Mining Graphs for Understanding Time-Varying Volumetric Data

Yi Gu, Chaoli Wang, *Senior Member, IEEE*, Tom Peterka, *Member, IEEE*, Robert Jacob, and Seung Hyun Kim

Abstract—A notable recent trend in time-varying volumetric data analysis and visualization is to extract data relationships and represent them in a low-dimensional abstract graph view for visual understanding and making connections to the underlying data. Nevertheless, the ever-growing size and complexity of data demands novel techniques that go beyond standard brushing and linking to allow significant reduction of cognition overhead and interaction cost. In this paper, we present a mining approach that automatically extracts meaningful features from a graph-based representation for exploring 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 investigate the most important transition relationships for time-varying data and evaluate our solution with several time-varying data sets of different sizes and characteristics. 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. We also collect expert feedback to confirm the usefulness of our approach.

Index Terms—Time-varying data visualization, graph simplification, community detection, visual recommendation.

1 Introduction

In scientific visualization, a notable recent trend in visualizing scalar and vector field data is to design effective user interfaces that integrate techniques from computational transformation and data analysis [25]. Visual representations such as parallel coordinates, treemaps, and graphs have been utilized to assist scientific visualization tasks such as transfer function specification, level-of-detail selection, and flow relationship exploration [2, 26, 15]. Instead of being confined to the original spatiotemporal domain, these solutions extract data and their relationships over space and time, display these relationships in a low-dimensional space, and enable users to perform queries and make connection to the original data.

The idea of transforming the data and their relationships into an abstract view for exploring complex relationships and improving data understanding has been accepted as a viable means to analyze and visualize scientific data. Nevertheless, most of these representations lack enough guidance for user exploration and navigation. In many cases, 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 approach may work for small data with simple relationships, but becomes increasingly inefficient for larger data with greater complexity. Therefore, solutions that help users sift through the data and their relationships for cost-effective understanding are highly desirable.

In this paper, we specifically focus on time-varying volume data visualization and investigate transition relationships among data items over time. We present a *mining* approach that automatically extracts features from a graph-based representation for understanding time-varying data. Beyond straightforward graph properties, users are given further guidance available 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 based on user selected items, enabling users to spend more time on the actual analysis instead of painstaking interaction. Furthermore, visual recommendation based on node groups actually recommends the groups according to their *structural relations* and suggests similar groups regardless of their volume values, spatial regions, and temporal ranges. When the data set is large and the relationships are complex, navigating and exploring the resulting graph is a daunting task. We show that the suite of *high-level* functions introduced provides the convenience and capabilities for graph exploration which are difficult or impossible to achieve through standard interaction techniques. Our solution thus represents a step forward in applying graph-based techniques for scientific data analysis and visualization.

- *Y. Gu and C. Wang are with the Department Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556. E-mail: {ygu5, chaoli.wang}@nd.edu.*
- *T. Peterka and R. Jocob are with the Division of Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL 60439. E-mail: {tpeterka, jacob}@mcs.anl.gov.*
- *S. H. Kim is with the Department of Mechanical and Aerospace Engineering, The Ohio State University, Columbus, OH 43210. E-mail: kim.5061@osu.edu.*

Manuscript received 31 Mar. 2015; accepted 1 Aug. 2015; date of publication xx Aug. 2015; date of current version 25 Oct. 2015. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

2 Related Work

Popular visual representations such as scatterplots, parallel coordinates, and treemaps have been extensively applied to scientific visualization. Unlike straightforward applications of these visual representations, leveraging the more generic and powerful visual graph representations requires a fully integrated pipeline of data transformation, representation, and visual mapping. The pioneering work of image graphs by Ma [16] encodes the visualization process, including parameters and results, into a graph representation.

For time-varying multivariate data visualization, Sauber et al. [22] introduced the Multifield-Graphs, a user interface that shows the overview of all possible correlation fields derived from several scalar fields for multivariate correlation exploration. Jänicke et al. [14] presented the attribute cloud, a two-dimensional projection of high-dimensional data that uses a minimum spanning tree to reduce edge crossing and layout the data graph. Their interface enables intuitive data brushing in 2D and connection with the underlying 3D data. Gu and Wang designed two representations, TransGraph [11] and iTree [12], for time-varying data visualization. TransGraph [11] encodes hierarchical transition relationships for a time-varying data set to guide relationship exploration and tracking. iTree [12] integrates efficient data compacting, indexing, and classification into a single framework. A hyperbolic layout algorithm is employed to draw the iTree with a large number of nodes and focus+context visualization is provided for interaction. Researchers also designed other graphs for exploring, analyzing and tracking temporal evolution of features in large-scale simulation data sets. Examples include dynamic tracking graphs [27], attributed relational graphs [17], and Petri Nets [19].

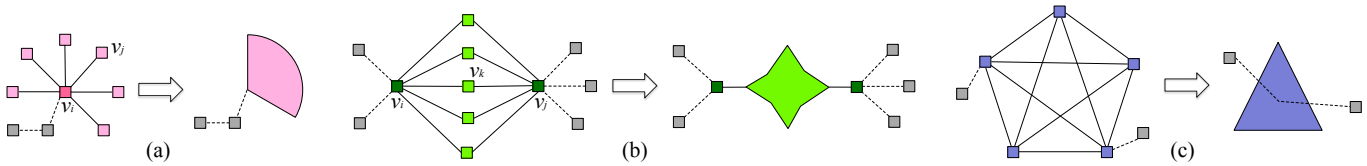


Fig. 1. Graph features for simplification. (a) to (c) show examples for fan, connector, and clique, respectively. We use the included angle of fan, the size of star, and the size of triangle to indicate the number of nodes simplified for the three graph features, respectively. Regular (i.e., unsimplified) nodes are drawn as squares. In (c), all other nodes connecting to the clique are connected to the center of the triangle.

Our work differs from all previous work in that we present a *mining* approach for viewing, exploring and navigating a graph representation extracted from a scientific data set. To handle a large number of nodes produced from a large data set, TransGraph builds a hierarchy and allows users to explore the details on demand. Users interact with TransGraph through standard interaction techniques, which is not always convenient and efficient. The techniques we employ include graph simplification, community detection, and visual recommendation. Graph simplification not only reduces the number of nodes displayed on the screen but also highlights important graph structures. To further reduce user workload, community detection groups similar nodes into communities. Node and community recommendation functions allow users to fetch similar volumetric regions from the original data via graph, and suggest similar nodes or graph structures to help the exploration. Users are given the flexibility to apply these techniques in sequence or selectively to the data.

3 Transition Graph

Our work is based on the transition relationships in time-varying volumetric data sets as defined in TransGraph [11]. 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 describe the construction of a transition graph.

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 [9] 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 [11].

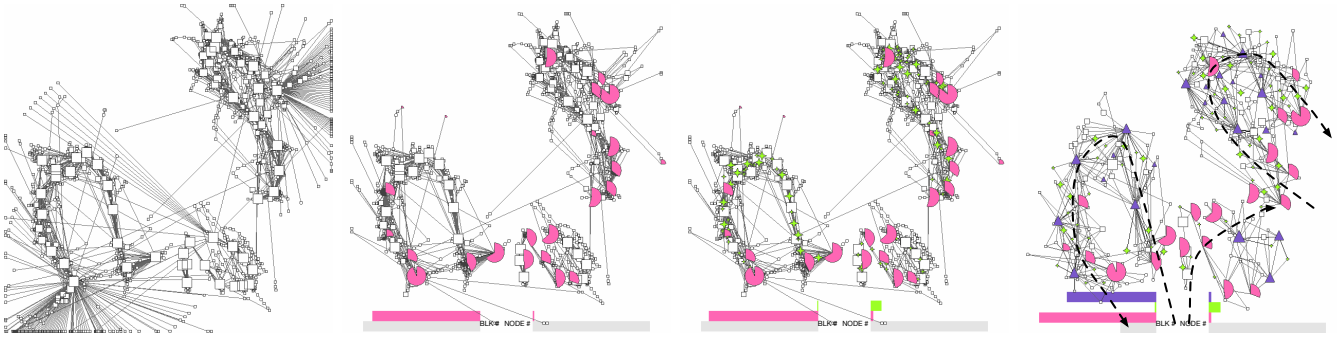
4 Graph Simplification

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 [18, 5, 21, 24, 8] identify the relations 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 [3] detects topological features such as trees, connected components and biconnected components, and motif simplification [7] 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 [7], we select three graph features for simplification because they represent meaningful transition relations and do not impose restrictions to the order of simplification. Figure 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 1 (a). The undirected edge between nodes v_i and v_j has three kinds of relation: $v_i \rightarrow v_j$, $v_j \rightarrow v_i$, and $v_i \leftrightarrow v_j$. 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 1 (b). Assume that two ending nodes are v_i and v_j and an intermediate node is v_k, 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 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 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



(a) (b) (c) (d)

Fig. 2. (a) to (d) are the original transition graph, the graph after detecting fans, connectors, and cliques, respectively. In (d), 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.

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. Figure 2 shows an example of the 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. In Table 1, we report the number of nodes originally created, after simplification, and finally displayed. 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.

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. [23]. This algorithm has a time complexity of $3^{|V|/3}$, where $|V|$ is the number of nodes 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 2 in the Appendix detects all the cliques in the graph. 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.

5 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 [28], an extended 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.

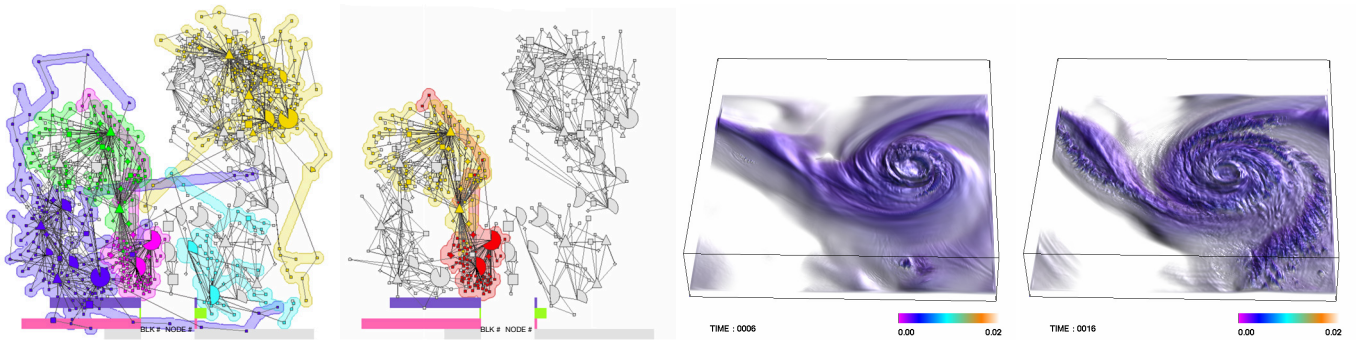
SLPA is outlined in Algorithm 3 in the Appendix. In SLPA, a *label* stands for a community identification, and each node has a buffer to store the labels received from neighboring nodes (i.e., having edges pointing to the node). SLPA detects the communities in an iterative manner. In each step, a node 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

$$L_i = \arg\max_{j \in N_i} p_{j \to i} L_j, \tag{1}$$

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 3 (a) shows an example of communities detected. We implement and draw Bubble Sets [4] to distinguish different communities. Bubble Sets use continuous, often concave, isocontours to delineate group memberships, maintaining the spatial arrangement of the primary data relation 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.

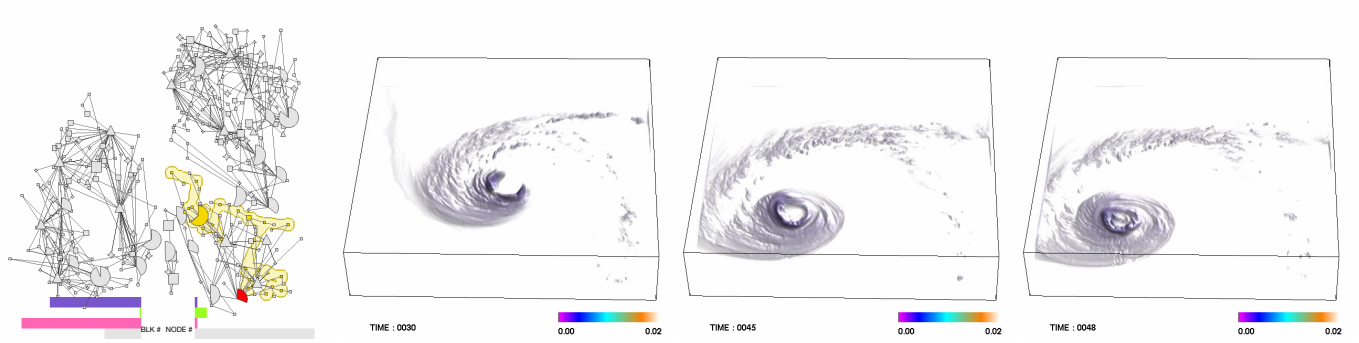
6 Visual Recommendation

While the focus of this paper 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 paper [11]. Here we focus on the newly introduced node and community recommendation functions.



(a) (b) (c) (d)

Fig. 3. (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. (c) and (d) are the rendering of the corresponding blocks of the chosen and recommended communities at two different time steps, respectively.



(a) (b) (c) (d)

Fig. 4. (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. (c) and (d) are the rendering of the corresponding blocks of the recommended nodes at two later time steps.

Node Recommendation. To further guide the exploration of graph, we recommend similar nodes when a node or multiple nodes are selected. Heer et al. [13] 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 on their similarities. We leverage the SimRank algorithm [10] 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

$$s(v_a,v_b) = \frac{\sum_{i=1}^{|I(v_a)|}\sum_{j=1}^{|I(v_b)|} s(I_i(v_a),I_j(v_b))}{|I(v_a)||I(v_b)|}, \quad (2)$$

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 according to Equation 2. SimRank has the time complexity of $k\bar{d}|V|^2$, where k is the number of iterations and $\bar{d}$ 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 4 in the Appendix, our modified SimRank algorithm takes transition probabilities into account. We modify Equation 2 to the following

$$s(v_a,v_b) = \frac{\sum_{i=1}^{|I(v_a)|}\sum_{j=1}^{|I(v_b)|} s(I_i(v_a),I_j(v_b)) \times p_{i\to v_a} \times p_{j\to v_b}}{|I(v_a)||I(v_b)|}, \quad (3)$$

where $p_{i\to v_a}$ is the transition probability from node $I_i(v_a)$ to node v_a. We also extend Equation 3 to outgoing edges

$$s(v_a,v_b) = \frac{\sum_{i=1}^{|O(v_a)|}\sum_{j=1}^{|O(v_b)|} s(O_i(v_a),O_j(v_b)) \times p'_{i\to v_a} \times p'_{j\to v_b}}{|O(v_a)||O(v_b)|}, \quad (4)$$

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 3, Equation 4, or a combination of both as the final similarity measure. Figure 4 shows 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.

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 [20]. 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 5 in the Appendix 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 nodes based on their importance and still preserve edge relations, 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. How-

data set	dimension	block size	state clustering time / w / δ	layout create / adjust	# nodes original / simplified / displayed	community / node recommendation
DCMIP cam-fv	$360\times180\times30\times31$	$20\times10\times5$	99.62 / 10 / 0.6	62.07 / 1.43	1909 / 1037 / 247	7.72 / 0.99
DCMIP cam-se	$512\times256\times30\times31$	$32\times16\times5$	123.32 / 10 / 0.5	62.25 / 1.5	1959 / 1063 / 223	7.03 / 0.99
DCMIP fim	$360\times180\times30\times31$	$20\times10\times5$	137.83 / 10 / 0.45	61.07 / 0.88	1868 / 786 / 212	1.18 / 0.54
earthquake	$256\times256\times96\times360$	$16\times16\times16$	43.71 / 10 / 0.4	86.74 / 0.94	2567 / 806 / 356	0.31 / 0.4
		$16\times16\times8$	155.64 / 10 / 0.5	241.37 / 2.52	3783 / 1353 / 406	2.46 / 1.31
		$32\times32\times16$	19.49 / 10 / 0.25	32.00 / 0.89	1390 / 780 / 558	0.48 / 0.37
hurricane	$500\times500\times100\times48$	$20\times20\times20$	187.67 / 12 / 0.25	69.88 / 1.43	2051 / 1143 / 313	3.89 / 0.43
ionization	$600\times248\times248\times200$	$30\times31\times31$	98.28 / 10 / 0.3	87.56 / 2.06	2320 / 1151 / 714	0.46 / 0.61
NOAA climate	$360\times66\times27\times120$	$15\times11\times9$	9.12 / 12 / 0.25	62.67 / 2.56	1926 / 1373 / 685	0.57 / 1.26
		$15\times6\times9$	17.16 / 12 / 0.3	126.57 / 1.90	2800 / 1947 / 909	2.44 / 3.1
		$9\times11\times9$	11.06 / 12 / 0.35	57.24 / 2.43	1849 / 1308 / 632	0.89 / 1.43

Table 1. The data sets, parameters used, and timing results. All timing results are in seconds.

ever, 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 relations.

After ordering the nodes, Algorithm 7 in the Appendix 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 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 [6] 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

$$d=\beta_{(a,b)}\times\beta_{(c,d)}\times R_1(a,c)\times R_2(b,d), \tag{5}$$

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

$$\beta_{(a,b)}=\frac{\max\{D_1'(a),D_2'(b)\}-\min\{D_1'(a),D_2'(b)\}}{\max\{D_1'(a),D_2'(b)\}}, \tag{6}$$

where $\mathbf{D}'$ is the degree matrix calculated in Algorithm 6 in the Appendix. $\beta_{(c,d)}$ can be calculated similarly. R_1 in Equation 5 can be calculated as

$$R_1(a,c)=\begin{cases} P_1'(a,c), & \text{if } A_1(a,c)=1 \\ \frac{2\times|V_1|\times|V_2|}{|V_1|+|V_2|}, & \text{otherwise} \end{cases}, \tag{7}$$

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 3 (b) shows an example of community recommendation. The user chooses a community with nodes highlighted in red. The most similar community recommended is shown with nodes highlight in yellow. The correspondence of volume highlighting results in Figure 3 (c) and (d) shows the effectiveness of community recommendation.

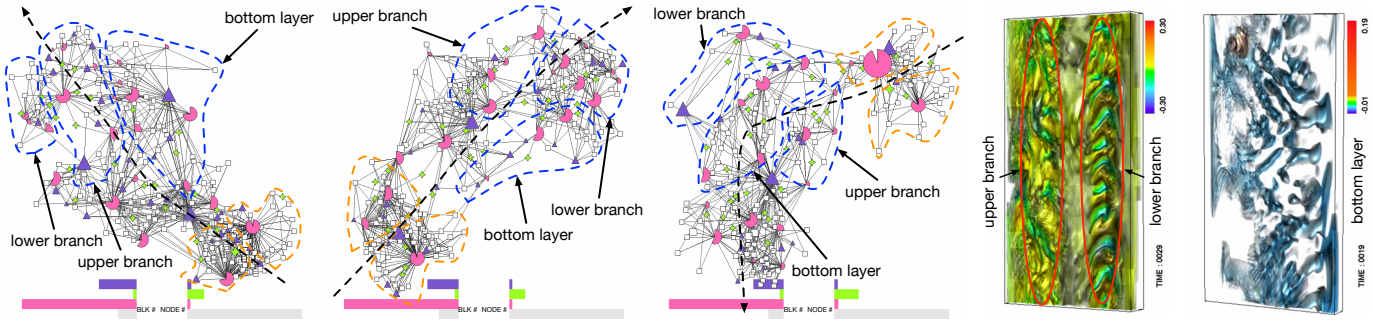
7 Case Study Results

Data Sets, Parameters, and Timing. We evaluated our approach with several time-varying data sets listed in Table 1. The timing results were collected using a workstation with an Intel Core i7-960 3.5GHz CPU and 24GB memory. We used MPI to 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.

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.

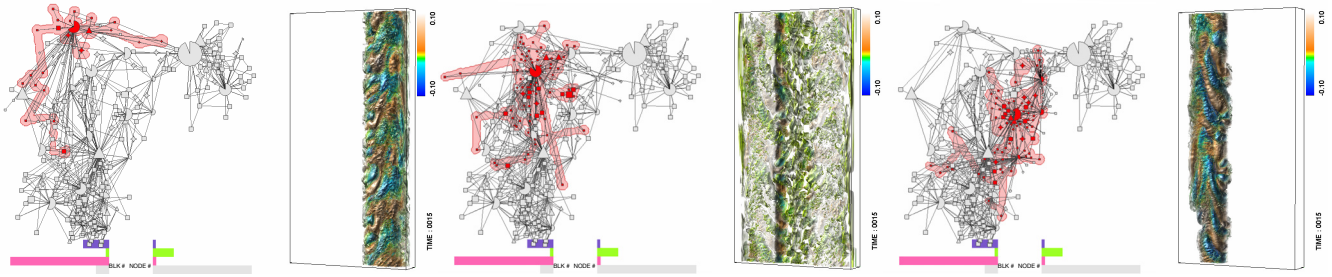
As expected, the graphs derived from these three data sets are similar, although not entirely the same, as shown in Figure 5 (a), (b) and (c). We can see that fans and cliques occupy the central locations in the graphs and they are surrounded by many connectors and regular nodes. Furthermore, at the 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 5 (d) and (e) 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 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 5 (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 5 (a) begin to form groups in the later time steps. In fact, the lower branch highlighted in (d) 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 5 (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 5 (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



(a) DCMIP cam-fv (b) DCMIP cam-se (c) DCMIP fim (d) all fans (e) all cliques

Fig. 5. (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. (d) and (e) are the rendering of the corresponding blocks of all fans and cliques from DCMIP cam-fv and DCMIP cam-se data sets.



(a) (b) (c) (d) (e) (f)

Fig. 6. Three communities, shown in (a), (c), and (e), detected from the DCMIP fim data set and the rendering of their corresponding blocks in (b), (d), and (f), respectively. (b), (d), and (f) correspond to the lower branch, the surrounding, and the upper branch, respectively.

diverge which leads 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 5 (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 Figure 6. 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 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 Figure 6. In addition, communities help us explain unusual events. In Figure 6 (a), (c), and (e), the communities in the graph correspond to the lower branch, the surrounding, and the upper branch of the volume data, respectively, as shown in Figure 6 (b), (d), and (f). 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 7. The same conclusion can be drawn for the DCMIP cam-fv data set.

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 8 (a) where the empty regions correspond to the continents. In Figure 8 (b), we can see clear separation among three node groups which are highlighted with the dashed boundaries. These three node groups

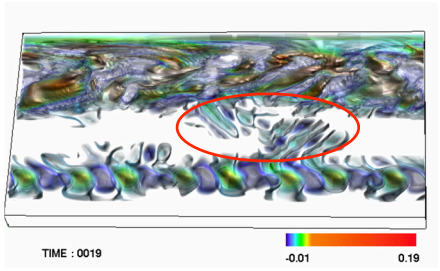
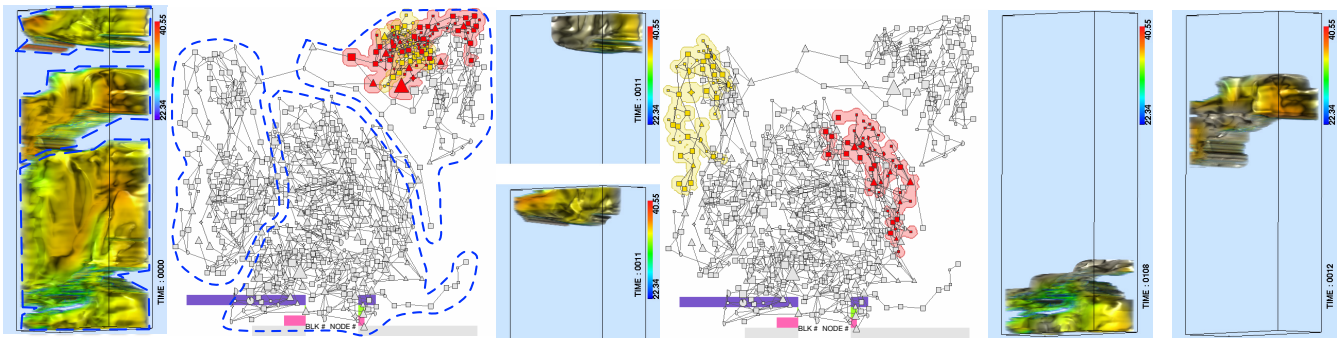


Fig. 7. The rendering of blocks corresponding to the two branches of the DCMIP cam-se data set, revealing their direct interaction.

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.

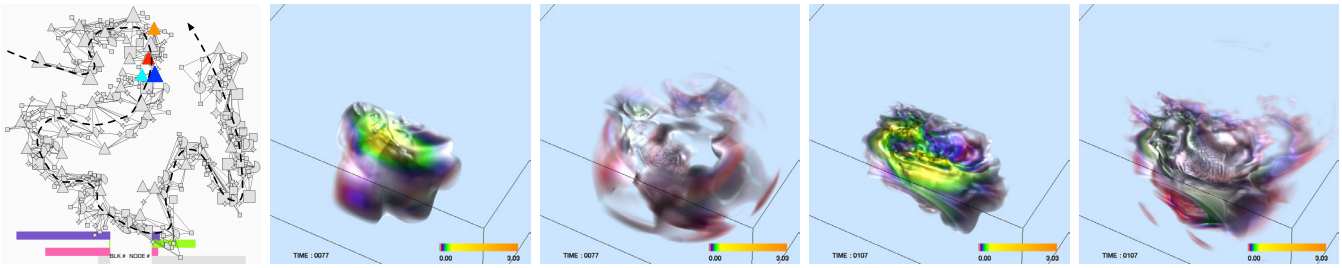
In Figure 8 (b), we select a community (shown in red) and the community shown in yellow is recommended. Their corresponding blocks are highlighted in the top and bottom images in Figure 8 (c), respectively. We can see that the recommended community corresponds to the spatial neighbors that are similar to the selected one at the same time step. In Figure 8 (d), the selected and recommended communities are in different time steps. Their corresponding blocks are highlighted in Figure 8 (e) and (f), 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.

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 9 (a) 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 central locations in the graph. The four cliques highlighted in Figure 9 (a) correspond



(a) (b) (c) (d) (e) (f)

Fig. 8. (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. (d) to (f): recommending the community (yellow) that is not the spatial neighbor of the selected community (red) at a different time step.



(a) (b) (c) (d) (e)

Fig. 9. (a) four cliques are selected in the transition graph of the earthquake data set. The evolution of time is marked with the dashed line. (b) to (e) are the rendering of the corresponding data blocks: red clique for (b), yellow clique for (c), green clique for (d), and blue clique for (e).

to different volumetric regions in Figure 9 (b) to (e). Specifically, (b) and (d) correspond to the inner layers of the earthquake's center, and (c) and (e) 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 10 (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 10 (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.

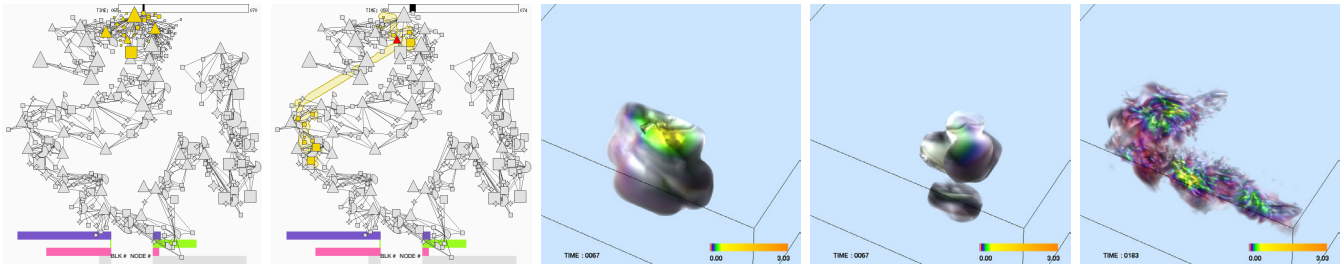
Ionization. The ionization data set was produced from a 3D radiation hydrodynamical simulation of ionization front instabilities for studying a variety of phenomena in interstellar medium such as the formation of stars. A small front appears at the beginning time steps. The front moves and grows along the x direction, followed by a cuboid with similar values. Figure 11 (j) 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 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 11 (j), 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 11 (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 11 (a) to (d), four communities are selected. Their corresponding volume regions are highlighted in Figure 11 (f) to (i). (f) to (h) correspond to three different layers of the front while (i) corresponds to the base.

8 Discussion

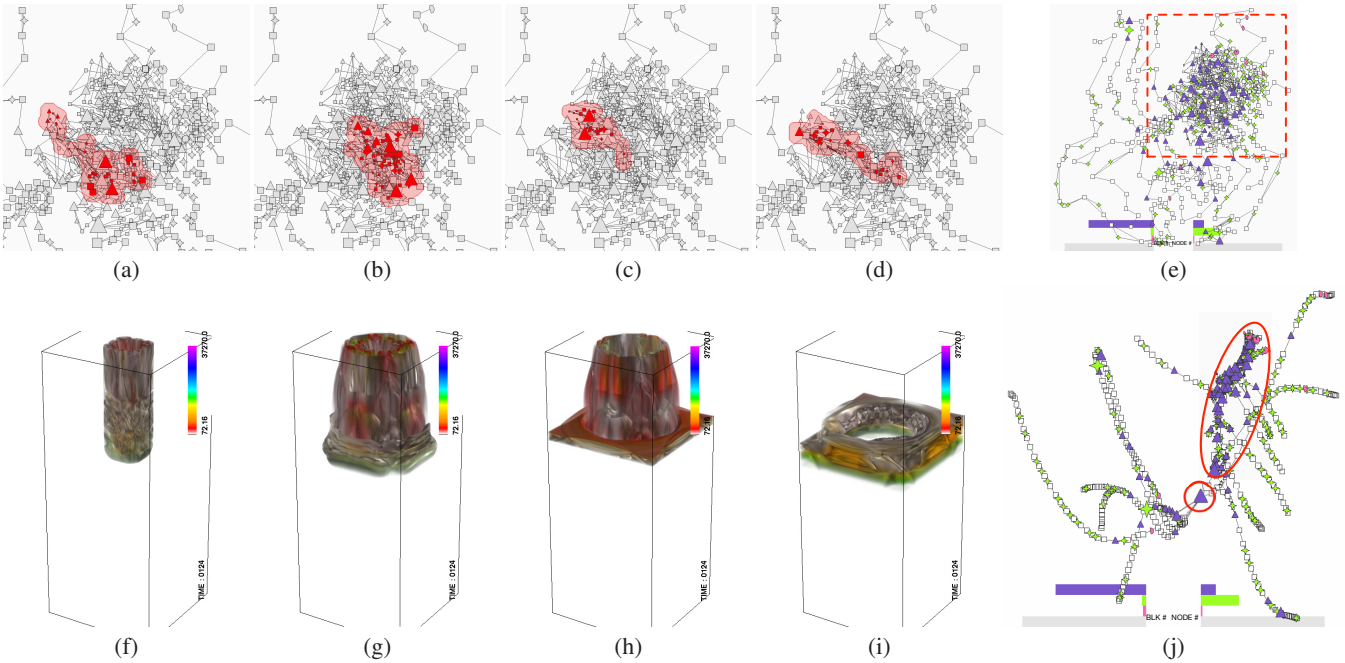
Parameter Selection. Figure 12 shows the graph layouts of two data sets under different parameter settings. Comparing Figure 12 (a), (b), and (c), we can see that there are many connectors in all three graphs. In addition, there are more cliques and less fans in the early time steps than in the later time steps. Note that the number of fans in Figure 12 (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 12 (d), (e), and (f), we can see clear separation of the nodes which correspond to the three regions in Figure 8 (a). Therefore, although different sets of 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.

Limitations. Our community detection and recommendation algorithms have the following limitations which we would like to improve. First, the community detection algorithm automatically suggests the number of communities. In some cases, this leads to a large number of communities with each having a small number of nodes. We will seek a different solution which allows users to specify the number of communities. Second, unlike community detection, we would like to



(a) (b) (c) (d) (e)

Fig. 10. (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. The clique corresponds to the rendering in (c). (d) shows the corresponding rendering of the recommended nodes at the same time step as (c). (e) shows the corresponding rendering of the recommended nodes at a later time step, essentially a tracking result for (d).



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

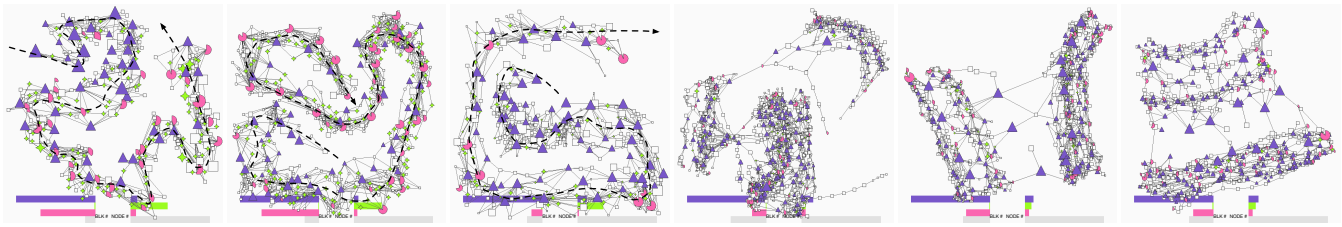
Fig. 11. (a) to (d) are four communities selected from the transition graph of the ionization data set. In order to better study the front area, the portion of the transition graph marked with a red rectangle in (e) is selected to zoom in for (a) and (d). (f) to (i) are their corresponding volume highlighting results. (j) the transition graph before layout adjustment reveals a large number of connectors lying on different branches.

allow users to select a subgraph of interest and apply *graph alignment* techniques to suggest similar subgraphs. This would give more flexibility to users in their selection and matching. Third, the community recommendation algorithm treats each community as an unweighted, undirected graph. As shown in Figure 13, given three communities with a similar number of nodes, our distance computation claims that (b) is more similar to (a) than (c). The algorithm [20] provides us a measure of the similarity between communities. However, the measure does not consider other information, such as node type, node importance, edge weight, and edge direction. We would like to seek a better solution that incorporates these graph attributes into similarity measure design. Finally, we will allow domain scientists to define general patterns and analysis procedures of interest based on their domain knowledge, enabling more customized analytics.

9 Feedback from Domain Experts

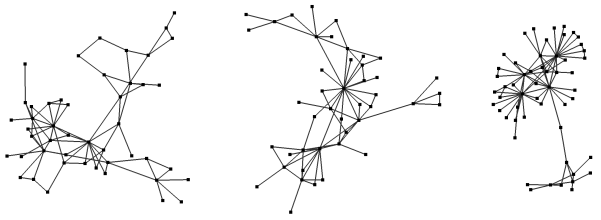
We also conducted expert evaluation with two domain experts, Drs. Robert Jacob and Seung Hyun Kim. Dr. Jacob's research focuses on ocean and coupled climate models and the computational, mathematical and theoretical issues involved in their construction. Dr. Kim's research focuses on modeling of multiscale and multiphysics problems in relation 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-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. Through 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.



(a) (b) (c) (d) (e) (f)

Fig. 12. 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. (d), (e), and (f) 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.



(a) (b) (c)

Fig. 13. Community comparison for the NOAA climate data set. (a) is the community selected and (b) and (c) are two communities recommended. The distance between (a) and (b) is 0.0058, and the distance between (a) and (c) is 0.0322.

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 relations 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* relation. 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.

10 Conclusions and Future Work

In scientific visualization, the goal for showing a graph view in conjunction with the original spatiotemporal data view is to provide both sufficient data reduction and enough visual guidance to reduce the time for users to analyze the data. Effective analytics of data at scale will require a change from simply generating graphs as another view of data to seeking immediate meanings from such an abstraction through graph mining. Our work includes functions for graph simplification, community detection, and node and community recommendation. The results and evaluation with different data sets demonstrate that such a graph 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.

We plan to extend our work to handle multivariate data sets. We can either fuse multiple variables into one type of node, or construct one type of node for each individual variable. In the latter case, we can visualize the relations between variables using compound graphs. We will also investigate time-evolving graphs derived from scientific data sets for identifying temporal hotspots, detecting anomaly, and aligning multiple graphs for finding common features and distinct patterns.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation through grants IIS-1456763 and IIS-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. Special thanks to Dr. Amanda Sgroi for her narration of the accompanying video and the anonymous reviewers for their insightful comments.

REFERENCES

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

[2] H. Akiba and K.-L. Ma. A tri-space visualization interface for analyzing time-varying multivariate volume data. In *Proceedings of Joint Eurographics - IEEE VGTC Symposium on Visualization*, pages 115–122, 2007.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

[10] J. Glen and J. Widom. SimRank: A measure of structural context similarity. In *Proceedings of ACM SIGKDD Conference*, pages 528–543, 2002.

[11] 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.

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

[13] J. Heer, M. Agrawala, and W. Willett. Generalized selection via interactive query relaxation. In *Proceedings of ACM SIGCHI Conference*, pages 959–968, 2008.

[14] 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.

[15] 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.

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

[17] W. McLendon, G. Bansal, P.-T. Bremer, J. Chen, H. Kolla, and J. Bennett. On the use of graph search techniques for the analysis of extreme-scale combustion simulation data. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, pages 57–63, 2012.

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

[19] S. Ozer, D. Silver, K. Bemis, and P. Martin. Activity detection in scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):377–390, 2014.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] W. Widanagamaachchi, C. Christensen, P.-T. Bremer, and V. Pascucci. Interactive exploration of large-scale time-varying data using dynamic tracking graphs. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, pages 9–17, 2012.

[28] J. Xie and B. K. Szymanski. Towards linear time overlapping community detection in social networks. In *Proceedings of Pacific Asia Knowledge Discovery and Data Mining Conference*, pages 25–36, 2012.

APPENDIX

1 CLIQUE DETECTION

Algorithm 2 detects all the cliques in the graph. Given a graph (subgraph), we always record the candidate nodes to expand as described in Algorithm 1. 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 non-adjacent nodes is used to avoid duplication. For each node q in Q, we find its adjacent nodes, treat them as the nodes in a subgraph, and repeat 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.

Algorithm 1 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
return Q
```

Algorithm 2 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'))
```

2 COMMUNITY DETECTION

The SLPA community detection algorithm is outlined in Algorithm 3. 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

$$L_i = \arg\max_{j \in N_i} p_{j \to i} L_j, \tag{1}$$

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.

3 NODE RECOMMENDATION

We utilize the SimRank algorithm to calculate the similarities between nodes for node recommendation. 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. However, in our graph, each edge carries a weight indicating the transition probability between two incident nodes, thus the original SimRank algorithm cannot be immediately applied. We therefore modify the similarity between v_a and v_b as follows

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

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

$$s(v_a, v_b) = \frac{\sum_{i=1}^{|I(v_a)|} \sum_{j=1}^{|I(v_b)|} s(I_i(v_a), I_j(v_b)) \times p_{i \to v_a} \times p_{j \to v_b}}{|I(v_a)||I(v_b)|}, \tag{3}$$

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 $I(v_a)$. $p_{i \to v_a}$ and $p_{j \to v_b}$ are the corresponding edge weights. 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. Algorithm 4 calculates the similarity matrix. 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 according to Equation 3.

Algorithm 4 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
        for each node v in V do
            s_t(v,v) ← 1
    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}
                    s_t(v_a,v_b) ← 1/(|I(v_a)||I(v_b)|) s_t(v_a,v_b)
    S ← S_t
```

4 COMMUNITY RECOMMENDATION

Our community recommendation 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 5 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 nodes based on their importance and still preserve edge relations, 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 relations.

Algorithm 5 GRAPHTOSTRING($\mathbf{A}, V$)

```
Create a degree matrix D
for each node i in V do
    for each node j in V do
        if i = j then
            D(i,i) ← 1 / Σ_{k=1}^{|V|} A(i,k)
        else
            D(i,j) ← 0
Create a normalized symmetric random walks probability matrix P'
P' ← D^{1/2} A D^{1/2}
Find the leading eigenvector φ of 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 ← 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 ← the neighbor in s' which has the largest value in φ
return s
```

Algorithm 6 DIST($\mathbf{L}, \mathbf{A}_1, \mathbf{A}_2, V_1, V_2, \mathbf{P}_1, \mathbf{P}_2$)

```
Create the degree matrices D'_1 and 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) ← Σ_{j=1}^{|V_1|} A_1(i,j)
        else
            D'_1(i,j) ← 0
for each node i in V_2 do
    for each node j in V_2 do
        if i = j then
            D'_2(i,j) ← Σ_{j=1}^{|V_2|} A_2(i,j)
        else
            D'_2(i,j) ← 0
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))
```

Algorithm 7 GRAPHEDITDISTANCE($\mathbf{A}_1, \mathbf{A}_2, G_1 = (V_1, E_1), G_2 = (V_2, E_2)$)

```
s_1 ← GRAPHTOSTRING(A_1, V_1)
s_2 ← GRAPHTOSTRING(A_2, V_2)
P_1 ← the normalized symmetric random walks probability matrix of A_1
P_2 ← the normalized symmetric random walks probability matrix of A_2
Create a lattice L with size |V_1| × |V_2|
d(G_1, G_2) ← DIST(L, A_1, A_2, V_1, V_2, P_1, P_2)
return d(G_1, G_2)
```

After ordering the nodes, Algorithm 7 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 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 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

$$d = \beta_{(a,b)} \times \beta_{(c,d)} \times R_1(a,c) \times R_2(b,d), \tag{5}$$

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

$$\beta_{(a,b)} = \frac{\max\{D'_1(a), D'_2(b)\} - \min\{D'_1(a), D'_2(b)\}}{\max\{D'_1(a), D'_2(b)\}}, \tag{6}$$

where $\mathbf{D}'$ is the degree matrix calculated in Algorithm 6. $\beta_{(c,d)}$ can be calculated similarly. R_1 in Equation 5 can be calculated as

$$R_1(a,c) = \begin{cases} P'_1(a,c), & \text{if } A_1(a,c) = 1 \\ \frac{2 \times |V_1| \times |V_2|}{|V_1| + |V_2|}, & \text{otherwise} \end{cases}, \tag{7}$$

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