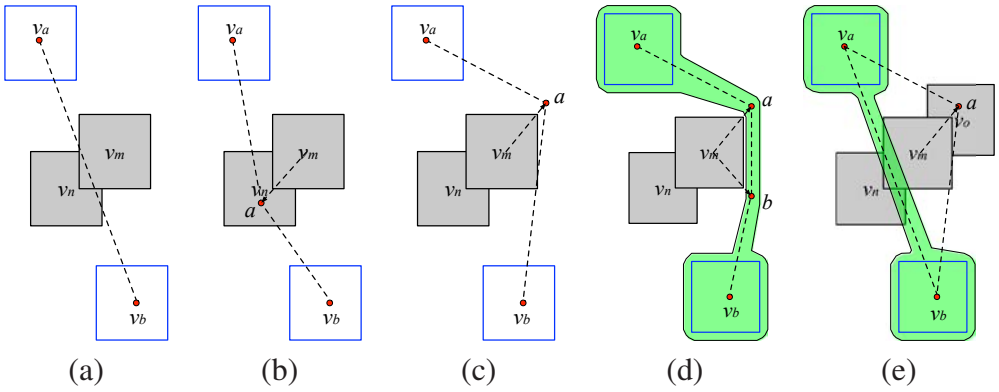


Figure 6.8. (a) A backbone edge $v_a v_b$ in S_b overlaps with nodes v_m and v_n in S_r. (b) The route tries to find an intermediate point a to avoid crossing v_m, but a is still in v_n. (c) The route tries the opposite corner of v_m and finds a so that $v_a a$ does not overlap with v_m. However, av_b still overlaps with v_m. (d) By adding a new point b, the final route avoids overlapping with v_m and v_n. The green region indicate the corresponding bubble set. (e) In another example, a is still in $v_o \in S_r$. Therefore, this algorithm could not always avoid the overlapping. In this case, we simply use the original backbone edge $v_a v_b$ as the route.

[28], we add recommended nodes to the current node being explored and rearrange the iGraph layout. The end result is that more nodes will show up in the surrounding in a focus+context fashion, and in the meanwhile, we selectively remove some nodes that are less relevant from the layout so that the total number of nodes displayed is kept as a constant. The criteria to remove less relevant nodes could be nodes which have longest paths from the current node being explored, or nodes which are least recently explored, etc.

We also utilize the image popularity and keyword frequency gathered from our data analysis stage for initial suggestions. The display sizes of the popular images are proportional to the logarithmic scale of their popularities. We also set the minimum and maximum display sizes and aspect ratios to avoid very large or small images. The frequent keywords will be highlighted with more saturated colors. The larger display sizes and more saturated colors direct the user's attention for more purposeful exploration.

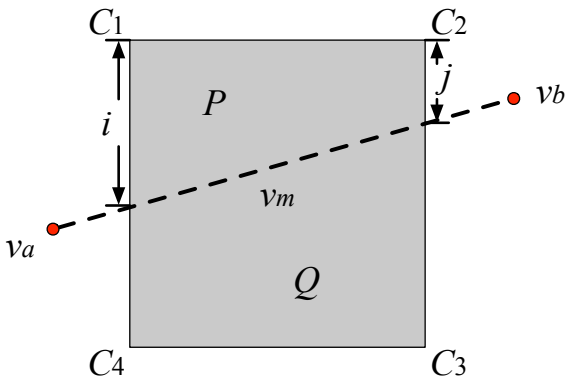


Figure 6.9. A backbone edge $v_a v_b$ in S_b overlaps with node v_m. The routing points can be identified using Algorithm 10.

6.5 Bubble Set Visualization

For node comparison and visual recommendation, we assigned different colors to the boundaries of the nodes to indicate their group memberships [53]. However, the force-directed layout algorithm attracts similar nodes together while pushing dissimilar nodes away. As a result, the nodes belonging to the same group may not be necessarily close to each other. Therefore, users have to go through each individual node to identify its membership. This is very inefficient with a large number of nodes in the graph. It is even more difficult if some nodes have multiple group memberships (Figure 6.5). To solve these problems, we utilize the bubble set visualization [27] to highlight the group membership of nodes. Bubble sets use implicit surfaces to create continuous and dynamic hulls. These hulls can be treated as visual containers that highlight group relationships without requiring layout adjustments to existing visualizations. Using bubble sets, we can produce tight capture of group members with less ambiguity compared with using convex hulls (e.g., Figures 6.6 and 6.7).

Kelp diagram [33] and KelpFusion [91] can also highlight group memberships. These algorithms first allocate the space of each node using the Voronoi diagram. Then they both

Algorithm 10 POINT v = FINDROUTEPOINT(v_a, v_b, v_m, S_r, *distbuff*) (Refer to Figure 6.9.)

```
The line connecting node v_a and v_b overlaps with node v_m
swap ← false
v ← null
while distbuff > 0 and v overlaps with nodes in S_r do
    if v_a v_b intersects with a corner c of v_m then
        v ← c
    else
        if Area(P) ≤ Area(Q) then
            if i > j then
                v ← (swap ? C_1 : C_3) + distbuff
            else
                v ← (swap ? C_2 : C_4) + distbuff
            end if
        else
            if i > j then
                v ← (swap ? C_3 : C_1) + distbuff
            else
                v ← (swap ? C_4 : C_2) + distbuff
            end if
        end if
    end if
    if swap then
        reduce distbuff
    end if
    swap = ¬swap
end while
return v
```

utilize the shortest paths to determine the linkage between nodes in the same group. Kelp diagram simply draws the edges while KelpFusion draws an area instead. We choose to use bubble set because nodes in iGraph have large space occupation while nodes in Kelp diagram and KelpFusion are rather small. In addition, bubble set visualization allows node overlapping, while node overlapping and space allocation are difficult to handle in Kelp diagram and KelpFusion.

In a graph G, assume that the nodes to be highlighted in a bubble set are in the set S_b, and the remaining nodes are in the set S_r. The bubble set should consist of all the nodes in S_b and try to avoid containing or overlapping the nodes in S_r. There are four steps to extract the bubble set. First, we extract a backbone of S_b to connect the nodes. This backbone forms an approximate shape of the final bubble set. Second, for each edge of the backbone that overlaps with the nodes in S_r, we use a routing algorithm to avoid

the overlapping. Third, we calculate an energy field to illustrate the display space of the bubble set. A positive value is given to a location near the backbone while a negative value is added if the location is close to the nodes in S_r. Finally, we utilize the marching squares algorithm to identify the contour of the energy field. The contour should contain all the nodes in S_b and thus becomes the boundary of the bubble set.

6.5.1 Backbone

The backbone that connects all the nodes in S_b should not only consider the length of the straight line connecting the centers of two nodes but also try to avoid crossing the nodes in S_r. We define the cost of connecting two nodes v_a and v_b in S_b as

$$C(v_a, v_b) = \delta(v_a, v_b) \times (N(v_a, v_b) + 1), \tag{6.7}$$

where $\delta(v_a, v_b)$ is the length of line segment $v_a v_b$, and $N(v_a, v_b)$ is the number of nodes in S_r that are crossed by $v_a v_b$. We use $(N(v_a, v_b) + 1)$ to make sure that the cost always considers the length of $v_a v_b$. By minimizing the total cost of connecting the nodes in S_b, we generate the backbone. To minimize the cost, Collins et al. [27] started with the node near the center of S_b and then iteratively added the node with the minimal cost with respect to the selected nodes. The disadvantages of their solution are that it is difficult to define where the center of S_b is and it does not always yield the optimal solution. In contrast, we leverage the minimum spanning tree (MST) to extract the backbone. This eliminates the needs to find the center of S_b and optimizes the solution.

6.5.2 Routing

Although the backbone extraction tries to choose the edges that overlap with few nodes in S_r, some edges may still cross the nodes in S_r. Routing is then applied to solve this problem. In Figure 6.8 (a), the backbone edge $v_a v_b$ intersects with nodes v_m and v_n in

S_r. We would add new points to avoid the crossing. In Figure 6.8 (b), a new point a is added. This point has a distance *distbuff* to the corner of v_m because we would like to reserve some space for the bubble set. However, a is in node v_n, the new edges $v_a a$ and $a v_b$ still overlap with node v_n in S_r. Therefore, we place a near the opposite corner of v_m as shown in Figure 6.8 (c). The new edge $v_a a$ does not overlap with nodes v_m and v_n, but $a v_b$ does. This prompts us to add a new point b as shown in Figure 6.8 (d). The green region illustrates the resulting bubble set. However, if the nodes in S_r are very cluttered, a may still overlap with another node, e.g., v_o as shown in Figure 6.8 (e). In this case, we will not add new points. The detail routing algorithm is described in Algorithm 10.

6.5.3 Energy Field and Contour

The energy field illustrates the coverage of the backbone. If a location is close to the backbone, it has a high energy value, otherwise it has a low energy value. If this location is close to the nodes in S_r, the energy value would be reduced or could even be negative. The bubble set is a contour with energy $E(x,y) = c$, where c is a given positive isovalue. This value is selected so that the bubble set will contain all the nodes in S_b and tries to avoid the nodes in S_r. To ensure the performance, we first uniformly partition the display screen into grid points and calculate the energy value for each grid point instead of each pixel. For each grid point P_i, an energy is calculated with respect to an item I_j (i.e., a node or edge) as

$$E(P_i, I_j) = w_j \frac{(D_1 - \delta(P_i, I_j))^2}{(D_1 - D_0)^2}, \tag{6.8}$$

where D_0 is the distance where the energy is 0, and D_1 is the distance where the energy is the maximum, $\delta(P_i, I_j)$ is the distance between grid point P_i and item I_j, and w_j is the weight assigned to item I_j. The nodes and edges in the backbone have positive weights while the nodes in S_r have negative ones.

Initially, we only calculate the energy of the grid points with respect to the nodes and edges in the backbone. The higher the absolute values of the item weights, the larger the item coverage areas are. Since edges are difficult to observe if they are too thin, in our implementation, we assign a larger value (2) to their weights. The weights of nodes in the backbone are assigned a smaller value 1. If the energy of a grid point is positive, the grid point is close to the backbone. However, the grid points may be close to the nodes in S_r. To avoid the overlapping, we go through the nodes in S_r and add their negative energies to the grid points. The weights of the nodes are assigned the value of -1. Finally, a positive isovalue c is specified to extract the corresponding contour. A larger value of c may lead to separated components but the contour will get tighter. We will adjust the value of c accordingly so that the contour will form a single connected component.

6.6 Text Outlier Detection

For an image collection that evolves over time, keywords with high frequencies normally are of interest. However, user interests may shift. Sometimes, an unpopular keyword may appear frequently in a limited time period while a popular keyword may seldom appear or even disappear. Such a phenomenon indicates that an anomalous event happens during that time period. Detecting text outliers can therefore help us gain a deep understanding of the collection by separating them out for further exploration. We exclude K-nodes with low frequencies from this analysis since they already belong to outliers. To detect text outliers with high frequencies, we utilize the algorithm introduced by Akoglu and Faloutsos [2]. Their algorithm allows us to detect anomalous time periods and nodes. In order to detect the anomaly, we first evaluate the importance of each node in each time period. The importance of a node can be treated as its current behavior in time period t. Then, a time window w is selected which covers a larger number of time steps than time period t. The importance of each node in w before t can be treated as its recent behavior. By comparing the current and recent behaviors of a node, we can tell how abnormal this

node behaves. In addition, by summarizing the abnormality of all the nodes, we know how abnormal this time period is. If the nodes behave similarly to their recent behaviors, then the current time period is normal. Otherwise, it is abnormal.

6.6.1 Node Importance

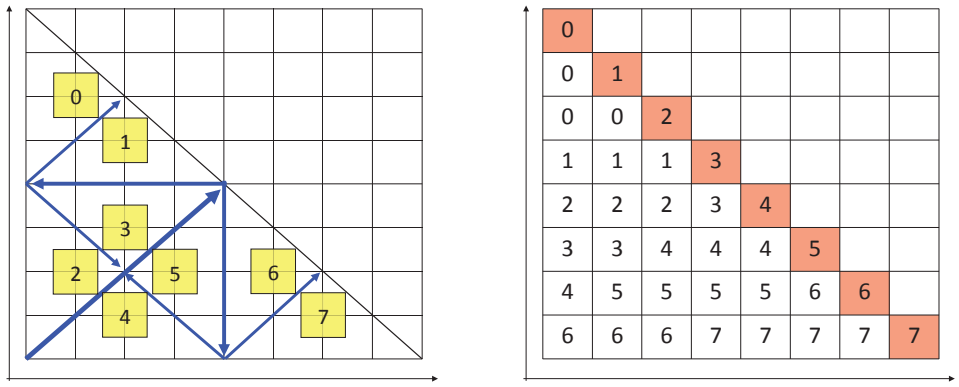
Given a time period t, we construct a co-occurrence matrix $\mathbf{C}_t$ to store co-occurrence frequencies of all the K-nodes. According to the Perron-Frobenius theorem [43, 97], the largest eigenvalue of $\mathbf{C}_t$ is positive and the value for each node in the corresponding eigenvector $\mathbf{e}_t$ indicates the importance of that node.

6.6.2 Time Period and Keyword Outlier Detection

By measuring the changes of node importance, we can identify anomalous time periods and nodes. We acquire the largest eigenvector of each time period in the time window w before t. Akoglu and Faloutsos [2] suggested that the average of these largest eigenvectors (denoted as $\mathbf{e}_w$) is good enough to evaluate node behaviors. By taking the dot product of the two unit vectors $\mathbf{e}_w$ and $\mathbf{e}_t$, we get the difference between time period t and its previous time periods in w. This difference value Z is calculated as follows

$$Z = 1 - \mathbf{e}_w{}^T \mathbf{e}_t. \tag{6.9}$$

If these two unit vectors are perpendicular to each other, Z is 1 and if they are exactly the same, then Z is 0. The larger the value of Z, the more anomalous time period t is. Given a node, we subtract its corresponding values in $\mathbf{e}_w$ and $\mathbf{e}_t$ to detect how abnormally this node behaves at time period t. After identifying anomalous time periods, users can extract related I-nodes and K-nodes for further exploration. If they are interested in an anomalous K-node at a certain time period, they can also use interactive filtering to extract related I-nodes.

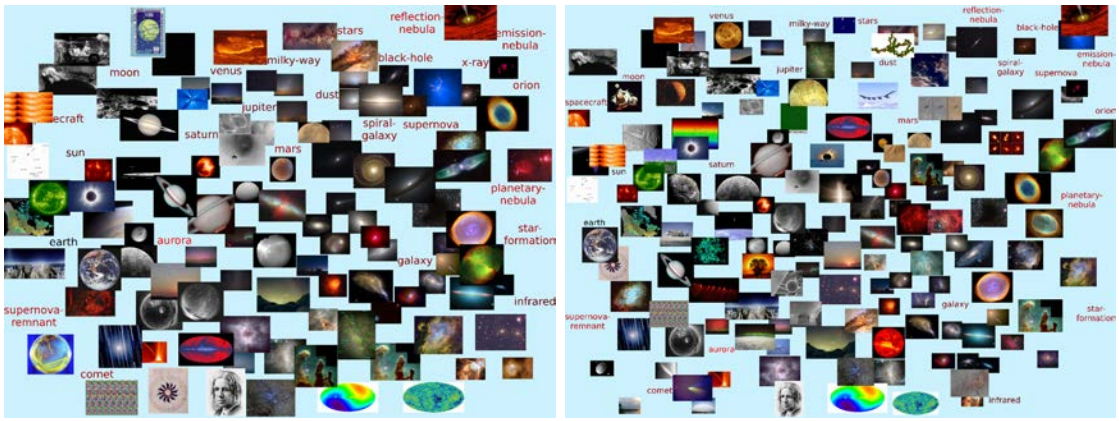


(a) (b)

Figure 6.10. Two different approaches to partition an half matrix, assuming eight GPUs (0 to 7) are used. (a) The first approach partitions the matrix into halves recursively. (b) The second approach works with any number of partitions.

6.7 Parallel Acceleration and Display Wall

To improve the performance of our approach, we leverage a nine-node GPU cluster along with a display wall employed in the Immersive Visualization Studio (IVS) at Michigan Technological University. The GPU cluster consists of one front-end node and eight computing nodes. The display wall consists of 6×4 thin-bezel 46-inch Samsung monitors, each with 1920×1080 pixels. In total, the display wall can display nearly 50 million pixels simultaneously. These 24 monitors are driven by eight computing nodes for computation and visualization. Each node comes with a quad-core CPU, two NVIDIA GeForce GTX 680 graphics cards, and 32GB of main memory. The aggregated disk storage space is over tens of terabytes for the GPU cluster. We use this cluster not only for visualization, but also for performance improvement of preprocessing and layout generation.



(a) (b)

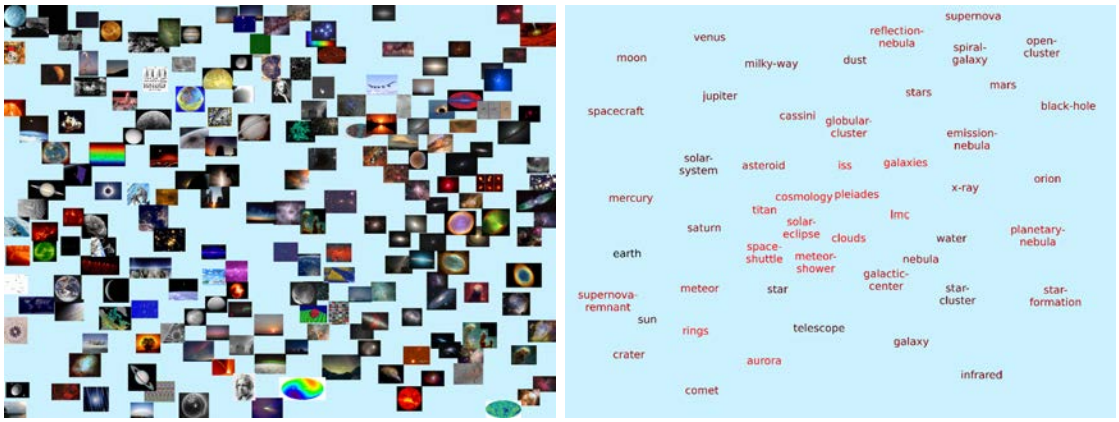
Figure 6.11. iGraphs with different numbers of I-nodes and K-nodes. For (a) and (b), the numbers of I-nodes and K-nodes are 110 and 25, 190 and 25, respectively.

6.7.1 GPU Parallel Preprocessing

For the computation of grayscale and spectrum image distance matrices, it could still take hours to complete when running on a single GPU. To further improve the performance, we use Open MPI and distribute the workload to several GPUs and perform the computation simultaneously.

There are two ways to evenly partition the symmetric distance matrix. As shown in Figure 6.10 (a), the first approach iteratively partitions the matrix into halves as indicated by blue lines, and the partition results are indicated with yellow rectangles. This approach achieves a perfectly balanced workload while the number of GPUs must be a power of two. Furthermore, since the triangle of each partition is stored as an array in CUDA and the number of images in a collection is an arbitrary number, it is challenging to combine the resulting triangles to the original half matrix in an efficient way.

Figure 6.10 (b) illustrates the second approach. Assuming that we need to partition the half matrix into p partitions, we first divide the matrix into p rows and p columns. Then



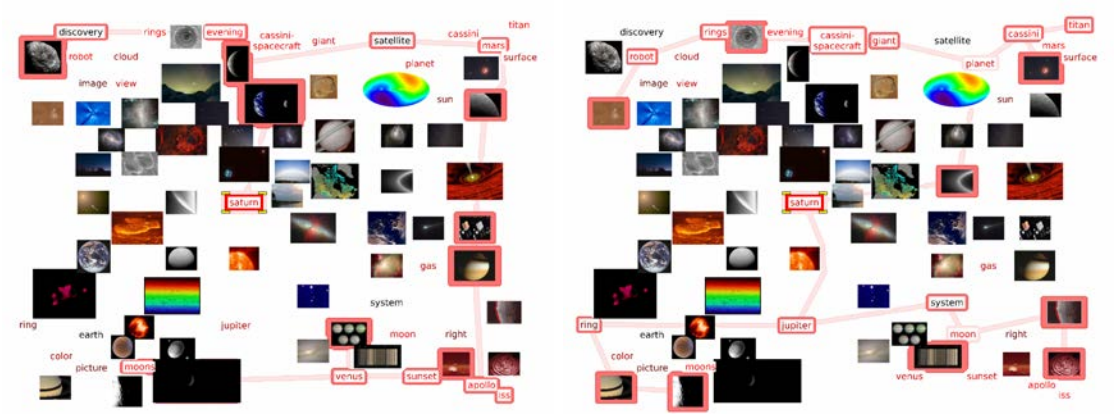
(a) (b)

Figure 6.12. iGraphs with different numbers of I-nodes and K-nodes. (a) and (b) show the single graphs with 270 I-nodes and 50 K-nodes, respectively.

for the p rectangles along the diagonal, each partition gets one. The number of remaining rectangles is $p \times (p-1)/2$, which means that each partition gets $(p-1)/2$ rectangles. If p is an odd number, then each partition gets the same number of rectangles. Otherwise, half of the partitions get one more rectangle than the rest. Although the number of images for each partition to load is not perfectly balanced, this approach still achieves a very balanced workload for a large p and it works well with any number of GPUs. In practice, p could be a large number (e.g., a multiple of the number of GPUs) and we can distribute the partitions to each GPU in a round robin manner. Meanwhile, because we store the computation results for each partitioned rectangle, it is easy to index and retrieve the similarity value for any pair of images. Therefore, we implement the second approach.

6.7.2 CPU Parallel Graph Layout

We use MPI for the parallel iGraph layout generation using multiple CPUs. The most time-consuming computations are initial backbone iGraph, constrained layout adjustment, and density adjustment. Thus we parallelize these three steps to achieve good perfor-



(a) (b)

Figure 6.13. The nodes recommended by the user-centric and primitive item-centric approaches in Figure 6.6 are highlighted in (a) and (b). Those nodes highlighted with dark red boundaries in (a) and (b) do not show up in the previous iGraph.

mances. The input is the information of all I-nodes and K-nodes (i.e., their initial locations, widths, and heights) while the output is the iGraph layout after applying these steps.

For initial backbone iGraph, each node needs to calculate the repulsive force with any other node. After we distribute the graph nodes evenly to the processors, they need to exchange position information in each iteration of the FR algorithm. The initial layout changes dramatically in early iterations and gets more and more stable in later iterations. To reduce the communication cost, we allow the processors to have more frequent communications in early iterations than later ones. For constrained layout adjustment, there are four forces. The bidirectional and unidirectional repulsive forces may occur between any two nodes. Therefore, after we distribute the graph nodes to different processors, they also need to exchange position information in each iteration. Our experience shows that the attractive and spring forces take much more time to compute than the two repulsive forces. As such, when the processors have similar numbers of attractive and spring forces, we consider that the workload balancing is achieved. Since the numbers of attractive and

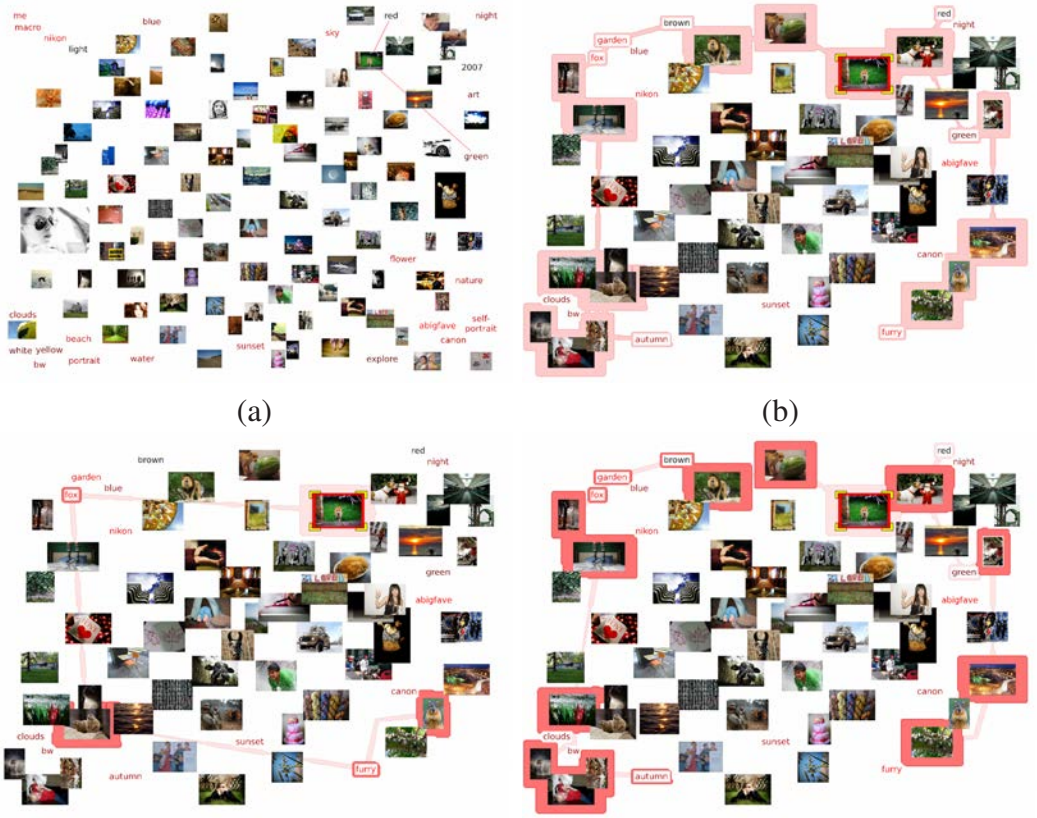


Figure 6.14. Visual recommendation of the MIR-flicker data set. (a) and (b) iGraphs before and after the recommendation based on an I-node showing the picture of a fox, respectively. The recommended nodes are highlighted with the bubble set in (b). The nodes recommended by the user-centric and primitive item-centric approaches are highlighted in (c) and (d). Those nodes highlighted with dark red boundaries in (c) and (d) do not show up in the previous iGraph.

TABLE 6.1

THE SIZES OF APOD AND MIR FLICKR DATA SETS

data set	# images	# keywords	# I-I edges	# K-K edges	# I-K edges
APOD	4,560	5,831	10M	17M	137K
MIR Flickr	25,000	19,558	312M	191M	173K

repulsive forces are related to the degrees of nodes, we sort node degrees in the decreasing order and allocate those graph nodes to different processors in a round robin manner. A processor would not receive any more nodes if the accumulated node degrees reaches the average total node degrees it should have. In this way, all the processors shall have a similar amount of accumulated node degrees. Similar to constrained layout adjustment, in density adjustment, we distribute K-nodes based on their degrees. In each iteration, after calculating the new positions of the K-nodes, the processors exchange position information, update the densities of the triangles, and repeat the process until certain criteria are met.

TABLE 6.2

THE PARAMETER AND TIMING RESULTS OF SIMILARITY

COMPUTATION AND LAYOUT GENERATION

data		similarity	block	thread	loading	computing	speedup	saving	# graph	layout	speedup
set	# images	type	config.	config.	GPU	GPU	factor	GPU	nodes	CPU	factor
APOD	4,560	grayscale	16,384	512	0.214s	762.312s	-	0.008s	125	0.38s	-
		spectrum	16,384	1,024	0.213s	515.042s	-	0.007s	225	1.29s	-
		histogram	16,384	1,024	0.002s	3.259s	-	0.008s	525	6.96s	-
MIR Flickr	25,000	grayscale	512	512	12.715s	25,722s	-	0.312s	550	10.24s	-
			768	768	3.179s*	3,127s*	8.22	0.088s*		0.90s*	11.37
		spectrum	512	512	12.623s	13,380s	-	0.272s	1050	47.60s	-
			768	768	3.163s*	2,151s*	6.22	0.089s*		2.07s*	22.99
		histogram	1,024	1,024	0.0296s	0.117s	-	0.274s	2050	5.61s*	-

6.7.3 Display Wall Visualization

To render iGraph on the display wall, we design four program components: the *master*, *computation*, *relay* and *slave* programs. The master program not only has a user interface to accept all the user interactions but also displays iGraph. It runs on a local computer which is located in the same room as the display wall. This computer captures user interactions, sends instructions to the computation program for generating the graph layout in parallel. After receiving the layout result from the computation program, the master program sends data to the cluster's front-end node. The data contain all the information for rendering, e.g., all the I-nodes and K-node positions, widths and heights. Meanwhile, this front-end node running the relay program receives the data and broadcasts it to the eight computing nodes of the cluster. The slave program is an OpenGL application running on these eight nodes. Each node receives the data sent from the relay program, decodes the data, and renders the visualization results to the three tiles which it is responsible for.

For the slave program, there are two problems that we need to address. The first is that the time gap between neighboring receiving actions is less than that of decoding so that the data may change undesirably during decoding. The second is the synchronization of receiving, decoding and displaying because the data may change if two of these three actions happen simultaneously. To address the first problem, we create a large buffer, use one thread to receive the data and save it to the buffer so that the new incoming data will not override the previous data. This also prevents receiving from interrupting decoding and rendering. Furthermore, to solve the synchronization of decoding and rendering, we create another thread to iteratively decode the data and invoke OpenGL to display. In our experiment, we did not encounter inconsistency among the eight computing nodes. Therefore, we do not synchronize the tasks among them.

6.8 Results

6.8.1 Data Sets and Performance

In our experiments, we used two data sets, APOD and MIR Flickr. Their data sizes are shown in Table 6.1. We can see that the connections between I-nodes and K-nodes are less in MIR Flickr than in APOD. On average there are only 6.9 K-nodes connected to each I-node in MIR Flickr while 31.1 in APOD. This results in two distinguished groups of I-nodes and K-nodes in the initial iGraph for the MIR Flickr data set. The configurations and timing results for computing image distance matrices are shown in Table 6.2. The single GPU computation time was reported using a desktop PC (not a cluster node) with an nVIDIA GeForce GTX 580 graphics card, while multiple GPUs computation time was reported using the GPU cluster (Section 6.7). At run time, all the tasks and interactions are interactive.

6.8.2 Initial Backbone iGraph and Single Graph

The initial backbone iGraph chooses representative I-nodes and K-nodes to show the overall structure. The representative I-nodes are preselected using affinity propagation, and the representative K-nodes are those with the highest frequencies. Meanwhile, users can choose the number of nodes as shown in Figures 6.11 and 6.12, where the number of I-nodes increases from 110 to 270 and the number of K-nodes increases from 25 to 50. When the number of nodes increases, to maintain the screen occupancy, the sizes of nodes decrease. However, we set the minimal sizes for I-nodes and K-nodes respectively in order to make the images and keywords readable. Figure 6.12 (a) and (b) are the single graphs with only I-nodes and K-nodes, respectively. Single graphs are displayed when users want to observe only I-I edge or K-K edge relationships.

6.8.3 Visual Recommendation

The initial backbone iGraph only gives an overview while our recommendation system serves as a navigation tool to help users in their exploration. The example with the APOD data set is shown in Figures 6.6 and 6.13. From Figure 6.13 (a) we know that users who are interested in "saturn" are also interested in "mars", "moons", and "satellite". In Figure 6.13 (b), for the keywords in the bubble set, "cassini" and "cassini spacecraft" correspond to the Cassini orbiter which reached Saturn and its moons in 2004. "ring" means the outer ring of Saturn.

Another example with the MIR Flickr data set is shown in Figure 6.14. In (b), we show iGraph after recommendation based on an image of a fox. From (c) we know that the users who are interested in the fox are also interested in "furry" animals such as squirrel and otter. In (d), "red", "green", "garden", and "fox" are related to the image of the fox.

The user-centric approach and the primitive approach have their own limitations. When we either do not have many users or do not have much use exploration history, the user-centric approach sometimes could not recommend any node since the selected node was not explored by any user previously. If an I-node (K-node) does not have many related K-nodes (I-nodes), the primitive approach also could not recommend enough nodes. To overcome the limitations, we implement the rooted PageRank and Katz algorithms. These two algorithms connect all the I-nodes and K-nodes together as a graph. The similarity between two nodes depends on their closeness in the graph. Therefore, two nodes that share a large number of related nodes will be similar. This is not the case for the primitive approach. In addition, since all the nodes are connected together, we can always recommend enough nodes for any selected node.

TABLE 6.3

COMPARING FOUR RECOMMENDATION APPROACHES OF APOD AND MIR FLICKR DATA SETS

	APOD					MIR Flicker				
# of recommended nodes	12	25	50	100	200	12	25	50	100	200
user-centric	7	7	9	5	0	1	5	4	0	0
primitive item-centric	6,457	6,012	4,670	1,528	49	33,572	29,152	22,728	17,524	8,015
rooted PageRank	**6,942**	**6,515**	**6,192**	4,769	1,227	**36,580**	**38,130**	**39,797**	**37,083**	**29,422**
Katz algorithm	638	1,266	2,855	**6,989**	**9,672**	1,105	3,070	6,050	8,325	14,843

To compare the performances of these four approaches, we go through each node in iGraph and make recommendation based on that node. For any given node v_a used for recommendation, if a node v_b recommended by an approach is also recommended by any other approach, we consider v_b as a valid recommendation. The approach that has the largest number of valid recommendations is the best for v_a. Table 6.3 lists the frequencies of the best approaches for the APOD and MIR Flicker data sets. From the table, we can observe the following four facts. First, the user-centric approach performs the worst simply because we do not have large user histories. Second, among the ten test cases, the rooted PageRank performs the best except for two cases with the APOD data set, for which the Katz algorithm performs the best. Third, the Katz algorithm performs better as the number of recommended nodes increases. This is due to the fact that this approach has very strict requirements. To measure the similarity of two nodes, it considers not only their closeness, but also the number of nodes which are close to both nodes. Since the requirement is so strong, the most similar nodes based on the Katz algorithm are not recommended by the other approaches. On the other hand, since the Katz algorithm considers so much information, the similarity measurement gets more accurate as the number of recommended nodes increases. Fourth, the primitive approach and rooted PageRank algorithm perform similarly. However, the performance of the primitive approach drops dramatically with the increasing number of nodes recommended. That is because as the number of recommended nodes increases, the primitive approach could not recommend enough nodes. Therefore, the number of valid recommendations decreases. Since none of the approaches work the best for all the cases, we suggest to use the most valid recommendations (i.e., nodes that recommended by the most number of approaches) as the recommendation results.

6.8.4 Node Comparison

While visual recommendation helps users explore similar nodes or nodes clicked by others, node comparison allows users to choose multiple nodes for detailed comparison.

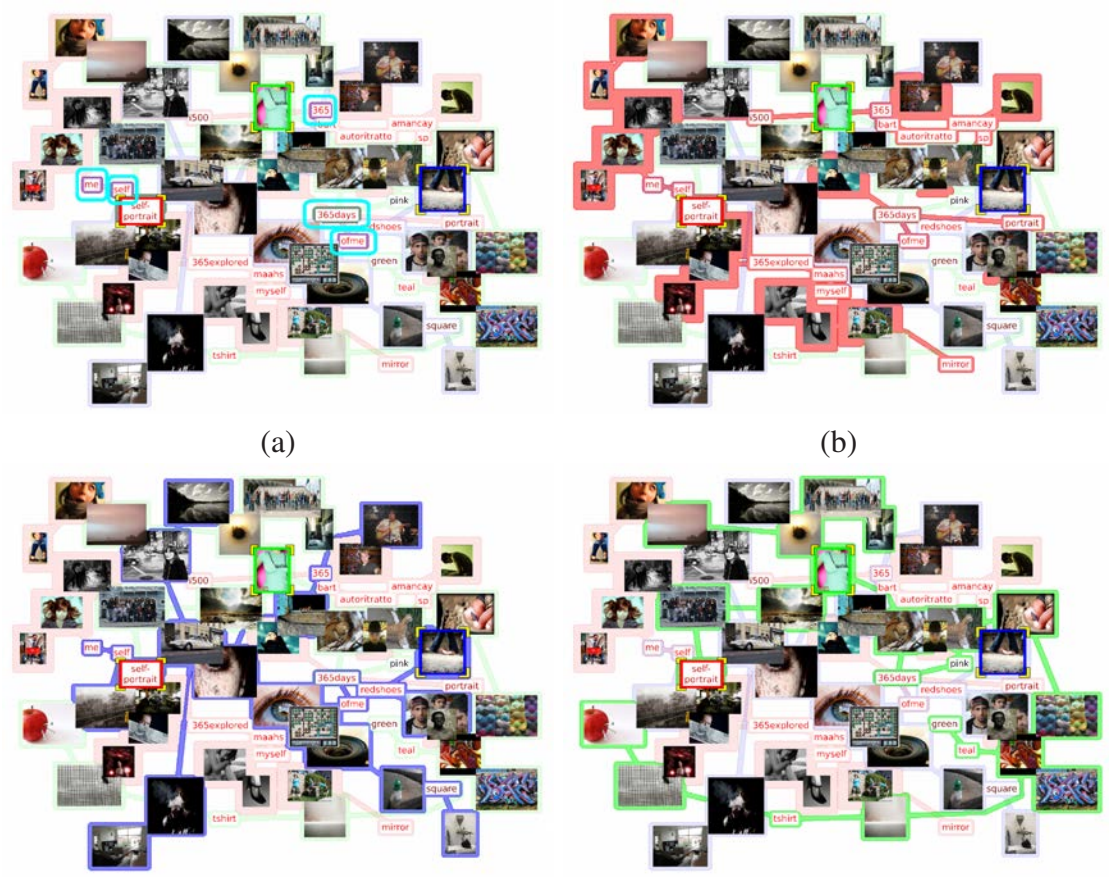
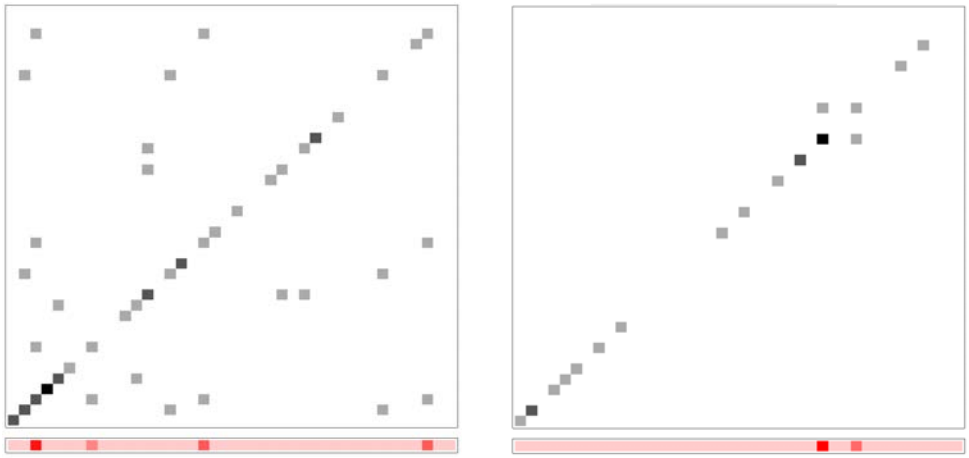


Figure 6.15. Node comparison of Figure 6.5 (b) using the bubble set visualization to highlight the belonging relationships.

For example, a user interested in the three types of nebulas would choose keywords “emission nebula”, “reflection nebula”, and “planetary nebula”. The comparison result is shown in Figure 6.5 (a). The images provide the user with a brief impression of what these nebulas look like. The images of nebulas with green bottom-left corners are “emission nebula”. Most of them are reddish because they are clouds in high temperature and emit radiation. The images of nebulas with blue upper-right corners are “reflection nebula”. They are bluish and darker because they are efficient to scatter the blue light. The images of nebula that look like planes and have red upper-left corners are “planetary nebula”. Meanwhile,



(a) (b)

Figure 6.16. Co-occurrence matrices and their corresponding eigenvectors of the APOD data set in (a) March 2001 and (b) April 2001. The 40 most frequent keywords from the entire data set are used to construct each matrix. Gray rectangles represent non-zero values with darker colors indicating higher frequencies. The corresponding eigenvector is below the co-occurrence matrix where darker red indicates higher values.

the user could also get knowledge from the other keywords. For example, "white dwarf" and "layer" are linked with "planetary nebula" because when an aged red supergiant collapses, the outer layer is the "planetary nebula" while the inside core is the "white dwarf". However, the nodes in the same group may not always be close to each other. As a result, users need to go through all the nodes to figure out their belonging relationships. Figure 6.7 (a) highlights the belonging relationships with the bubble sets. When users prefer to study the nodes related to a selected node, they will be further highlighted with dark boundaries as shown in Figure 6.7 (b), (c), and (d).

In another example, we choose one keyword "self-portrait" and two images of people. The comparison result is shown in Figure 6.5 (b). The keyword "365 days" is marked by three different colors, which means that it is related to the K-node and two I-nodes chosen. "me", "ofme", and "self" are highlighted with both red and blue boundaries which

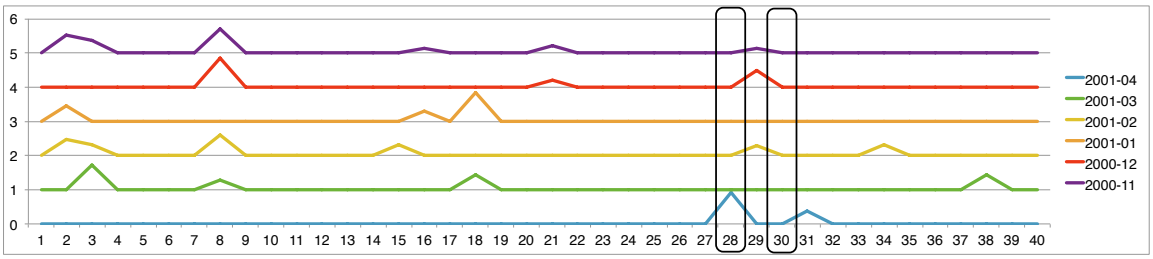


Figure 6.17. The eigenvectors of the 40 most frequent keywords from November 2000 to April 2001. The significant difference is the dramatic jump of importance value for keywords “space shuttle” (index 28) and “iss” (index 31). The interest shift to the international space station and space shuttle is due to the launching of the space shuttle on April 23, 2001.

means that they are related to the keyword “self-portrait” and the image showing a pair of feet. Keywords “red shoes” and “square” with blue boundaries indicate that there is a pair of shoes in the picture and the photo was taken at a square. Figure 6.15 (a) shows the same result using the bubble set visualization. The K-nodes “365 days”, “me”, “ofme”, and “self” are highlighted with blended colors to indicate that they are related to multiple selected nodes. Figure 6.15 (b), (c), and (d) are the results of highlighting the related nodes of a selected node.

6.8.5 Text Outlier Detection

Figures 6.16 and 6.17 demonstrate an anomalous time period and its anomalous K-nodes. Figure 6.16 (a) and (b) show the co-occurrence matrices and their corresponding eigenvectors of the APOD data set in March and April of 2001, respectively. The 40 most frequent keywords in the meta-tagged keywords are used to construct each matrix. These keywords are selected because they appear at least 50 times in the entire data set. The gray rectangles in Figure 6.16 represent the non-zero values in the co-occurrence matrix. Darker colors indicate higher frequencies. The corresponding eigenvector is shown below the co-occurrence matrix, where darker red indicates higher values. Many gray rectangles spread

Figure 6.18. Photos showing iGraph of the MIR Flickr data set running on the 24-tile display wall (1000 images and 50 keywords are displayed).

in Figure 6.16 (a), which indicates no distinct interest in March 2001. The four darker red rectangles in the eigenvector indicate four important K-nodes: “saturn”, “jupiter”, “spacecraft”, and “rings”. The co-occurrence matrix and its corresponding eigenvector in Figure 6.16 (b) allow us identify the most important keywords in April 2001: “space shuttle” and “iss” (international space station). Figure 6.17 lists the eigenvectors of the APOD data set within the five months before April 2001. The outlier detection function indicates that the APOD collection behaved differently in April 2001 compared with its previous five months. The abnormal K-nodes indicate the interest shifted to “space shuttle” and “international space station” since the two keywords were not important in the previous months but became very important in April 2001. This interest shifting may be caused by the launching of the space shuttle on April 23, 2001. Closer examination shows that the APOD creators not only posted figures related to the space shuttle, but also posted some related topics, such as the first man entering the space, the first space quidditch match between the United States and Russia, and STS-1 (the first shuttle launch).

6.8.6 Running on Display Wall

Figure 6.18 shows three iGraph photos of the MIR Flickr data set rendered on the display wall. With the display wall, we are able to display thousands of images and key-

words simultaneously for comfortable viewing. Currently, we are using this display wall for showing iGraph demos to visitors, including university alumni, visitors, and summer students. Initial feedback from several groups of visitors is fairly positive as they commented that running iGraph on this life-size tiled display is much more expressive and fun to watch compared with running on a regular desktop display. The advantage of using the display wall is that it allows more than a dozen of people to comfortably view and discuss the results together in such a collaborative environment. Nevertheless, with the dramatic expanding of display area, it takes more effort for a viewer to correlate and compare images that are on the opposite sides of the display wall, especially for those images close to the wall's boundary.

6.9 User Evaluation

We recruited five unpaid researchers in astronomy for a user study of iGraph on a PC: one professor, three PhD students, and one master student. Four of them are researchers in physics studying astronomy and the remaining one is a computer science PhD student with a master in physics focusing on astronomy. The user study was conducted in a lab using the same PC. The PC has a 27-inch monitor with 1920×1080 resolution, where the visualization result occupied an area of 1600×900. Users were first introduced to iGraph about its goal and main functions. Then they were given the APOD data set for free exploration to get familiar with the system.

The iGraph survey includes seven tasks and five general questions. The seven tasks (**T1** to **T7**) were designed to evaluate the effectiveness of iGraph. **T1** and **T2** asked the user to compare the compound graph with the single graphs and select the numbers of images and keywords that she was satisfied with for the initial backbone graph. **T3** asked the user to go over all the images and keywords to get an impression of the APOD data set. **T4** and **T5**

TABLE 6.4

THE DESCRIPTION OF THE SEVEN TASKS IN THE USER STUDY

task	description
T1	Compare the compound graph with the single graphs
T2	Select the numbers of images and keywords that the user is satisfied for the initial backbone graph
T3	Go over all the images and keywords to get an impression of the APOD data set
T4	Evaluate the visual recommendation function with a given example
T5	Evaluate the visual recommendation function based on user preferences
T6	Evaluate the node comparison function with a given example
T7	Evaluate the node comparison function based on user preferences

TABLE 6.5

SEVEN TASKS IN THE USER STUDY AND USERS' RESPONSE TO SUMMARIZED QUESTIONS

task	summarized question(s)	average rating
T1	Was iGraph helpful in showing the relationship among images and keywords?	5.0
T2	Were the images more useful than the keywords?	4.0
T3	Were the functions that allow the user to go through all images (keywords) helpful?	4.0 (5.0)
T4	Was the visual recommendation function helpful with respect to the given example?	5.0
T5	Was the visual recommendation function helpful with respect to user preferences?	5.0
T6	Was the node comparison function helpful with respect to the given example?	5.0
T7	Was the node comparison function helpful with respect to user preferences?	5.0

asked the user to evaluate the recommendation function (user- and primitive item-centric approaches) with a given example and based on her preference, respectively. **T6** and **T7** asked the user to evaluate the comparison function with a given example and based on her preference, respectively. Every task has several survey questions and write-in questions except **T2** which consists of only write-in questions. All the survey questions have the same set of choices on a 5-point Likert scale: 1: strongly disagree, 2: disagree, 3: neutral, 4: agree, and 5: strongly agree.

We summarized the questions of the seven tasks as shown in Tables 6.4 and 6.5. For simplicity, if the user rated the question as 4 or 5, we consider that she agreed; if she rated it as 1 or 2, she disagreed; otherwise, she was neutral. We used the number of users that agreed to measure the effectiveness of iGraph functions.

Users could perform the tasks whenever they felt comfortable. Each study took about one hour to complete. Although each task could be performed within a few minutes, users frequently returned to the interface for further verification when writing their comments, which took most of the time. Users were informed that the tasks were not timed and their comments were of crucial importance. Among these tasks, two of them are related to the free explorations of the data set with recommendation and comparison functions. Each of these two tasks took them a little more time till they were ready to write comments. The outcome of the user study is summarized in Table 6.5.

Next we provide further explanation on these outcomes. For **T1** and **T2**, we asked users to shift freely between the single graphs and compound graph. In addition, they were asked to change the numbers of images and keywords displayed on the screen in order to get an impression of the best ratio between images and keywords. In terms of **T1**, the feedback was positive. All users stated that iGraph was helpful to show the relationships among images and keywords in both compound and single graphs. Furthermore, a compound graph was considered better than a single graph. The animation of changing between compound and single graphs was helpful to gain an impression of node changes. In terms

of **T2**, one PhD student preferred more keywords than images because the keywords were easier to understand and intuitive to query. The rest of users indicated that since images were more attractive, the ratio between the numbers of images and keywords should be within the range of 3 : 1 to 10 : 1.

For **T3**, we asked users to go through all the images and keywords to gain an impression of the APOD collection. All the images were shown in the order of their published dates and all the keywords were sorted based on their frequencies. Four users considered the functions that allow the user to go through all images were helpful to gain an overall impression of the data set and to select images of interest. The remaining user was neutral on this since she was not familiar with all images and viewing individual images did not make much sense to her. For better browsing of the collection, we can group images based on their similarities and show them one group at a time. Moreover, all users indicated that going through all keywords was helpful. Three agreed that iGraph could be helpful to identify popular and rare keywords while two were neutral on it. This was probably due to the equal weights of the keywords extracted from the explanation and keyword sections. Since the number of keywords extracted from the explanation section is larger than that from the keyword section, equal weights lead to the reduced importance of the keywords from the keyword section. Finally, all users agreed that color and opacity encoding was helpful to identify I-nodes and K-nodes already in iGraph so that they could select new nodes into iGraph for recommendation and comparison.

For **T4**, users were asked to type keyword "earth" for recommendation and see the images and keywords recommended using different approaches. For **T5**, users could select any I-node or K-node from iGraph for recommendation and repeat this process as long as they wanted to. All users agreed that the recommendation function was useful, especially for the primitive item-centric approach because it recommended more relevant nodes than the user-centric approach. However, we believe that the user-centric approach will recommend more relevant nodes when the number of users increases to a certain level. The

functions that allow the user to add any image and keyword made the recommendation function more effective. They also believed that the highlighted boundaries were helpful to identify the recommended nodes.

For **T6**, users were asked to select an image related to "jupiter" and the keyword "moon" for comparison. For **T7**, users could select up to four nodes to compare, and then select any image or keyword in the collection for further exploration. All users agreed that the comparison function helped them find the nodes related to one or multiple selected nodes. Three of them could further dig out the underlying relationships of the selected nodes while two could not. This was because we linked both images and keywords to the selected nodes to show their relationships, which was not a direct approach. All of them agreed that the highlighted boundaries could help identify the belonging relationships of the nodes. Two of them thought that the animation could not demonstrate node changes. This was because when users selected multiple nodes for comparison, most of the nodes after comparison did not appear before. Thus, the animation only gave them the impression of layout change.

For the general questions, two users mentioned that iGraph could not only support researchers to find the underlying relationships of the APOD data set but also help teachers teach in class. Besides that, they stated that iGraph could be extended to handle other image or image and text collections (such as movie collections) and to assist in image search. Users also pointed out some possible improvements for iGraph. First, the techniques related to object identification could be helpful for measuring image similarity. Second, they suggested us to provide more sorting options while going through the APOD collections, e.g., sorting according to the image colors, alphabetical order of keywords, or user preference. Third, they also mentioned that we could provide more flexible query functions. Therefore, we provided text outlier detection to identify the outliers in a certain time period to help users narrow down to certain keywords of interest during the query. Finally, one suggested that we should show the detailed information of a selected image.

CHAPTER 7

ETGRAPH FOR VISUAL ANALYTICS OF EYE-TRACKING DATA

7.1 Overview

In this chapter, we transform the eye-tracking data gathered from a reading study into a graph view named ETGraph (i.e., eye-tracking graph) for visual analytics. To analyze and visualize the eye-tracking data at four different levels: single participant single page (SPSP), single participant multiple pages (SPMP), multiple participants single page (MPSP), and multiple participants multiple pages (MPMP). For single participant, we first cluster the fixations into clusters and build a transition graph to support interactive examination of the underlying structure in the eye-tracking data. For multiple participants, we treat the fixations of all the participants as input. Multiple coordinated views are utilized to dynamically link the graph view with the standard page view during the interaction. ETGraph helps users analyze the reading patterns of a single participant of a single page or among multiple pages, the common patterns of all the participants, and possible patterns around mind wanderings (MWs).

The eye-tracking data set was generated from eye gaze data collected while participants read an excerpt from a book entitled "Soap-bubble and the Forces which Mould Them" [13]. This text was chosen as it was on a novel topic which would be relatively unfamiliar to a majority of readers. The eye gaze data was collected with a Tobii TX300 remote eye tracker affixed below a monitor set to a resolution of 1920×1080, which displayed the text. The excerpt consisted of text from the first 35 pages of the book and contained around 5700 words across 10 pages. Each participant read the text for 20 minutes. The

eye gaze data were collected from both eyes and the data from each eye were filtered and averaged together prior to eye movement detection. The data were then converted into a series of fixations using a dispersion based filter. Using the Open Gaze And Mouse Analyzer (OGAMA) [126], the filter was set to detect fixations if there were consecutive gaze points within a range of 57 pixels (approximately 1 degree of visual angle) for longer than 100 ms, which is the shortest duration for naturalistic eye movements during reading [58, 100]. Saccades were then calculated from the fixations.

7.2 Research Questions

Blascheck et al. [11] defined the basic terminology related to eye-tracking data. We briefly introduce several of them that are used in this work. First of all, *gaze points* are aggregated based on a specified area and timespan, and a *fixation* is an aggregation of gaze points. Furthermore, *saccades* describe a rapid eye movement from one fixation to another, and a *scanpath* is a sequence of alternating fixations and saccades. Since our eye-tracking data are gathered from multiple participants reading multiple pages of a book, we propose the following research questions categorized into four different levels of detail:

- SPSP (*Single Participant Single Page*):

  Q1. What is the scanpath structure of each participant when reading a single page?

  Q2. Does the participant exert a different amount of effort reading different parts of the page?

  Q3. Does the scanpath involve forward and/or backward saccade outliers? (i.e., saccades with amplitudes larger than a given threshold)? If yes, when and where do these saccade outliers occur and how frequent are they? Does the same saccade outlier occur multiple times?

  Q4. Does the scanpath involve repeated scanpaths (i.e., a scanpath that represents rereading previously read text along the same path)? If yes, when and where do

these repeated scanpaths occur and how frequent are they? Does the appearance of saccade outlier have any correlation with the appearance of repeated scanpaths?

Q5. Does the participant MW on a page? If yes, when and where does MW occur?

- SPMP (*Single Participant Multiple Pages*):

  Q6. Is the reading pattern of a participant consistent across all pages?

  Q7. What are the temporal dynamics of reading behavior across consecutive pages? For example, do the saccade outliers on the current page have a relationship with the saccade outliers on the next page?

- MPSP (*Multiple Participants Single Page*):

  Q8. What are the common patterns of multiple participants when reading the same page?

  Q9. Do they spend a different amount of time reading different parts of the page?

  Q10. Do they encounter different reading difficulties on different parts of the page?

  Q11. What are the outliers (who, when and where)? Can we cluster participants based on different reading patterns exhibited on the same page?

- MPMP (*Multiple Participants Multiple Pages*):

  Q12. How can the reading patterns of different participants across all pages be summarized and compared? Do they have similar or different reading strategies?

  Q13. Could we cluster participants based on commonalities in their reading behaviors across all pages? Who are the outliers?

Our goal is to design a visual analytics framework that can help users understand reading patterns of the participants, identify the anomalous behaviors, and group participants. Specifically, how can we apply the above four levels of analysis to obtain new insights

on reading patterns? Can we visually discriminate reading patterns from the graph representations? Can we find out the relationships between saccade outliers and repeated scanpaths? Furthermore, can we visually figure out the similarities and/or differences between the reading patterns on different pages or between different participants? Finally, can we embed the visualization of MWs to help scientists analyze them?

7.3 Our Approach and Employed Techniques

7.3.1 Overall Considerations

Our strategy to analyze the given eye-tracking data is to first cluster fixations into fixation clusters to produce a coarse-level representation of the data. There are two benefits to doing this. First, our fixation clustering provides spatial closeness. Fixation clusters are spatially close fixations which are similar to gazes as defined in Blascheck et al. [11]. However, unlike gazes, we do not consider the temporal ordering of fixations when we perform clustering. Second, fixation clusters are not as coarse as AOIs which represent regions of specific interest on the stimulus. Fixation clusters can be created for a single page using fixations from a single participant or by combining all the fixations from multiple participants. Combining all fixations would allow us to visualize the reading patterns of different participants with a common ground of the same fixation clusters.

We propose a graph-based representation for analyzing eye movements using fixation clusters as nodes and a set of saccades as a directed edge between nodes. We call this visual representation the *ETGraph*, i.e., the eye-tracking graph. Visualizing such a graph can be achieved using force-directed graph layout algorithms or projection-based methods such as multidimensional scaling. In the original page view, fixations and saccades can be plotted to produce scanpaths. However, nodes are solely constrained by their spatial locations on the page. With this stringent constraint, large saccadic amplitudes may not always be of interest (e.g., a saccade moves from the end of one line to the beginning of

the next line). Unlike the page view, nodes in the graph view are not constrained by their corresponding spatial locations of fixation clusters and the graph structure will be dictated by the underlying connectivity of nodes (i.e., the actual reading). Therefore, the graph view can reveal the underlying nature of the reading pattern.

The research questions outlined in Section 7.2 can be classified into six categories: **C1** scanpath structures (**Q1**, **Q6**, **Q8**, and **Q10**); **C2** saccade outliers (**Q3**); **C3** repeated scanpaths (**Q4**); **C4** MW(**Q5**); **C5** participant clustering (**Q11** and **Q13**); and **C6** reading efforts (**Q2**, **Q7**, **Q9**, and **Q12**). To better answer these questions, we introduce four views as shown in Figure 7.1: page view, graph view, time view, and statistics view. Categories 1 to 4 can be answered using the graph, page, and time views. Categories 5 and 6 can be answered using the statistics view. In addition, these four views could be combined together to help users better explore and understand the data.

Finally, we design ETGraph so that users can smoothly switch between SPSP and SPMP, and between MPSP and MPMP. This will facilitate the examination of reading patterns across both global and local perspectives while making connections between them. We do not aim to create a transition between single participant levels (SPSP and SPMP) and multiple participant levels (MPSP and MPMP) for two reasons. First, the transition between single participant levels and multiple participant levels is less effective because the fixation clusters generated at each level do not match due to different inputs for clustering. Second, the transition is unnecessary because participants may exhibit very different reading patterns.

7.3.2 Mean-Shift Algorithm

We use the mean-shift algorithm for fixation clustering. The mean-shift algorithm is a density-based clustering algorithm with three benefits. First, it is deterministic. Second, it does not require users to input the number of clusters. Third, it is robust to outliers. Santella and DeCarlo [105] also demonstrated better quality of clusters produced by the mean-shift

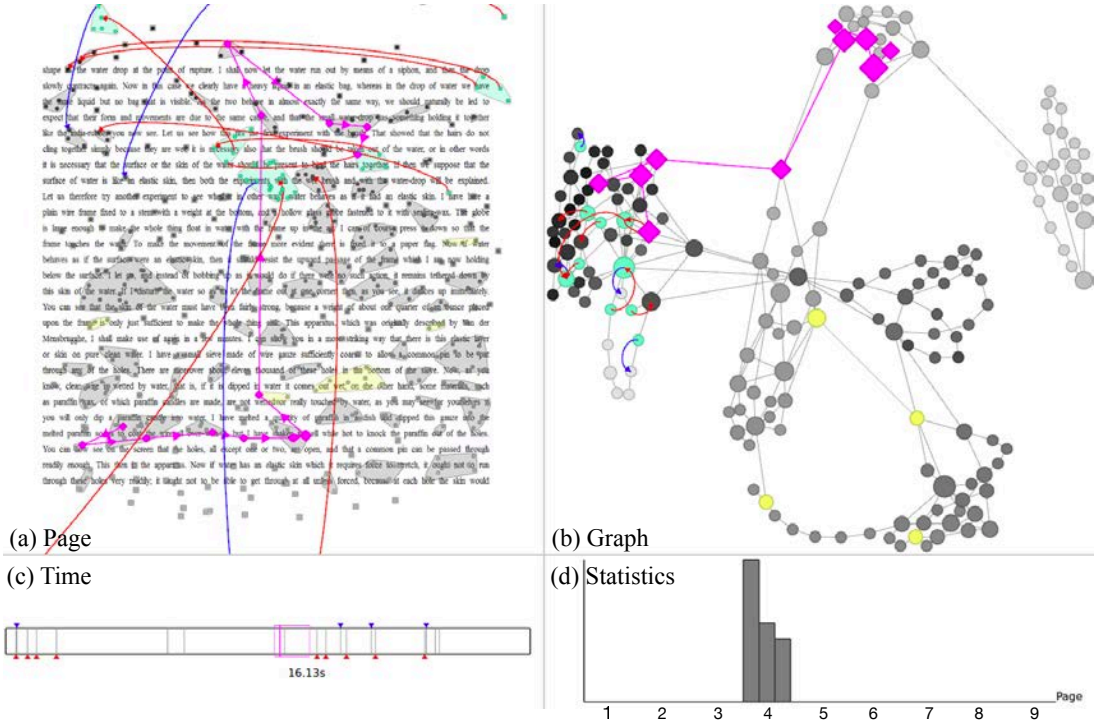


Figure 7.1. The four views of ETGraph. (a) page view, (b) graph view, (c) time view, and (d) statistics views show saccade outliers and MWs of SPSP. In (a) and (b), fixation clusters and nodes corresponding to saccade outliers are highlighted in yellow. The selected clusters and nodes are in green. Red and blue edges indicate backward and forward saccade outliers, respectively. The fixations and nodes around MW are highlighted in pink using diamond shapes. The arrows between the diamond fixations in the page view shows the scanpaths around MW. In (b), the shading of nodes (from dark to light) indicates the presumed reading order. In (c), the vertical lines indicates the time points that the saccade outliers occurred. Red and blue arrows are corresponding to the selected saccade outliers. The pink rectangles and the numbers below show the time ranges around MW. (d) shows the distribution of the numbers of saccade outliers in page sections (each page is partitioned into three equal sections: top, middle and bottom).

Algorithm 11 CLUSTER INDICES Idx = FINDCLUSTERS(V)

```
Create a temporary point set P to store the locations of input points V after each movement
for each point p_i in P do
    p_i = v_i
end for
for each iteration j do
    for each point p_i in P do
        p_i^j = S(p_i^{j-1})
    end for
end for
Create an empty point set C to store the centroids of clusters
for each point p_i in P do
    if the location of p_i is not identical to any point in C then
        Add p_i to C
    end if
end for
Create a set Idx to store the cluster indices
for each point p_i in P do
    for each point c_j in C do
        if the location of p_i is identical to c_j then
            idx_i = j
        end if
    end for
end for
return Idx
```

method compared to the k-means clustering or expectation-maximization (EM) algorithms. In the mean-shift algorithm, the input points are moved to a denser configuration so that they are naturally grouped into clusters. Moving each point p to a new location is based on the locations of its neighbors

$$S(p) = \frac{\sum_j \ker(p - p_j) p_j}{\sum_j \ker(p - p_j)}, \tag{7.1}$$

where the kernel function estimates the contribution of each neighbor p_j, i.e.,

$$\ker(d) = \exp\left(\frac{-{d_x}^2 - {d_y}^2}{\varepsilon^2}\right), \tag{7.2}$$

where d is the Euclidean distance between p and p_j, and d_x and d_y are the projections of d along the x and y directions, respectively. ε is a given threshold and the points with a distance to p larger than ε will not be considered. Since our data are from text reading experiments, the x and y directions are related to word length and line spacing, respectively,

we revise the kernel function to

$$\ker(d) = \exp\left(\frac{-{d_x}^2}{\varepsilon_x^2}\right) \times \exp\left(\frac{-{d_y}^2}{\varepsilon_y^2}\right), \tag{7.3}$$

where ε_x is the average word length and ε_y is the line spacing.

Algorithm 11 first moves the input points V to a denser configuration P. This process stops when the number of iterations reaches a user-defined threshold or the locations of points in P do not change. The points in P that move to the same location become a cluster. The locations are treated as the centroids of clusters. We assign the points to clusters by measuring the distances between the points and the centroids.

7.3.3 Transition Graph

After clustering all fixations of each page using the mean-shift algorithm, we construct a transition graph. Each node in the graph denotes a fixation cluster. A directed edge between two nodes represents a transition. A transition $i \rightarrow j$ occurs between two clusters i and j if there is a saccade from a fixation in i to another fixation in j. The transition frequency $f_{i\rightarrow j}$ is the number of transitions from i to j. The directional transition probability $p_{i\rightarrow j}$ is the proportion between $f_{i\rightarrow j}$ and the total number of transitions from i to all the clusters (including itself). As such, a transition indicates a chance for one cluster to transfer to another, and its probability measures how high the chance is. To draw the transition graph, we modify the Fruchterman-Reingold algorithm [44] to create the graph layout. In our modification, we consider the transition probability when computing the attractive forces, therefore, two nodes with strong transitions will be placed closely. However, the nodes may overlap with one another due to their sizes in the drawing. To reduce the overlap while preserving the overall graph structure, we follow the layout adjustment solution given by Gu and Wang [51] by first triangulating the graph and then applying four additional forces (bidirectional, unidirectional, spring, and attractive forces).

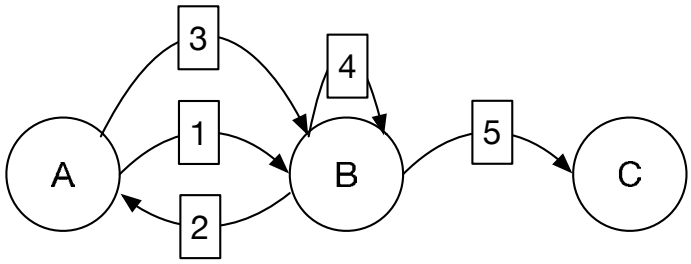


Figure 7.2. An example of the scanpath in the transition graph. The scanpath is *ABABBC*, the corresponding string is *ABABC*, and the repeated scanpath is *AB*.

Algorithm 12 SUFFIX TREE T = CONSTRUCTSUFFIXTREE(S, n)

```
Create a temporary string S' where a unique symbol is added at the end of S
Create a tree T with an empty root
for i from 0 to n do
    substring s = S'[0, i]
    for j from 0 to i + 1 do
        if s[j, i] starts from root to a leaf edge then
            Add s[i + 1] to the leaf edge
        else if s[j, i] starts from root and ends at a non-leaf edge, but s[i + 1] is not the next character of the
        edge then
            A new leaf edge is created for s[i + 1] from the separation
        end if
    end for
end for
return T
```

7.3.4 Repeated Scanpath Detection

The constructed transition graph is an abstraction of the whole scanpath. The transition graph could not be used to visualize consecutive transitions between fixations of the whole scanpath because each edge in this graph is a group of saccades that exist between a pair of fixation clusters. To solve this problem, we convert the scanpath into a string, and this string stores all the transitions between graph nodes. Since we are more interested in transitions between nodes, we ignore all the transitions within the same node. Figure 7.2 shows an example of the scanpath in the transition graph. We first assign IDs to the nodes, therefore, the scanpath is *ABABBC*. Since we ignore all the self-transitions, the

corresponding string becomes *ABABC*. By analyzing this string, we can detect interesting phenomenas, such as area-revisitings (repeated characters, *A* and *B*), repeated scanpaths (repeated substrings, *AB*), and similar behaviors between two participants (common substrings of two given strings). To detect these phenomena, we utilize the suffix tree [122] to identify the repeated substrings in a given string. This suffix tree is a tree structure that stores all the suffixes of a given string. Each edge represents a substring. Therefore, all the non-leaf edges in the suffix tree represents repeated substrings. Constructing and searching this suffix tree occurs in linear time which allows for efficient queries. We first add a unique symbol at the ending of the given string to become a new string. This symbol is used to indicate the ending of the given string. At each iteration, we consider a substring from string indices 0 to i, where i increases from 0 to $n-1$, and n is the new string length. Then, within each iteration, all the suffixes of the substring are inserted into the suffix tree as shown in Algorithm 12. To identify the common substrings of two given strings, we further add another special symbol between the two given strings and use the whole string as an input for the suffix tree. To allow users to focus on prominent substrings, we removed the substrings which are substrings of others, or less than a given length.

7.3.5 Edge Bundling

To show the overall trend of a given graph, we apply the force-directed edge bundling algorithm [59] proposed by Holten and van Wijk [59] to bundle the edges. Their algorithm first partitions all the edges into the same number of segments. Each segment point is assigned an index to indicate its position in the edge. In addition, each segment point is attracted by two kinds of forces: F_e and F_s as shown in Figure 7.3 (a). F_e occurs between two points of the same index at different edges. This force is used to bend the edges. F_s occurs between two adjacent point pairs on the same edge. This force is used to balance F_e. The force on point p_i is calculated as follows

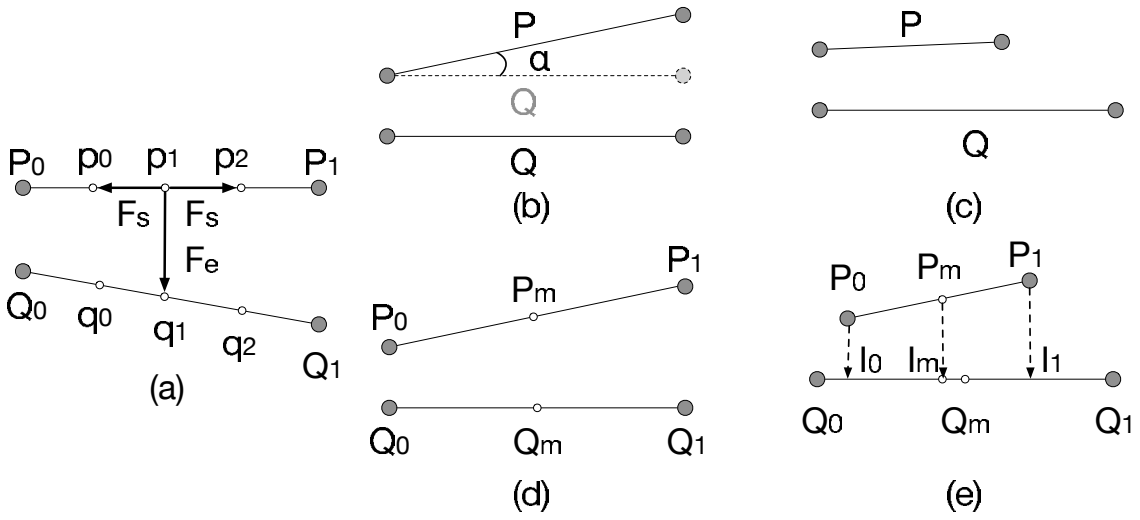


Figure 7.3. (a) Lines P_0P_1 and Q_0Q_1 are partitioned into four segments. Each segment point p_i are attracted by its adjacent segment points p_{i-1} and p_{i+1} by F_s and its corresponding segment point q_i by F_e. (b) Angle compatibility. (c) Scale compatibility. (d) Position compatibility. (e) Visibility compatibility.

$$F_{p_i} = k_p \times (||p_{i-1} - p_i|| + ||p_i - p_{i+1}||) + \sum_{Q \in E \setminus P} \frac{C(P,Q)}{||p_i - q_i||}, \tag{7.4}$$

where k_p is a local constant that denotes the stiffness of edge P, $C_e(P,Q)$ measures the compatibility between edges P and Q, and E is the set of all the edges. $k_p = K/(||P|| \times n)$, where K is a global constant, $||P||$ is the initial length of edge P, and n is the number of segments. The compatibility $C(P,Q)$ is used to control the amount of interactions between edges. As shown in Figure 7.3, we consider the following four kinds of compatibility:

- The edges that are almost perpendicular should not be bundled. Therefore the *angle compatibility*

$$C_a(P,Q) = |\cos(\alpha)| \tag{7.5}$$

  is introduced, where

$$\alpha = \arccos\left(\frac{P \cdot Q}{||P|| \cdot ||Q||}\right), \tag{7.6}$$

as shown in Figure 7.3 (b).

- The edges whose lengths are considerably different should not be bundled. Therefore the *scale compatibility*

$$C_s(P,Q) = \frac{2}{l_{avg} \cdot \min(||P||, ||Q||) + \max(||P||, ||Q||)/l_{avg}} \tag{7.7}$$

is introduced, where

$$l_{avg} = \frac{||P|| + ||Q||}{2} \tag{7.8}$$

as shown in Figure 7.3 (c).

- The edges that are distant away should not be bundled. Therefore the *position compatibility*

$$C_p(P,Q) = \frac{l_{avg}}{l_{avg} + ||P_m - Q_m||} \tag{7.9}$$

is introduced, where P_m and Q_m are the midpoints of P and Q respectively, as shown in Figure 7.3 (d).

- In Figure 7.3 (e), P and Q should not be bundled although they are parallel, equal in length, and close together. That is because bundling them would stretch edges too much. As a result, the *visibility compatibility*

$$C_v(P,Q) = \min(V(P,Q), V(Q,P)) \tag{7.10}$$

is introduced, where

$$V(P,Q) = \max\left(1 - \frac{2||P_m - I_m||}{||I_0 - I_1||}, 0\right), \tag{7.11}$$

and I_m is the midpoint of $I_0 I_1$.

The compatibility $C(P,Q)$ is the multiplication of the above four compatibility values, i.e.,

$$C(P,Q) = C_a(P,Q)C_s(P,Q)C_p(P,Q)C_v(P,Q). \tag{7.12}$$

At the beginning of computation, we treat each edge as one segment. Then we divide each segment evenly into two segments and run a few iterations of bundling. This process terminates until a certain requirement is met. Finally, a smoothing strategy is used to smooth the edge bundles. The force-directed edge bundling algorithm [59] terminates the process when the number of segments reaches 32. In our implementation, we terminate when the number reaches 64 because it would generate a finer results.

7.4 SPSP and SPMP

To help users better understand detailed reading behaviors for SPSP, we provide several query functions, e.g., *saccade outlier detection*, *MW highlighting*, *repeated scanpath detection*, *graph filtering*, and *path animation*.

Saccade outlier detection automatically identifies the saccade outliers that traverse a large distance (larger than a given threshold) along the x or y direction. These saccade outliers indicate long eye movements which may indicate abnormal reading patterns and possible MW. We further differentiate backward- and forward-reading saccades. Backward-reading saccades indicate revisiting earlier portions of the text while forward-reading saccades may indicate foreshadowing. Figure 7.1 shows an example of saccade outlier detection. Specifically, we show the information of saccade outliers in each of the four views.

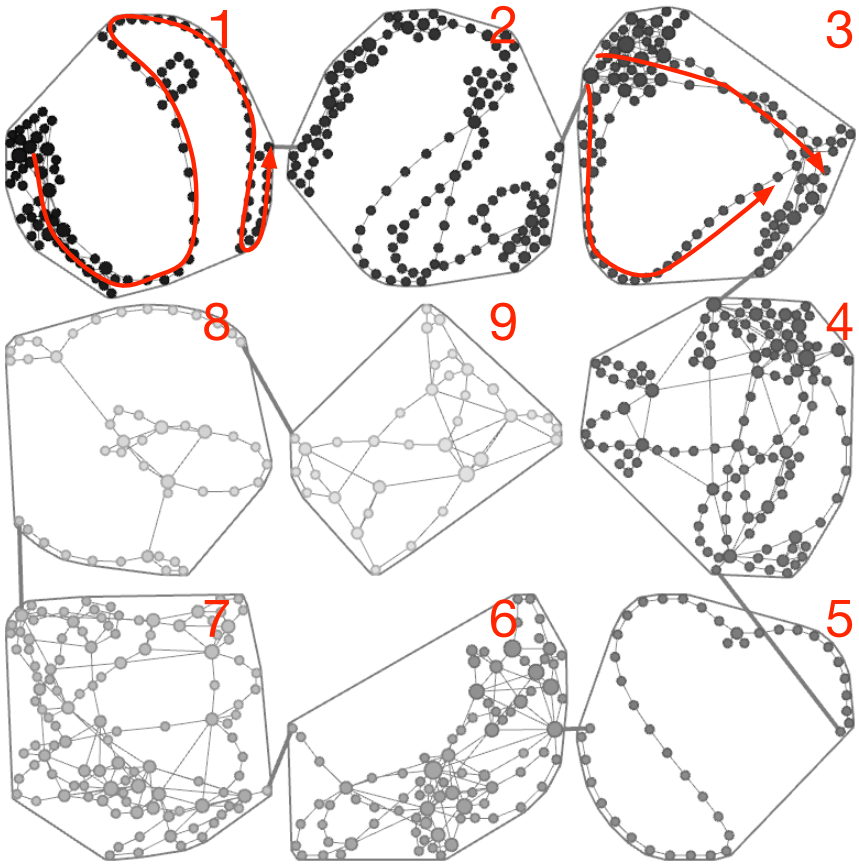


Figure 7.4. The supergraph for SPMP. Each subgraph corresponds to the transition graph of each page. The subgraphs are marked with their page numbers and are placed in a spiral along the clockwise order. In addition, we rotate each subgraph to reduce the length of the edge that connects two adjacent pages. The shading of nodes (from dark to light) indicates the presumed reading order.

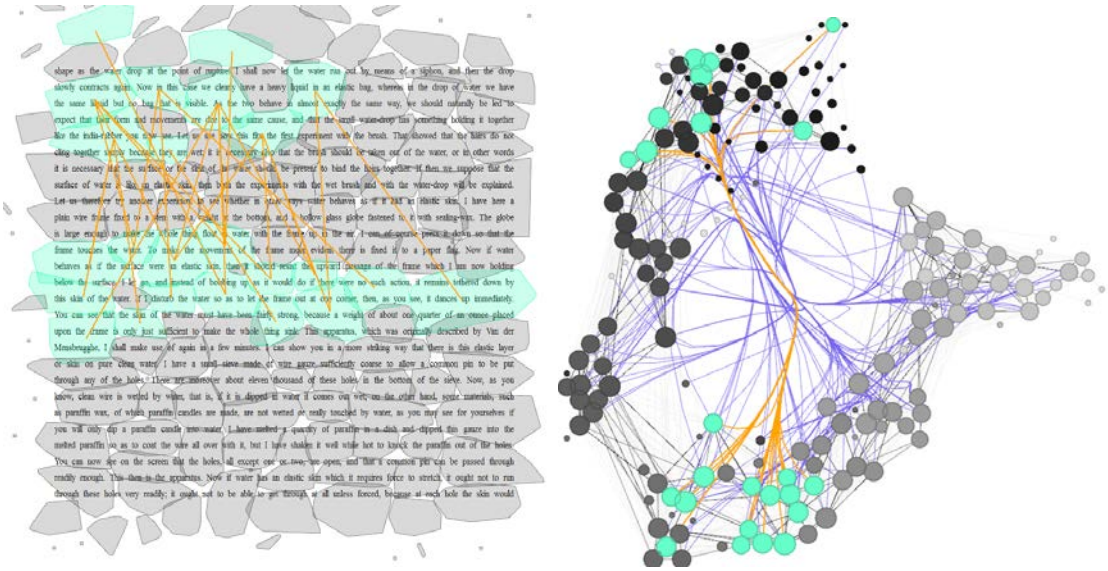
MW highlighting visualizes the scanpath immediately before and after instances of MW. As shown in Figure 7.1 (a) and (b), the scanpath and its corresponding subgraph are highlighted in pink. This function provides a way for users to study MW in the page, graph, and time views.

Repeated scanpath detection automatically detects repeated scanpaths. We allow users to visualize them and find out their corresponding locations and time periods from the graph, page, and time views. In addition, we allow users to play back the scanpaths to compare similarities and differences in scanpaths between different pages and different participants.

Graph filtering allows users to hide nodes (fixations) and edges (saccades) that are not of interest. As users select one or a group of nodes, we automatically hide the nodes that are beyond a given distance to the selected node(s). The graph distance between two nodes is calculated using Dijkstra's algorithm. If two nodes that are not supposed to be connected in the presumed reading order are linked in the graph view, this indicates that at least one saccade outlier is present. Graph filtering also allows users to filter the graph based on time information to analyze the corresponding subgraph.

Path animation provides users the convenience of reviewing scanpath animation. We identify the starting and ending fixations f_s and f_e for the animation. By default, they are the first and last fixations in the page. However, users are allowed to select fixations from the page view or nodes from the graph view for f_s and f_e. If the user selects two nodes from the graph view, we identify the first fixation in the first node as f_s and the last fixation in the second node as f_e. If the two fixations selected do not follow the actual reading order, we swap f_s and f_e. If the user selects a node from the graph view and a fixation from the page view, we ensure that f_s occurs before f_e in the actual reading order.

To understand and compare different behaviors on different pages (SPMP), we generate a supergraph that displays the transition graphs of all pages. An example is shown in Figure 7.4. The transition graphs are arranged clockwise in a spiral shape. To reduce edge crossing



(a) (b)

Figure 7.5. Edge bundling for MPSP. (a) The saccades of the selected bundles illustrate a close relationship between the two green areas in the page view. (b) The bundled edges are highlighted in blue while user-selected edge bundles are shown in orange. The regular (non-outlier) edges are shown in gray with higher frequency edges shown in darker gray.

between pages, we first fix the positions of the first and last nodes at the middle-left and middle-right part of each subgraph. Then we rotate each subgraph one by one to reduce the length of the edge connecting the adjacent pages.

7.5 MPSP and MPMP

For MPSP, we cluster fixations on each page with all the fixations of all participants on that page. After clustering, we construct a supergraph consisting of all the clusters of all participants of that page. As a result, edges with high frequencies form the major reading trend of all participants. ETGraph encodes higher frequency edges with darker gray colors so that users can easily understand the overall reading patterns. However, it is difficult to notice the trend of saccade outliers since those edges are usually drawn with lighter gray

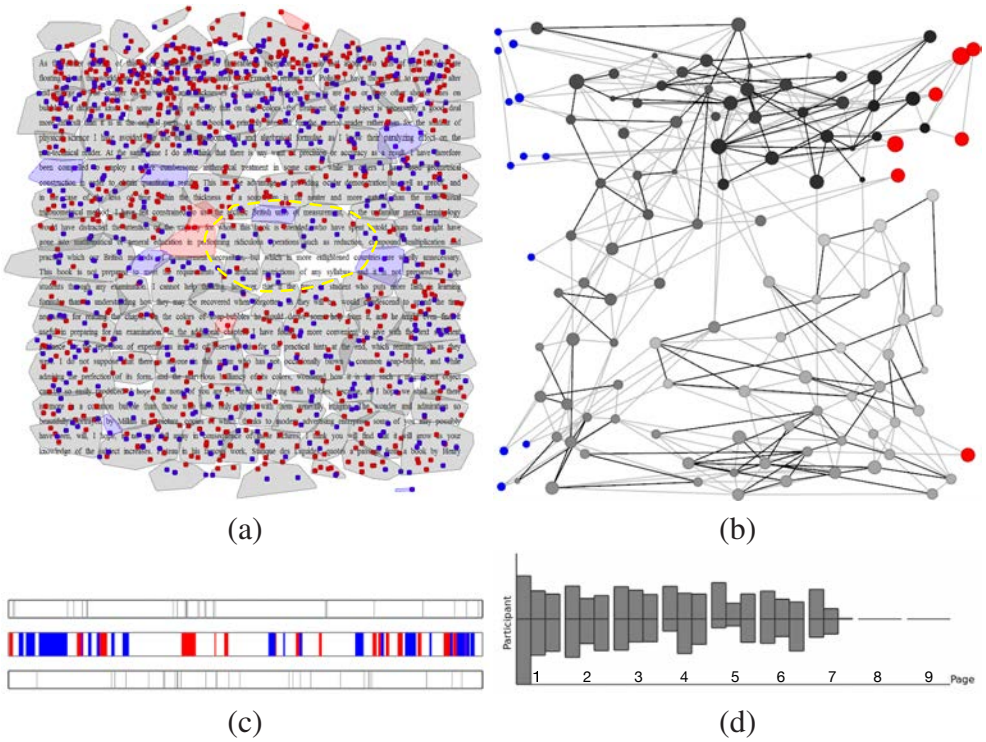


Figure 7.6. Comparison of two participants with similar fixation distributions. (a) The fixation distributions of the two participants. Note that both participants have few fixations in the middle area (highlighted in yellow). (b) Blue and red nodes belong to a single participant only and gray nodes belong to both participants. Dark edges belong to both participants. (c) Saccade outliers for the two participants are shown in the upper and bottom time views, while their shared repeated scanpaths are shown in the middle view. (d) A comparison of the statistics of fixations of the three sections of each page for the two participants. The participant shown in red nodes is at the top while the participant shown in blue nodes is at the bottom.

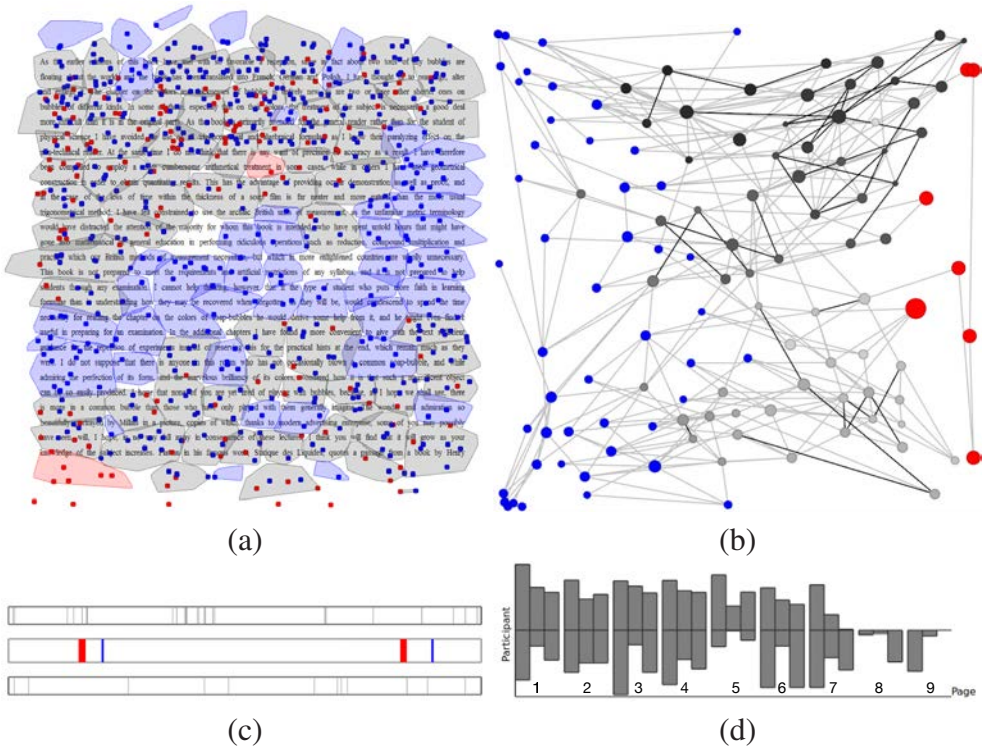


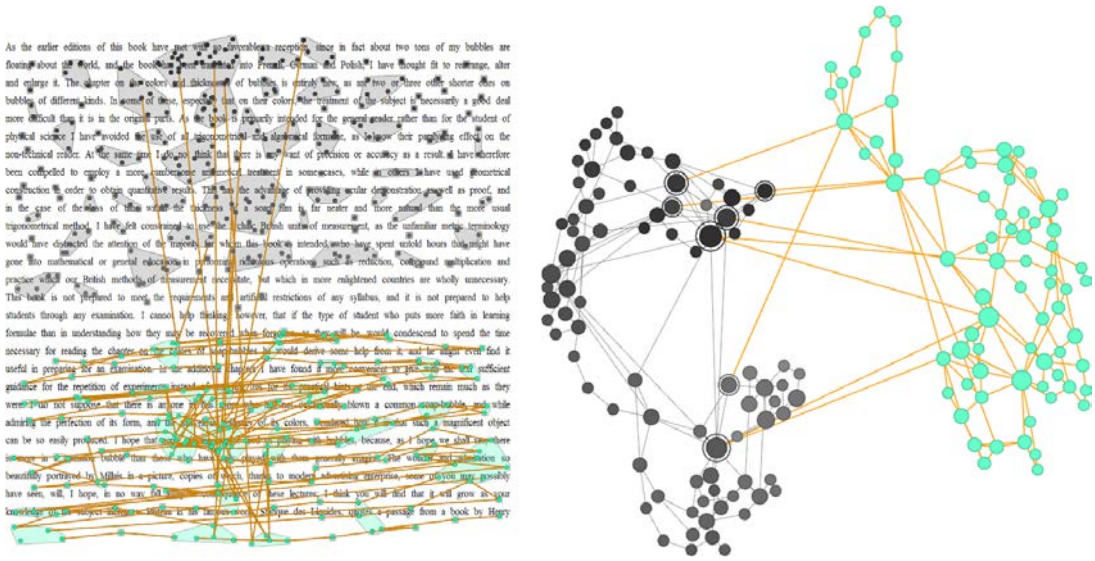
Figure 7.7. Comparison of two participants with different fixation distributions. (a) The fixation distributions of the two participants. Notice that the participant in red have no fixations in the middle area. (b) Blue and red nodes belong to a single participant only and gray nodes belong to both participants. Dark edges belong to both participants. (c) Saccade outliers for the two participants are shown in the upper and bottom time views, while their shared repeated scanpaths are shown in the middle view. (d) A comparison of the statistics of fixations of the three sections of each page for the two participants. The participant shown in red nodes is at the top while the participant shown in blue nodes is at the bottom.

colors due to their low frequencies. To highlight saccade outliers and identify their trend, we apply the edge bundling technique to bundle saccade outliers as shown in Figure 7.5.

It is also important to cluster and compare participants in order to analyze their similarities and differences. We provide two approaches. First, we utilize stacked bar charts and histograms in the time and statistics views to show explicit information (e.g., section length in milliseconds, saccade outlier distribution) for comparison. Second, we calculate the similarities between participants based on the graph information. For MPSP, the transition graph of a participant for a page is a subgraph of the supergraph. Therefore, by comparing the similarities between two subgraphs, we could calculate the similarities between these two participants. The difference between two participants can also be calculated based on their fixation distributions, repeated scanpaths, etc. For each type of difference, we construct a distance matrix. We then normalize each distance matrices and add them together to form the final distance matrix. Finally, we utilize the k-means algorithm to cluster participants. The number of clusters n_c is chosen as $n_c = \sqrt{N/2}$, where N is the number of participants. Based on the clustering, we allow users to select two participants for comparison as shown in Figures 7.6 and 7.7. For MPMP, the distance between two participants is the sum of their distances across all pages.

7.6 Results

In the section, we first report the data set and timing performance. Then we demonstrate the results for SPSP, SPMP, MPSP and MPMP as well as the benefits and knowledge gained using ETGraph. For the 13 research questions (**Q1** $\sim$ **Q13**) grouped into six categories (**C1** $\sim$ **C6**), we add a note such as (**C1**-**Q1**) to show which category and which question each case study result answers.



(a) (b)

Figure 7.8. The transition graph of SPSP view showing the clear separation of two parts in reading by the participant. An interesting repeated rereading pattern is also identified in green.

7.6.1 Data Set and Timing Performance

The data set was generated from eye gaze data collected while participants read an excerpt from a book entitled "Soap-bubble and the Forces which Mould Them" [13]. This text was chosen as it was on a novel topic which would be relatively unfamiliar to a majority of readers. Eye gaze data was collected with a Tobii TX300 remote eye tracker affixed below a monitor set to a resolution of 1920×1080, which displayed the text. The excerpt consisted of text from the first 35 pages of the book and contained around 5700 words across 10 pages. Each participant read the text for 20 minutes, and not every one was able to finish reading all pages. Few participants read the final page (Page 10), so it has not been included in our analysis. Eye gaze data were collected from both eyes and the data from each eye was filtered and averaged together prior to eye movement detection. The data were then converted into a series of fixations using a dispersion based filter. Using

TABLE 7.1

THE TIMING RESULTS AND PARAMETERS

	mean shift	layout generation	layout adjustment	repeated scanpath	edge bundling
single participant (SP)	1.283	18.941	3.968	0.143	-
multiple participants (MP)	28.433	0.951	0.244	-	1.193

the Open Gaze And Mouse Analyzer (OGAMA) [126], the filter was set to detect fixations if there were consecutive gaze points within a range of 57 pixels (approximately 1 degree of visual angle) for longer than 100 ms, which is the shortest duration for naturalistic eye movements during reading [58, 100]. Saccades were then calculated from the fixations. The timing was collected on a PC with an Intel 3.6GHz CPU and 32GB Memory. The processed data set consists of 27 participants and 9 pages. The timing result is shown in Table 7.1.

7.6.2 SPSP

Figure 7.8 shows an example of SPSP. In (a), each dot in the page view represents a fixation and the fixation clusters are highlighted using convex hulls. In (b), each node in the graph view represents a fixation cluster and an edge represents a transition between two clusters. In the graph, the nodes with stronger transitions are placed closer to each other. The graph view provides an overview of the reading pattern for a single page. The darkness of nodes (from dark to light) indicates the presumed reading order. Therefore, in general, nodes with similar darkness values are placed nearby. We can observe that the

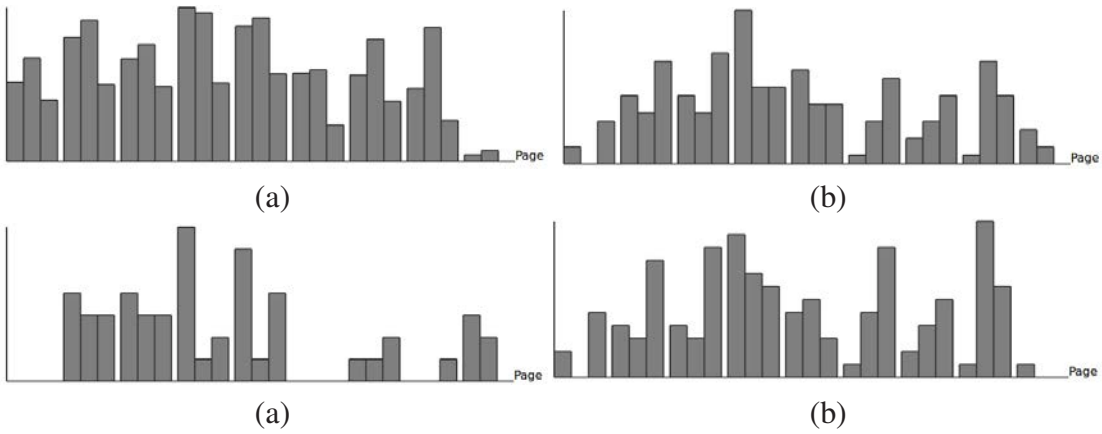
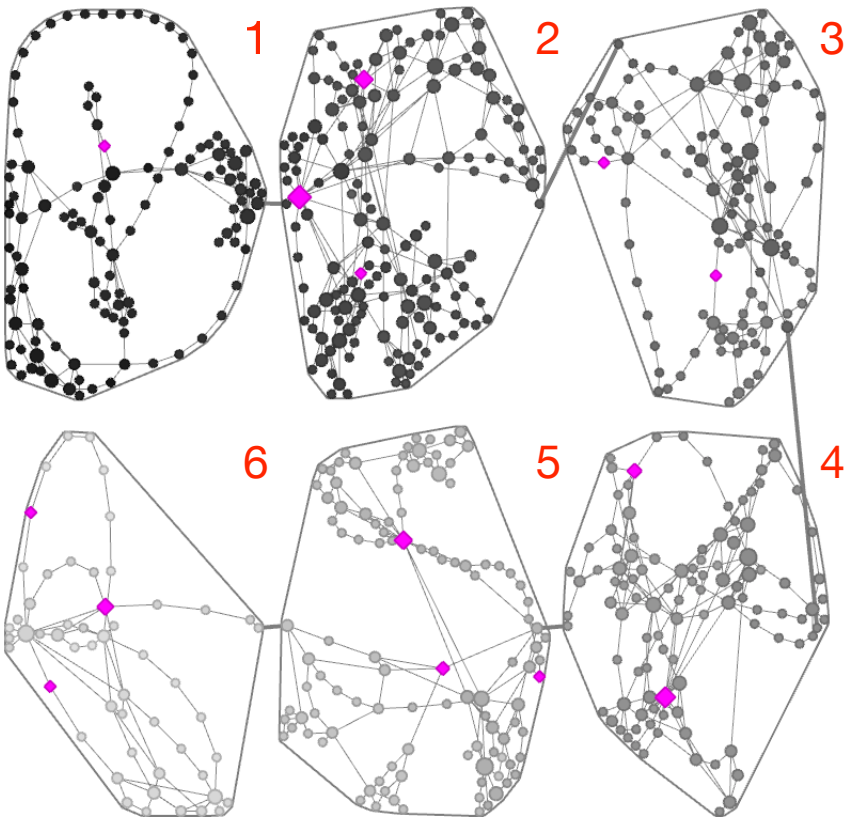


Figure 7.9. Frequency distributions of (a) fixations, (b) saccade outliers, (c) saccade outliers with large *y* distances, (d) saccade outliers with large *x* distances.

nodes in (b) are clearly separated into two groups. The nodes in one group are in green and their corresponding fixations are located in the lower portion of the page as shown in (a). The separation between these two groups of nodes could indicate that this participant read the portion of text at the top of the page separately from the portion of text at the bottom of the page. Although some saccade outliers connect the top and bottom portions of the page, more connections exist within the two portions. This indicates that there are stronger connections within each portion than between the two portions. Of particular interest are the nodes in (b) that are not selected but are connected to nodes that are. The corresponding fixations are located in the first few lines of the page, which could indicate that the participant always returned to this location of the page to reread (**C1**-**Q1**).

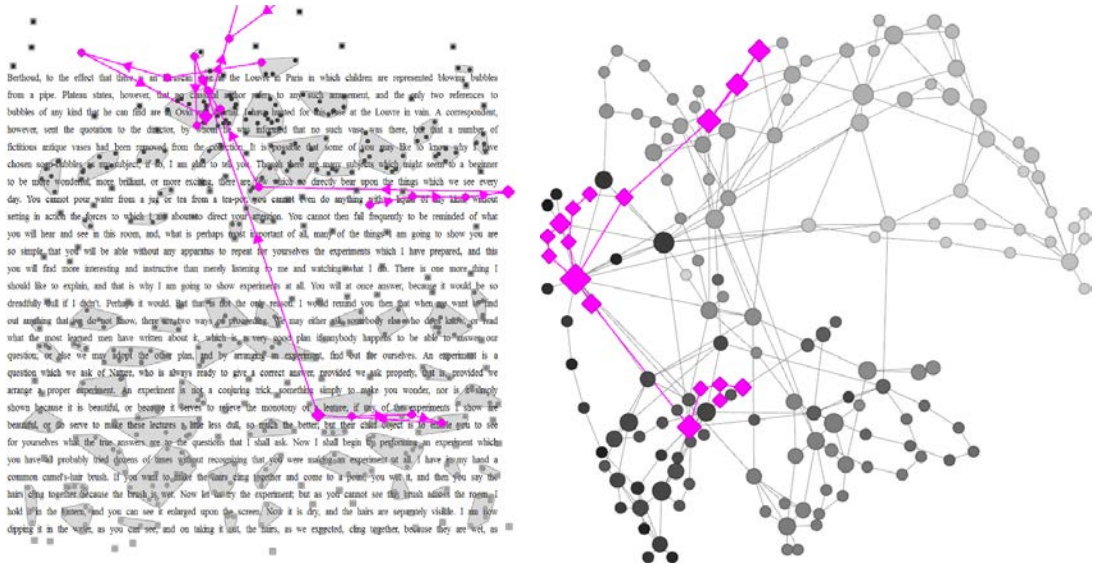
To understand the scanpath structures, besides regular reading patterns, it is important to analyze saccade outliers. They could indicate a portion of text that is difficult to understand or a rereading pattern. In Figure 7.1 (a) and (b), the fixation clusters and nodes of the saccade outliers are in yellow. The selected clusters and nodes are in green. Red and blue edges indicate backward and forward saccade outliers, respectively. Most of the saccade outliers in Figure 7.1 (a) are targeted at or moving from the upper portion of text.



(a)

Figure 7.10. The SPMP view and the pink nodes indicate the occurrence of MW.

This could indicate that the participant reread this portion of text. In addition, there is a saccade outlier from the middle of the page toward the bottom of the page (outside of the screen), but the connected nodes of the saccade outlier are close in Figure 7.1 (b), which demonstrate that ETGraph gathers the nodes based on their transition relationships instead of their spatial closeness. In the time view of Figure 7.1 (c), the vertical lines indicates the time slots where saccade outliers occurred. The saccade outliers are displayed in red and blue. Figure 7.1 (d) shows the distribution of the saccade outliers in each of the three sections of the page. This distribution shows that this participant had a large number of saccade outliers in the first section of the page (**C2-Q3**).



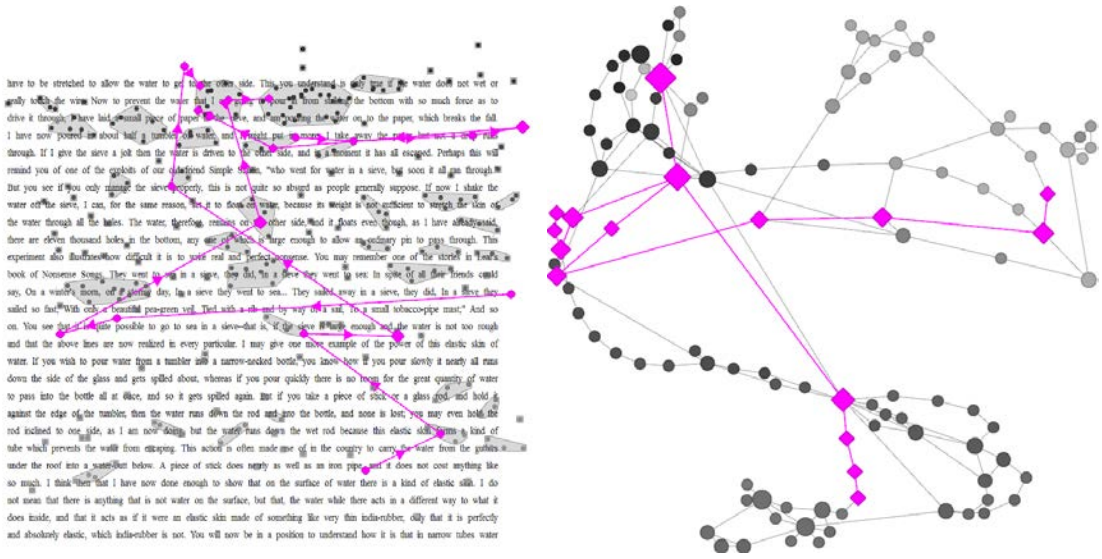
(a) (b)

Figure 7.11. (a) and (b) are the page and graph views of Page 2 of the participant in Figure 7.10, respectively.

7.6.3 SPMP

An analysis of the reading patterns of a participant for multiple pages can be done by studying the supergraph of the chosen pages along with their statistical information. Figure 7.4 displays a supergraph of nine pages for a single participant. The transition graphs of Pages 1, 5 and 8 are quite simple since each transition graph forms a smooth curve. The graph of Page 3 is very interesting because it consists of two paths from the beginning to the end, which could mean that this participant read the page twice. However, this graph structure is still simple compared to the graphs for Pages 4, 6 and 7. These graphs consist of complex relationships between nodes which could indicate that the participant read these pages backward and forward many times (**C1**-**Q6**). Shifting to the corresponding SPSP view allows for a more detailed examination of these pages, which could offer additional insights.

SPMP also allows a comparison of pages by displaying statistical information for each



(a) (b)

Figure 7.12. (a) and (b) are the page and graph views of Page 5 of the participant in Figure 7.10, respectively.

page, which could indicate consistency of reading patterns across all pages. The charts in Figure 7.9 (a), (b), (c), and (d) show the distribution of fixations, saccade outliers, saccade outliers with large *y* distances, and saccade outliers with large *x* distances, respectively. In (a), the fixation distributions are quite similar among the first seven pages, which indicates similar reading patterns. In (b) and (d), the saccade outlier distributions are similar between Page 2 and Page 3, Page 4 and Page 5, and among Page 6, Page 7 and Page 8. In (b), (c) and (d), the saccade outlier distributions are similar between Page 2 and Page 3, Page 4 and Page 5. We conclude that there were similar reading patterns between these pages, especially adjacent pages (**C6**-**Q7**).

Besides showing the structures of the scanpaths, ETGraph may be useful to find the patterns of MWs. Figures 7.10, 7.11, and 7.12 show MWs of a participant. In Figure 7.10, the pink nodes indicate the appearance of MWs. Figure 7.11 (a) and (b) show the page and graph views of Page 2, respectively. Figure 7.12 (a) and (b) show the page and

graph views of Page 5, respectively. In Figures 7.11 (a) and 7.12 (a), the scanpaths around the MWs consist of saccade outliers. So, in Figures 7.11 (b) and 7.12 (b), the subgraphs around the MWs span a large area. We can see that ETGraph could provide a hint for users to detect pages consisting of MWs. If there are more than one episode of MW per page, the subgraphs of MWs may consist of saccade outliers and have overlap between them (**C4**-**Q5**).

7.6.4 MPSP

To show the overall reading patterns of MPSP, we bundle the edges to observe their trends, as shown in Figure 7.5. In (b), only the saccade outliers are bundled since bundling all edges would hide the trend in the saccade outliers. The regular (non-outlier) edges are shown in various gray colors and the darkness of each edge shows its transition frequency. These gray-color edges give an impression of common reading patterns (**C1**-**Q8**). In contrast, the saccade outliers are bundled and highlighted in blue. Nodes or bundles can be selected to see their corresponding fixations and saccades. The selected bundles are shown in orange and their connected nodes are shown in green. The corresponding page view in (a) illustrates a strong relationship between the areas highlighted in green.

Besides showing the overall patterns of all participants, we can also cluster participants based on their attributes, e.g., fixation distribution, graph similarities, and repeated scanpaths (**C5**-**Q11**). Figure 7.6 shows a comparison of two participants who are in the same cluster based on fixation distribution. In (b), blue nodes belong to one participant and red nodes belong to another, while gray nodes belong to both of them. The dark edges belong to both participants and the gray ones do not. In this example, most of the nodes and a large number of edges are shared, which indicates that their corresponding graphs are also quite similar. From (a), we can see that the two participants share a lot of fixation clusters shown in gray. The clusters that differ are located at the boundaries of the page, shown in blue and red. In addition, they both have a relative lack of fixations in the middle area,

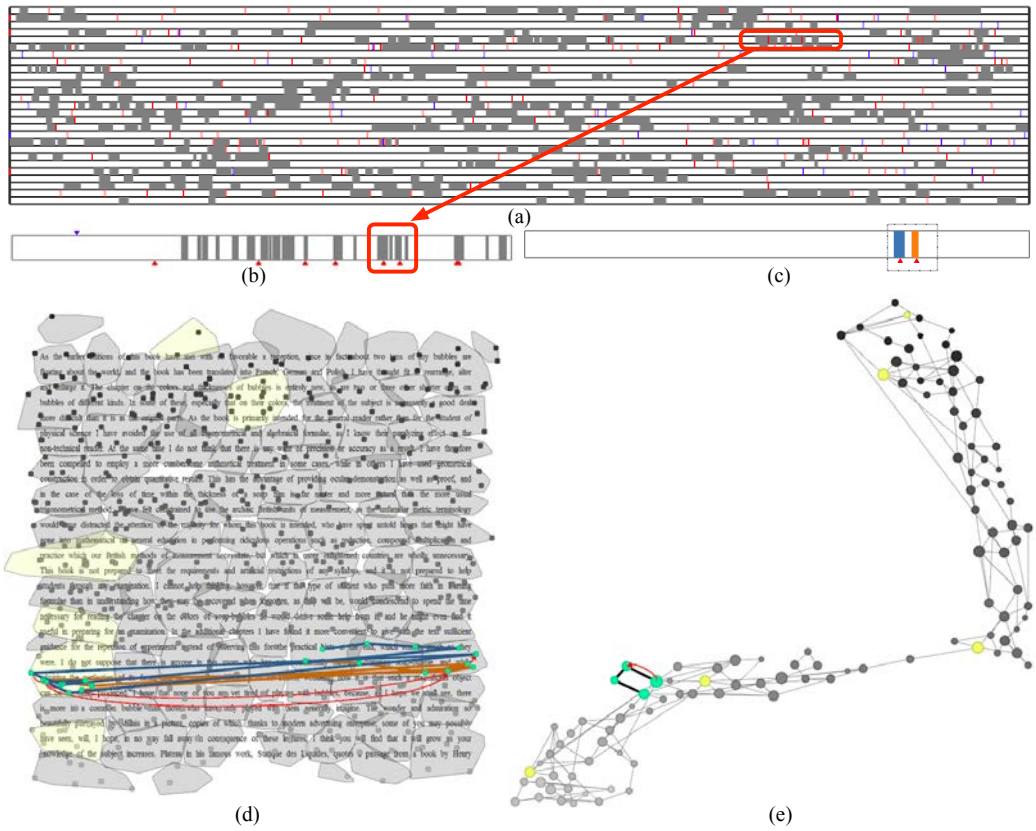


Figure 7.13. (a) The time view of saccade outliers and repeated scanpaths for MPSP, organized in a stacked bar chart. Repeated scanpaths are shown in gray while saccade outliers are highlighted in red and blue bars, indicating backward and forward saccade outliers, respectively. The red rectangle marks the repeated scanpaths that overlap with saccade outliers. (b) A participant is selected from (a) and the corresponding time view is shown. (c) The repeated scanpath is selected. (d) and (e) are the corresponding page view and graph view for the repeated scanpath, respectively.

highlighted in yellow. (c) displays the time of saccade outliers in the gray time views at the top and bottom, and the shared repeated scanpaths in the middle time view. There are a large number of colored bars which shows that the two participants shared a large number of repeated scanpaths. This further indicates that the two participants had similar reading patterns. (d) compares the fixation distributions of all pages and similar fixation distributions can be observed (**C6-Q2**, **C6-Q9**).

Figure 7.7 shows a comparison of two participants who are in two different clusters based on fixation distribution. In (b), the number of blue nodes is much larger than the red nodes. From (a), we can see that there is a large number of blue fixation clusters in the middle of the page and there are no red fixation clusters, which indicates that the participant in red did not read those portions at all. In (c) there are only four colored bars which shows that the two participants only shared two repeated scanpaths. This further indicates that the two participants have different reading patterns. (d) compares the statistics of fixations of all the pages and shows different distributions (**C6-Q2**, **C6-Q9**).

We also provide statistical information to help users identify similarities and differences between all participants. For example, in our data, we found that the saccade outliers with large y distances do not occur in any repeated scanpaths for all participants. This indicates that the participants did not revisit two portions of text that have a large y distance. However, it is not the case for saccade outliers with large x distances. Figure 7.13 (a) shows the repeated scanpaths and saccade outliers with large x distances. The repeated scanpaths are shown in gray while the saccade outliers are highlighted in red and blue bars indicating backward and forward saccade outliers, respectively. Repeated scanpaths occur when a participant rereads some portion of text. A repeated scanpath may represent a scan of the text or a return to a particular sentence (**C3-Q4**). When a participant tries to understand a large paragraph and a repeated scanpath is only part of it, then this repeated scanpath is only a scan. This is different from when the time gap between the corresponding scanpaths of a repeated scanpath is short and there is a saccade outlier between them. In Figure 7.13,

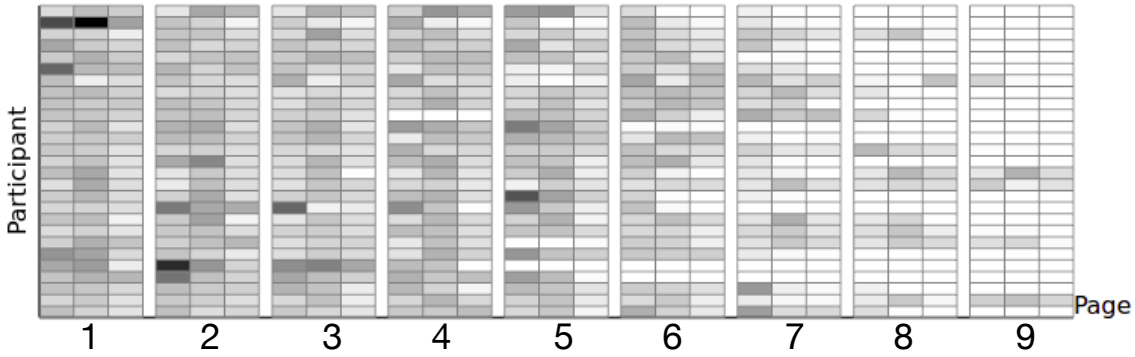


Figure 7.14. The statistics view of time usage for MPMP. In the stacked bar chart, there are a total of nine pages and each page is partitioned into three equal sections: top, middle and bottom. The shading of a section (from dark to light) indicates the time spent on reading it (from more to less).

we can see from Figure 7.13 (a) that there are some repeated scanpaths overlapped with saccade outliers with large x distances highlighted in the red rectangle. These repeated scanpaths may be more interesting than others. In this figure, participant's repeated scanpaths that consist of saccade outliers are shown in the red rectangle. The corresponding time view is shown in Figure 7.13 (b). The corresponding selected repeated scanpaths are shown in Figure 7.13 (c). The corresponding graph and page views are shown in Figure 7.13 (d) and (e), respectively. The page view shows that this participant reread this sentence twice, ostensibly for better understanding.

7.6.5 MPMP

Figure 7.14 shows a summarization of the time information for MPMP. It can be seen from the stacked bar chart that most of the participants did not finish reading all the pages. In addition, those participants that behaved differently (e.g., have sections that are relatively darker) may warrant further analysis (**C6**-**Q12**).

7.7 User Study and Expert Evaluation

7.7.1 User Study

We recruited five unpaid PhD students in our university to evaluate the effectiveness of ETGraph. One student is from the Department of Psychology and four students are from the Department of Computer Science and Engineering. All five users are studying MW in their respective PhD studies using different kinds of data, e.g., physiology, facial features or eye gaze. The user study was conducted in a lab using the same PC for each user. The users were first introduced to ETGraph and were instructed about its design goals and main functions. Then they were given ten minutes for free exploration to get familiar with the system. After that, they were asked to complete six tasks and a survey of seven general questions on the design of ETGraph. These tasks were written on paper and the users hand wrote their responses. The observer (i.e., one author of this work) stood near by and took notes. After the study, he then interviewed the users about their thoughts of the tasks and ETGraph.

As shown in Table 7.2, **T1** asked the users to compare three given graphs and observe their corresponding scanpaths. The task evaluated if the user was able to identify the graph with an abnormal scanpath compared to the other two. **T2** asked the user to circle clusters in the graph and observe their corresponding portions in the page view. This task evaluated if the user understood that the nodes are placed nearby in the graph due to their strong relationships in the scanpath. **T3** asked the user to estimate which edges in the graph represent saccade outliers and to verify their guesses using ETGraph. This task evaluated if the user understood that saccade outliers with large y distances exist between two nodes with very different degrees of gray-scale colors or have longer edges. **T4** asked the user to watch the scanpath animation and identify whether MW was reported on the page. **T5** and **T6** asked the user to identify whether MW was reported on the page based on the graph structure and saccade outliers, respectively.

TABLE 7.2

THE SIX TASKS IN THE USER STUDY AND USER SCORES OF THE TASKS

task	description	average score	standard deviation
T1	Identify the abnormal reading pattern based on the graph representations, and study their corresponding scanpaths.	1	0
T2	Circle the clusters in the graph and observe their corresponding portions in the page view.	1	0
T3	Circle the saccade outliers with large *y* distances in the graph view and verify your guesses in the page view	0.8	0
T4	Identity the pages with MW based on the animation of the whole scanpath.	0.7	0.21
T5	Identify the pages with MW based on the graph structures.	0.8	0.27
T6	Identify the pages with MW based on the saccade outliers in the graph view.	0.8	0.45

Users could perform the tasks at their own pace. Each session took about 30 to 60 minutes to complete. We summarized the binary task completion scores for the six tasks in Table 7.2.

We note that all users answered **T1** and **T2** correctly. For **T1**, users noticed that when a participant read from left to right and from top to bottom, then the corresponding graph presents a continuous transition from darker nodes to lighter nodes, which conforms to the presumed reading order. However, the graph structure became complex when there were a large number of saccade outliers. For **T2**, users drew circles to highlight the clusters in the graph, and selected each group to verify their estimates. They concluded that if the participants separated the text into portions and read each portion carefully, then each portion would form a cluster in the graph.

For **T3**, users can identify most of the saccade outliers with large y distances based on the edge lengths of the graph, but there was one saccade outlier that they all failed to identify. The edge of that saccade outlier linked two nodes that are close in the graph but one node was very dark and the other was very light. The fixation of the light node was very close to the bottom of the page and the dark node consisted of a lot of fixations at the top of the page. Therefore, the dark node pulled the light one close to itself and the corresponding edge length was small. Failing to identify such an outlier shows that we should have explained the details of ETGraph construction and the layout generation algorithm to users. This would help users better understand the graph so they would know that the edge represents a saccade outlier when they observe such a phenomenon.

For **T4**, users tended to animate the entire scanpath to understand the reading pattern and identify possible MW. However, it was sometimes difficult for users to do this because of visual clutter and the effort of remembering the previous scanpath. To help users keep track of the reading pattern, ETGraph only displays a few of the most currently displayed saccades and all the previous saccade outliers during the animation. The average score of successfully identifying MW was 0.7 for this task and most of the users considered this

task difficult to complete. All users except one made at least one incorrect judgement. Most of the users stated that viewing an animation based on time that included saccade outliers helped them identify rereading behaviors and abnormal reading patterns that may include MW, but this function still requires users to have some knowledge about typical reading patterns associated with MW.

For **T5**, the average score of successfully identifying MW was 0.8. All users stated that the graph structure could help determine whether the participant read normally or was MW. This task consisted of two subtasks. Three users who considered this task easy completed it correctly. The remaining two users only answered one small task correctly. They both agreed that Figure 7.11 (b) that was used in **T5** showed a graph with a normal structure. This result indicates that we need to train users about the graph structure in order to distinguish the differences between graphs with and without MW.

For **T6**, the average score of successfully identifying MW was 0.8. This task consisted of two subtasks. Four users who considered this task easy completed it correctly. The remaining user did not respond correctly. She studied the content connected by the saccade and decided that there was no MW present if the content was related. However, this is not necessarily always the case.

On the general questions, all users agreed that ETGraph was helpful for studying eye-tracking data. Based on the graph structure, they could get an impression on whether the participant followed a regular reading pattern or not. Saccade outlier detection helped them identify saccade outliers and possible MW. Users also had some suggestions to improve ETGraph. Four of them suggested that we should provide more training and explanation of ETGraph for better use. One even suggested that we list pages with or without MW, so that their differences would be more obvious in ETGraph. Two users suggested that we develop an iPad or Windows version for them to explore further as our current system runs on Linux.

7.7.2 Expert Evaluation

We also invited two experts: a professor and a PhD student whose research interest is identifying MW in eye-tracking data. We utilized the think-aloud protocol during the evaluation. The experts described their thoughts while completing the tasks, and we summarized their comments after the evaluation. They both agreed that ETGraph is a very helpful tool for researchers who are interested in studying eye movements.

The professor considered ETGraph a useful tool to identify the reading patterns around MWs. He pointed out some suggestions to improve ETGraph. First, he suggested that screen space should be added for the graph view so that users can select multiple graphs for comparison. Second, he suggested that ETGraph should focus more on saccade outliers with large y distances because they are important to identify rereading and MW. Saccade outliers with large x distances might be due to the poor calibration of eye trackers and line jumps. For MPSP, he thought bundling outliers was helpful to identify the trend of abnormal reading patterns. However, he suggested that we bundle the outliers separately for pages with or without MW. This could help researchers study common patterns of MWs. Finally, he thought that after identifying the common patterns of the graphs with MW, using graph similarity measures could help identify the pages with MW.

The student expert considered ETGraph to be a useful tool because it provides a new approach to visualize eye-tracking data. He has used tools like the Open Gaze and Mouse Analyzer (OGAMA) [126] and has even written his own programs to analyze eye movements. These tools render the data using a scanpath or heatmap. Visual clutter is inevitable when displaying a scanpath, while heatmaps lack saccade information. ETGraph addresses both limitations. In addition, he thought that ETGraph helps to not only visualize eye movements but also gain an understanding of patterns that are not obvious with other tools. In addition, he is particularly interested in the repeated scanpath detection function. For him, this function provides additional information drawn from ETGraph besides saccade outliers that could be used to detect MW. Finally, he thought that our approach could

help to engineer novel features for use in machine learning based on observations drawn from ETGraph.

CHAPTER 8

CONCLUSIONS

This dissertation presents a graph-based visualization approach that aims to assist users in better navigating, exploring, and understanding various high-dimensional, heterogeneous, and complex data. This is achieved by carefully defining nodes and edges in graphs to encode the underlying data relationships and providing multi-view interfaces for effective interaction. Different definitions are designed to meet specific needs according to the distinctive characteristics of a particular type of data. A set of graph-based techniques is developed to filter the nodes and connections among them, so that complex data can be simplified into less cluttered graph structures for cost-effective visual understanding. With these graph representations, users can quickly build mental connections among graph features and data features, and identify important data relationships and track changes over time. Furthermore, this dissertation proposes solutions to tackle the ever-growing data size from multiple aspects. For example, graph hierarchies are constructed to allow users to explore the graph in a coarse to fine manner. Graph mining techniques are applied to simplify graphs and provide guidance for exploration. In addition, parallel algorithms are also implemented for performance speedup.

For time-varying volumetric data, I present three works: TransGraph, a hierarchical graph representation to visualize transition relationships; an extended work based on TransGraph which utilizes graph mining techniques to understand large transition graphs; and iTree for data impacting, indexing and classification. In a single 2D graph view, I design TransGraph that visualizes the transition relationships among data blocks of all time steps, and allows the direction of time evolution easily readable. The correspondences between

volume regions and graph nodes can be inferred through brushing and linking. In other words, TransGraph augments our ability to better understand time-varying volumetric data by providing an occlusion-free overview map that encompasses all time steps, controllable interactions that helps users track data transition over space and time, and adaptive exploration that allows us to go beyond small- and medium-scale data sets. However, the transition relationships presented in TransGraph can still be complicated for large-scale and complex data sets. In the second work, I apply a series of graph mining techniques including graph simplification, community detection, and visual recommendation, to simplify the graph and provide high-level visual guidance for user exploration. The results and evaluation with different data sets demonstrate that such a mining approach points out a promising direction to shield users from undue distraction by the complexity of graphs and thus enables them to concentrate on effective reasoning about the data. In the third work, I introduce iTree for time-varying data impacting and indexing. Built on an indexable hierarchical structure, iTree nicely combines data compacting, indexable data searching and querying with visually adaptive data exploration, a unique feature that makes our framework amenable for tackling large-scale time-varying data sets.

To visualize large image-text collections, I present iGraph that provides essential capabilities including interactive filtering, node comparison, and visual recommendation to support progressive graph drawing and user-driven exploration. The general approach proposed for iGraph is applicable to many other large graph drawing, visualization, and exploration applications. I integrate additional features including the bubble sets to visually differentiate node groups, suggestion of abnormal keywords and time periods, and comparison of different recommendation solutions to enrich the power of iGraph. In addition, I leverage multiple GPUs for preprocessing and multiple CPUs for runtime graph layout generation and interactive visualization. The effectiveness of iGraph is demonstrated through experimental results and a user study.

For the visualization of eye-tracking data, I present ETGraph that transforms eye-

tracking data of a reading study from the original page view to an abstract graph representation. The graph view represents fixation clusters as nodes and saccades as edges. For graph drawing, the positions of nodes are adjusted based on graph layout algorithms. Therefore, instead of indicating the locations of fixation clusters, node positions actually reveal the very nature of the underlying reading pattern. Through brushing and linking, users are able to explore eye-tracking data from multiple perspectives. I demonstrate the usefulness of ETGraph by studying the reading patterns in four different settings: single participant single page, single participant multiple pages, multiple participants single page, and multiple participants multiple pages. The feedback from the domain scientist and a group of student researchers confirms the effectiveness of our approach.

BIBLIOGRAPHY

1. The 2012 dynamical core model intercomparison project (DCMIP). `https://earthsystemcog.org/projects/dcmip-2012/`.

2. L. Akoglu and C. Faloutsos. Event detection in time series of mobile communication graphs. In *Proceedings of Army Science Conference*, pages 77–79, 2010.

3. A. B. Alencar, M. C. F. de Oliveira, and F. V. Paulovich. Seeing beyond reading: A survey on visual text analytics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):476–492, 2012.

4. G. Andrienko, N. Andrienko, M. Burch, and D. Weiskopf. Visual analytics methodology for eye movement studies. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2889–2898, 2012.

5. D. Archambault, T. Munzner, and D. Auber. TopoLayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007.

6. S. Bachthaler and D. Weiskopf. Continuous scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1428–1435, 2008.

7. S. Bachthaler and D. Weiskopf. Efficient and adaptive rendering of 2-D continuous scatterplots. *Computer Graphics Forum*, 28(3):743–750, 2009.

8. B. B. Bederson. PhotoMesa: A zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of ACM Symposium on User Interface Software and Technology*, pages 71–90, 2001.

9. K. Berkner, E. L. Schwartz, and C. Marle. SmartNails: Display- and image-dependent thumbnails. In *Proceedings of SPIE Conference on Document Recognition and Retrieval*, pages 54–65, 2003.

10. A. Biswas, S. Dutta, H.-W. Shen, and J. Woodring. An information-aware framework for exploring multivariate data sets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2683–2692, 2013.

11. T. Blascheck, K. Kurzhals, M. Raschke, M. Burch, D. Weiskopf, and T. Ertl. State-of-the-art of visualization for eye tracking data. In *Eurographics Conference on Visualization - State-of-the-Art Reports*, pages 83–82, 2014.

12. T. Blascheck, M. John, K. Kurzhals, S. Koch, and T. Ertl. VA^2: A visual analytics approach for evaluating visual analytics applications. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):61–70, 2016.

13. C. V. Boys. *Soap-Bubbles, and the Forces which Mould Them*. Cornell University Library, 1890.

14. P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2011.

15. T. M. Breuel, W. C. Janssen, K. Popat, and H. S. Baird. Paper to pda. In *Proceedings of IRPA International Conference on Pattern Recognition*, pages 476–479, 2002.

16. P. Brivio, M. Tarini, and P. Cignoni. Browsing large image datasets through Voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1261–1270, 2010.

17. M. Burch, A. Kull, and D. Weiskopf. AOI Rivers for visualizing dynamic eye gaze frequencies. *Computer Graphics Forum*, 32(3):281–290, 2013.

18. M. Burch, F. Beck, M. Raschke, T. Blascheck, and D. Weiskopf. A dynamic graph visualization perspective on eye movement data. In *Proceedings of Symposium on Eye Tracking Research and Applications*, pages 151–158, 2014.

19. A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *Proceedings of IEEE International Conference on Data Mining*, pages 58–67, 2010.

20. C. Chen, G. Gagaudakis, and P. Rosin. Content-based image visualization. In *Proceedings of International Conference on Information Visualisation*, pages 13–18, 2000.

21. C.-K. Chen, C. Wang, K.-L. Ma, and A. T. Wittenberg. Static correlation visualization for large time-varying volume data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 27–34, 2011.

22. C.-M. Chen and H.-W. Shen. Graph-based seed scheduling for out-of-core FTLE and pathline computation. In *Proceedings of IEEE Symposium on Large-Scale Data Analysis and Visualization*, pages 15–23, 2013.

23. C.-M. Chen, B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Flow-guided file layout for out-of-core pathline computation. In *Proceedings of IEEE Symposium on large Data Analysis and Visualization*, pages 109–112, 2012.

24. C.-M. Chen, L. Xu, T.-Y. Lee, and H.-W. Shen. A flow-guided file layout for out-of-core streamline computation. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 145–152, 2012.

25. R. L. Cilibrasi and P. M. B. Vitányi. The Google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.

26. E. Clarkson, K. Desai, and J. D. Foley. ResultMaps: Visualization for search interfaces. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1057–1064, 2009.

27. C. Collins, G. Penn, and S. Carpendale. Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009.

28. T. Crnovrsanin, I. Liao, Y. Wu, and K.-L. Ma. Visual recommendations for network navigation. *Computer Graphics Forum*, 30(3):1081–1090, 2011.

29. W. Cui, Y. Wu, S. Liu, F. Wei, M. X. Zhou, and H. Qu. Context preserving dynamic word cloud visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 121–128, 2010.

30. M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications*, 9(1):31–52, 2005.

31. S. Diehl and C. Görg. Graphs, they are changing. In *Proceedings of International Symposium on Graph Drawing*, pages 23–30, 2002.

32. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

33. K. Dinkla, M. V. Kreveld, B. Speckmann, and M. Westenberg. Kelp diagrams: Point set membership visualization. *Computer Graphics Forum*, 31(3):875–884, 2012.

34. M. Dörk, N. H. Riche, G. Ramos, and S. Dumais. PivotPaths: Strolling through faceted information spaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2709–2718, 2012.

35. Z. Du, Y.-J. Chiang, and H.-W. Shen. Out-of-core volume rendering for time-varying fields using a space-partitioning time (SPT) tree. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 73–80, 2009.

36. A. T. Duchowski. A breadth-first survey of eye tracking applications. *Behavior Research Methods, Instruments, and Computers*, 34(4):455–470, 2002.

37. C. Dunne and B. Shneiderman. Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of ACM SIGCHI Conference*, pages 3247–3256, 2013.

38. C. Dunne and B. Shneiderman. SimRank: A measure of structural context similarity. In *Proceedings of ACM SIGKDD Conference*, pages 528–543, 2002.

39. T. Dwyer, Y. Koren, and K. Marriott. IPSEP-COLA: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):821–828, 2006.

40. T. Dwyer, N. H. Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2596–2605, 2013.

41. D. Ellsworth, L.-J. Chiang, and H.-W. Shen. Accelerating time-varying hardware volume rendering using TSP trees and color-based error metrics. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 119–128, 2000.

42. B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

43. G. Frobenius. *Über Matrizen Aus Nicht Negativen Elementen*. Walter De Gruyter Incorporated, 1912.

44. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

45. P. Gambette and J. Véronis. Visualising a text with a tree cloud. In *Proceedings of International Federation of Classification Societies Conference*, pages 561–569, 2010.

46. Q. Gan, M. Zhu, M. Li, T. Liang, Y. Cao, and B. Zhou. Document visualization: An overview of current research. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(1):19–36, 2014.

47. J. Gao, C. Wang, L. Li, and H.-W. Shen. A parallel multiresolution volume rendering algorithm for large data visualization. *Parallel Computing (Special Issue on Parallel Graphics and Visualization)*, 31(2):185–204, 2005.

48. J. H. Goldberg and J. I. Helfman. Scanpath clustering and aggregation. In *Proceedings of Symposium on Eye Tracking Research and Applications*, pages 227–234, 2010.

49. W. H. Gomaa and A. A. Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.

50. Y. Gu and C. Wang. A study of hierarchical correlation clustering for scientific volume data. In *Proceedings of International Symposium on Visual Computing*, pages 437–446, 2010.

51. Y. Gu and C. Wang. TransGraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2015–2024, 2011.

52. Y. Gu and C. Wang. iTree: Exploring time-varying data using indexable tree. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 137–144, 2013.

53. Y. Gu, C. Wang, J. Ma, R. J. Nemiroff, and D. L. Kao. iGraph: A graph-based technique for visual analytics of image and text collections. In *Proceedings of IS&T / SPIE Conference on Visualization and Data Analysis*, 2015.

54. Y. Gu, C. Wang, J. Ma, R. J. Nemiroff, D. L. Kao, and D. Parra. Visualization and recommendation of large image and text collections toward effective sensemaking. *Information Visualization*, 2016. Accepted.

55. Y. Gu, C. Wang, T. Peterka, R. Jacob, and S. H. Kim. Mining graphs for understanding time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):965–974, 2016.

56. S. Havre, E. Hetzler, P. Whitney, and L. Nowell. ThemeRiver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, 2002.

57. J. Heer, J. Mackinlay, C. Stolte, and M. Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, 2008.

58. K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, and J. Van de Weijer. *Eye Tracking: A Comprehensive Guide to Methods and Measures*. Oxford University Press, 2011.

59. D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

60. R. Huang and K.-L. Ma. RGVis: Region growing based techniques for volume visualization. In *Proceedings of Pacific Conference on Computer Graphics and Applications*, pages 355–363, 2003.

61. R. Huang, K.-L. Ma, P. McCormick, and W. Ward. Visualizing industrial CT volume data for nondestructive testing applications. In *Proceedings of IEEE Visualization Conference*, pages 547–554, 2003.

62. M. J. Huiskes and M. S. Lew. The MIR Flickr retrieval evaluation. In *Proceedings of ACM International Conference on Multimedia Information Retrieval*, pages 39–43, 2008.

63. M. Imoto and T. Itoh. A 3D visualization technique for large scale time-varying data. In *Proceedings of International Conference on Information Visualisation*, pages 17–22, 2010.

64. C. Y. Ip, A. Varshney, and J. JaJa. Hierarchical exploration of volumes using multilevel segmentation of the intensity-gradient histograms. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2355–2363, 2012.

65. H. Jänicke and G. Scheuermann. Steady visualization of the dynamics in fluids using epsilon-machines. *Computers & Graphics*, 33(5):597–606, 2009.

66. H. Jänicke and G. Scheuermann. Visual analysis of flow features using information theory. *IEEE Computer Graphics and Applications*, 30(1):40–49, 2010.

67. H. Jänicke, M. Böttinger, and G. Scheuermann. Brushing of attribute clouds for the visualization of multivariate data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1459–1466, 2008.

68. T. J. Jankun-Kelly and K.-L. Ma. A spreadsheet interface for visualization exploration. In *Proceedings of IEEE Visualization Conference*, pages 69–76, 2000.

69. T. J. Jankun-Kelly and K.-L. Ma. MoireGraphs: Radial focus+context visualization and interaction for graphs with visual nodes. In *Proceedings of IEEE Symposium on Information Visualization*, pages 59–66, 2003.

70. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

71. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of ACM Symposium on Computational Geometry*, pages 10–18, 2002.

72. L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18 (1):39–43, 1953.

73. D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. Visual analytics: Scope and challenges. In *Visual Data Mining*, pages 76–90. Springer-Verlag Berlin Heidelberg, 2008.

74. K. Koh, B. Lee, B. Kim, and J. Seo. ManiWordle: Providing flexible control over wordle. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1190–1197, 2010.

75. K. Kucher and A. Kerren. Text visualization techniques: Taxonomy, visual survey, and community insights. In *Proceedings of IEEE Pacific Visualization Symposium (Visualization Notes)*, pages 117–121, 2015.

76. K. Kurzhals and D. Weiskopf. AOI transition trees. In *Proceedings of Graphics Interface Conference*, pages 41–48, 2015.

77. K. Kurzhals and D. Weiskopf. Space-time visual analytics of eye-tracking data for dynamic stimuli. *IEEE Transactions on Visualization and Computer Graphics*, 19 (12):2129–2138, 2013.

78. K. Kurzhals, F. Heimerl, and D. Weiskopf. ISeeCube: Visual analysis of gaze data for video. In *Proceedings of Symposium on Eye Tracking Research and Applications*, pages 43–50, 2014.

79. K. Kurzhals, M. Hlawatsch, F. Heimerl, M. Burch, T. Ertl, and D. Weiskopf. Gaze stripes: Image-based visualization of eye tracking data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1005–1014, 2016.

80. J. Lamping and R. Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 7(1): 33–55, 1996.

81. B. Lee, N. H. Riche, A. K. Karlson, and S. Carpendale. SparkClouds: Visualizing trends in tag clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1182–1189, 2010.

82. D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of International Conference on Information and Knowledge Management*, pages 556–559, 2003.

83. J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 2–11, 2003.

84. J. Lin, E. J. Keogh, S. Lonardi, J. P. Lankford, and D. M. Nystrom. Viztree: A tool for visually mining and monitoring massive time series databases. In *Proceedings of International Conference on Very Large Data Bases*, pages 1269–1272, 2004.

85. G. Linden, B. Smith, and J. York. Amazon.com Recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

86. L. Linsen, T. V. Long, P. Rosenthal, and S. Rosswog. Surface extraction from multi-field particle volume data using multi-dimensional cluster visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1483–1490, 2008.

87. A. Lu and H.-W. Shen. Interactive storyboard for overall time-varying data visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 143–150, 2008.

88. J. Ma, C. Wang, and C.-K. Shene. FlowGraph: A compound hierarchical graph for flow field exploration. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 233–240, 2013.

89. J. Ma, C. Wang, C.-K. Shene, and J. Jiang. A graph-based interface for visual analytics of 3D streamlines and pathlines. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1127–1140, 2013.

90. K.-L. Ma. Image graphs - a novel approach to visual data exploration. In *Proceedings of IEEE Visualization Conference*, pages 81–88, 1999.

91. W. Meulemans, N. H. Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1846–1858, 2013.

92. C. Muelder and K.-L. Ma. Interactive feature extraction and tracking by utilizing region coherency. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 17–24, 2009.

93. R. J. Nemiroff and J. T. Bonnell. Astronomy picture of the day: `http://antwrp.gsfc.nasa.gov/apod/astropix.html`. In *Bulletin of the American Astronomical Society*, page 1291, 1995.

94. F. J. Newbery. Edge concentration: A method for clustering directed graphs. *ACM SIGSOFT Software Engineering Notes*, 14(7):76–85, 1989.

95. B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011.

96. S. Oeltze, W. Freiler, R. Hillert, H. Doleisch, B. Preim, and W. Schubert. Interactive, graph-based visual analysis of high-dimensional, multi-parameter fluorescence microscopy data in toponomics. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1882–1891, 2011.

97. O. Perron. Zur theorie der matrices. *Mathematische Annalen*, 64(2):248–263, 1907.

98. J. C. Platt, M. Czerwinski, and B. A. Field. PhotoTOC: Automatic clustering for browsing personal photographs. In *Proceedings of IEEE Pacific Rim Conference on Multimedia*, pages 6–10, 2003.

99. H. Qu, W.-Y. Chan, A. Xu, K.-L. Chung, K.-H. Lau, and P. Guo. Visual analysis of the air pollution problem in Hong Kong. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1408–1415, 2007.

100. K. Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3):372–422, 1998.

101. F. Reinders, F. H. Post, and H. J. W. Spoelder. Visualization of time-dependent data with feature tracking and event detection. *The Visual Computer*, 17(1):55–71, 2001.

102. A. Robles-Kelly and E. R. Hancock. String edit distance, random walks and graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):315–327, 2004.

103. L. Royer, M. Reimann, B. Andreopoulos, and M. Schroeder. Unraveling protein networks with power graph analysis. *PLoS Computational Biology*, 4(7):e1000108, 2008.

104. R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tacking their evolution. *IEEE Computer*, 27(7):20–27, 1994.

105. A. Santella and D. DeCarlo. Robust clustering of eye movement recordings for quantification of visual interest. In *Proceedings of Symposium on Eye Tracking Research and Applications*, pages 27–34, 2004.

106. N. Sauber, H. Theisel, and H.-P. Seidel. Multifield-Graphs: An approach to visualizing correlations in multifield scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):917–924, 2006.

107. H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *Proceedings of IEEE Visualization Conference*, pages 371–377, 1999.

108. J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Proceedings of ACM Workshop on Applied Computational Geometry*, pages 203–222, 1996.

109. J. Shieh and E. Keogh. iSAX: Indexing and mining terabyte sized time series. In *Proceedings of ACM SIGKDD Conference*, pages 623–631, 2008.

110. D. Silver and X. Wang. Tracking and visualizing turbulent 3D features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, 1997.

111. D. Silver and X. Wang. Tracking scalar features in unstructured data sets. In *Proceedings of IEEE Visualization Conference*, pages 79–86, 1998.

112. D. Silver and X. Wang. Visualizing evolving scalar phenomena. *Future Generation Computer Systems*, 15(1):99–108, 1999.

113. J. Smallwood and J. W. Schooler. The restless mind. *Psychological Bulletin*, 132(6): 946–958, 2006.

114. B.-S. Sohn and C. L. Bajaj. Time-varying contour topology. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):14–25, 2006.

115. H. Strobelt, D. Oelke, C. Rohrdantz, A. Stoffel, D. A. Keim, and O. Deussen. Document Cards: A top trumps visualization for documents. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1145–1152, 2009.

116. J. Sukharev, C. Wang, K.-L. Ma, and A. T. Wittenberg. Correlation study of time-varying multivariate climate data sets. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 161–168, 2009.

117. E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

118. R. Torres, C. Silva, C. Medeiros, and H. Rocha. Visual structures for image browsing. In *Proceedings of International Conference on Information and Knowledge Management*, pages 49–55, 2003.

119. M. Tory, M. S. Atkins, A. E. Kirkpatrick, M. Nicolaou, and G.-Z. Yang. Eyegaze analysis of displays with combined 2D and 3D views. In *Proceedings of IEEE Symposium on Information Visualization*, pages 519–526, 2005.

120. K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of NAACL Conference on Human Language Technology*, pages 252–259, 2003.

121. H. Y. Tsang, M. Tory, and C. Swindells. eSeeTrack - visualizing sequential fixation patterns. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):953–962, 2010.

122. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

123. A. Šilić and B. D. Baısić. Visualization of text streams: A survey. In *Proceedings of International Conference on Knowledge-Based and Intelligent Information and Engineering Systems: Part II*, pages 31–43, 2010.

124. F. van Ham, M. Wattenberg, and F. B. Viégas. Mapping text with phrase nets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1169–1176, 2009.

125. F. B. Viégas, M. Wattenberg, and J. Feinberg. Participatory visualization with Wordle. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1137–1144, 2009.

126. A. Voßkühler, V. Nordmeier, L. Kuchinke, and A. M. Jacobs. OGAMA (open gaze and mouse analyzer): Open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior Research Methods*, 40(4):1150–1162, 2008.

127. C. Wang. A survey of graph-based representations and techniques for scientific visualization. In *Eurographics Conference on Visualization - State of The Art Reports*, pages 41–60, 2015.

128. C. Wang and K.-L. Ma. A statistical approach to volume data quality assessment. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):590–602, 2008.

129. C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy (Special Issue on Advances in Information Theory)*, 13(1):254–273, 2011.

130. C. Wang and H.-W. Shen. A framework for rendering large time-varying data using wavelet-based time-space partitioning (WTSP) tree. Technical Report OSU-CISRC-1/04-TR05, The Ohio State University, 2004.

131. C. Wang and H.-W. Shen. Hierarchical navigation interface: Leveraging multiple coordinated views for level-of-detail multiresolution volume rendering of large scientific data sets. In *Proceedings of International Conference on Information Visualisation*, pages 259–267, 2005.

132. C. Wang and H.-W. Shen. LOD map - a visual interface for navigating multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1029–1036, 2006.

133. C. Wang and J. Tao. Graphs in scientific visualization: A survey. *Computer Graphics Forum*, 2016. Accepted.

134. C. Wang, J. Gao, and H.-W. Shen. Parallel multiresolution volume rendering of large data sets with error-guided load balancing. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 23–30, 2004.

135. C. Wang, J. Gao, L. Li, and H.-W. Shen. A multiresolution volume rendering framework for large-scale time-varying data visualization. In *Proceedings of Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 11–19, 2005.

136. C. Wang, H. Yu, and K.-L. Ma. Application-driven compression for visualizing large-scale time-varying data. *IEEE Computer Graphics and Applications*, 30(1):59–69, 2010.

137. C. Wang, H. Yu, R. W. Grout, K.-L. Ma, and J. H. Chen. Analyzing information transfer in time-varying multivariate data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 99–106, 2011.

138. C. Wang, J. P. Reese, H. Zhang, J. Tao, and R. J. Nemiroff. iMap: A stable layout for navigating large image collections with embedded search. In *Proceedings of IS&T / SPIE Conference on Visualization and Data Analysis*, 2013.

139. C. Wang, J. P. Reese, H. Zhang, J. Tao, Y. Gu, J. Ma, and R. J. Nemiroff. Similarity-based visualization of large image collections. *Information Visualization*, 14(3):183–203, 2015.

140. Y. Wang, H. Yu, and K.-L. Ma. Scalable parallel feature extraction and tracking for large time-varying 3D volume data. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 17–24, 2013.

141. F. Wanner, A. Stoffel, D. Jäckle, B. C. Kwon, A. Weiler, and D. A. Keim. State-of-the-art report of visual analysis for event detection in text data streams. In *Eurographics Conference on Visualization - State of The Art Reports*, 2014.

142. M. Wattenberg and F. B. Viegas. The word tree, an interactive visual concordance. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1221–1228, 2008.

143. F. Wei, S. Liu, Y. Song, S. Pan, M. X. Zhou, W. Qian, L. Shi, L. Tan, and Q. Zhang. TIARA: A visual exploratory text analytic system. In *Proceedings of ACM SIGKDD Conference*, pages 153–162, 2010.

144. J. Wilhelms and A. V. Gelder. Multi-dimensional trees for controlled volume rendering and compression. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 27–34, 1994.

145. J. Woodring and H.-W. Shen. Semi-automatic time-series transfer functions via temporal clustering and sequencing. *Computer Graphics Forum*, 28(3):791–798, 2009.

146. J. Xie and B. K. Szymanski. Towards linear time overlapping community detection in social networks. In *Proceedings of Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 25–36, 2012.

147. L. Xu and H.-W. Shen. Flow web: A graph based user interface for 3D flow field exploration. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, 2010.

148. J. Yang, J. Fan, D. Hubball, Y. Gao, H. Luo, W. Ribarsky, and M. Ward. Semantic image browser: Bridging information visualization with automated intelligent image analysis. In *Proceedings of IEEE Symposium on Visual Analytics Science and Technology*, pages 191–198, 2006.

149. H. Zhang, J. Tao, F. Ruan, and C. Wang. A study of animated transition in similarity-based tiled image layout. *Tsinghua Science and Technology*, 18(2):157–170, 2013.