Rapid, Progressive Sub-Graph Explorations for Interactive Visual Analytics over Large-Scale Graph Datasets

Sam Armstrong*
Sam.Armstrong@rams.colostate.edu
Colorado State University
Fort Collins, Colorado

Kevin Bruhwiler*
Kevin.Bruhwiler@rams.colostate.edu
Colorado State University
Fort Collins, Colorado

Sangmi Lee Pallickara
Sangmi.Pallickara@colostate.edu
Colorado State University
Fort Collins, Colorado

ABSTRACT

Exploring a voluminous graph dataset visually is a challenging task due to the sheer amount of data and the lack of structure to rely on during the navigation. Indra, our framework for large-scale graph data, provides responsive visual analytics over large-scale graph datasets. In this study, we propose a novel graph indexing scheme that pivots the view of the graph to a hierarchical structure while preserving the semantic importance of vertices within the user's analysis scenario. Indra allows users to compare and track multiple aspects of sub-graphs by supporting linked multi-views and multi-resolution operations such as drill-in and roll-ups. We have performed a set of empirical benchmarks profiling Indra and these demonstrate that several operations are executed with sub-second latency to effectively support interactive visual analytics.

CCS CONCEPTS

• **Information systems** → **Summarization**; • **Human-centered computing** → **Visual analytics**; • **Software and its engineering** → *Client-server architectures*; • **Networks** → Network topology types.

KEYWORDS

Visual Analytics; Indra; Large Graph; Network Analysis; Graph Indexing; Graph Summarization

ACM Reference Format:
Sam Armstrong, Kevin Bruhwiler, and Sangmi Lee Pallickara. 2019. Rapid, Progressive Sub-Graph Explorations for Interactive Visual Analytics over Large-Scale Graph Datasets. In *Proceedings of the IEEE/ACM 6th International Conference on Big Data Computing, Applications and Technologies (BDCAT '19), December 2–5, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3365109.3368793

1 INTRODUCTION

The scale of graphs capturing real-world phenomena has grown dramatically. As of 2018, Facebook has 2.38 billion active users with an average 338 friends per account [1]. Similarly, there are more than 335 million Twitter users and 500 million tweets were sent daily in 2014 [2]. Effective visualization is critical to explore the dynamic and flexible graph data. However, the sheer number of objects that must be rendered far exceeds the physical memory limits; furthermore, flexibly defined interconnectivity included in a graph exacerbates the complexity of depicting them in a 2-dimensional graphical space.

There have been active research on summarizing graph structure to reduce data volumes, speedup graph mining, and provide visual analytics while preserving semantics of the original data. This includes approaches using clustering and classification [3–6], community detection [7–10], pattern set mining [11, 12], and visualization [13, 14]. However, applying graph summarization techniques to interactive visual analytics poses a unique set of challenges. First, most of the summarizing algorithms target volume reduction while preserving the key characteristics of the original dataset - often, this involves prioritizing vertices and edges to store. This process often results in loss of details that may be critical to discovering insights. Second, a user's interest in a large dataset is hard to capture at the beginning of data exploration. Interactive analytics frequently involves a series of actions to maneuver users' interests such as drill-in, roll-up, zooming, or panning. Applying a single data reduction algorithm (tuned to a particular usage) may limit the capability of data exploration.

We have designed and developed Indra, an infrastructure for large-scale graph visual analytics. In this paper, we propose a hierarchical indexing scheme for large graph datasets that is well-aligned with characteristics of the visual analytics applications such as a high-level pattern recognition and a progressive exploration. Indra provides interactive visualization features such as multi-linked views, OLAP style explorations (e.g. drill-down/roll-up/panning), and sub-graph analysis capability.

1.1 Research Questions

As part of this study, we explore the following research questions:

RQ-1: How can we provide an effective overview of a large graph? Views of a massive graph structure should capture an accurate and comprehensive summary at a given level to facilitate subsequent explorations.

RQ-2: How can we retrieve the details of sub-graphs as user's visual analytics progresses? The sub-graph visualized at each level must provide enough details so that users can understand characteristics and discover patterns in the current view.

RQ-3: How can the system process flexible queries to support real time interaction? Each exploration will involve 100s of data queries over the voluminous graph data. The back-end must provide a scalable and high-throughput query evaluation capability.

*Both authors contributed equally to this research.

BDCAT '19, December 2–5, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-7016-5/19/12…$15.00
https://doi.org/10.1145/3365109.3368793

RQ-4: Finally, how can the application compose different views of graph data and provide a comprehensive analytics environment?

1.2 Overview of Approach

In this study, we describe our hierarchical, sub-graph indexing scheme and comprehensive visual analytics features for a large-scale graph dataset. We have designed an algorithm to navigate sub-graph structures using dynamic hierarchical indexing schemes. First, to discover users' initial interests or support a coarse-grained pattern recognition, we index the graph to identify a set of the most representative nodes and edges. Since there may be multiple analytical aspects to the same dataset, INDRA allows users to select different indexing scheme algorithms for their analysis. The result of initial indexing forms the highest level of graph overview in INDRA. Based on the highly representative subset, INDRA performs sub-graph indexing as an off-line computation.

Data exploration features supported in OLAP (Online analytical processing) are widely adopted among the visual analytics tools. This includes support for features such as dicing, drill-down, roll-up, or panning. Data retrieval queries should be well-aligned with the aforementioned features to reduce query evaluation latency and avoid excessive memory consumption in the visualization application. Unlike other datasets such as geospatial or temporal datasets, defining resolutions of data representation in a graph is not straightforward. In our paper, we define the highest level (the coarsest resolution) as the set of the most representative vertices and their edges. We assume that traversing from one of the most representative vertices to any other vertex increases the level of resolution. The level of resolution in INDRA is defined as the shortest distance from a selected subset of the most representative vertices. Our hierarchical indexing scheme, called the *viewpoint path*, encapsulates the traversal path from the representative vertex to their reachable vertices. Instead of performing a graph traversal algorithm to generate each view, we directly retrieve the sub-graph information using our pre-calculated index.

Using INDRA's hierarchical sub-graph query, we designed and developed a comprehensive graph analytics application to support multi-linked views, hierarchical sub-graph exploration, and real-time drill-down/roll-up/panning features.

1.3 Paper Contributions

Our methodology allows real-time data retrievals of large-scale graph datasets. Our novel graph indexing scheme enables effective summarization and progressive navigation within analytics applications such as interactive visualizations. Our specific contributions include:

(1) A graph indexing scheme that preserves potential path of graph exploration across different resolution levels.

(2) A data indexing scheme that reduces the management overhead and dynamic updates of the graph data via a scalable, offline job orchestrated in a distributed computing environment.

(3) A high-throughput visual graph query evaluation that scales across applications that require graph visualization and analytics.

(4) Highly integrated graph visualization features that provide a comprehensive view of the graph and sub-graphs in real-time.

1.4 Paper Organization

The remainder of this paper is organized as follows. Section 2 discusses related work in graph summarization algorithms and visual analytics. Section 3 describes the methodology used in INDRA. The system architecture and visual analytics capabilities are described in Section 4. Experimental setups, performance benchmarks, and analysis of results are outlined in Section 5. Finally, our conclusions and future research directions are described in Section 6.

2 RELATED WORK

2.1 Graph Summarization

Any graph summarization algorithm can be generally categorized into one of two different approaches: importance summarization or similarity summarization. Importance summarization algorithms identify a subset of nodes and/or edges as important and create a summarized graph consisting exclusively of that subset. Similarity summarization algorithms attempt to discover nodes with similarities, in their topology or their attributes, and either combine them or discard them as redundant. In both algorithms the result is a graph consisting of a much smaller number of nodes that can be visually analyzed.

2.2 Importance Summarization Algorithms

Determining the importance of nodes in a graph is highly dependent on the graph in question, however there are a number of algorithms based on the graph's topology that can be reasonably applied to most datasets. *Shortest-path based*[15–18] summarization are common and applicable to many real-world datasets, including social media and collaboration networks. They rank nodes by their location in the network. There are two major variations on shortest-path based summarization: nodes are ranked either by their distance from some predetermined important node or by the number of shortest paths which the node lies on. *Degree based*[15] summarization, on the other hand, ranks the importance of a node by its degree under the assumption that the most connected nodes are the most important.

2.3 Similarity Summarization Algorithms

Like importance, there are many different ways of determining node similarity based on attributes, such as by real-world geographic location, node category, or edge types. However, there are a few universally applicable methods based on the graph's topology. The most popular of these is *redundancy elimination*[15, 19] (sometimes called *structural equivalence*[20]) which classifies two nodes as similar if they share a certain fraction of neighbors, the assumption being that those two nodes fulfill the same role in the network. A similar topological similarity summarization is *clique elimination*[15, 21, 22], which identifies cliques (sub-graphs in which every

node connects to every other node) in the graph and merges them into a single node. Additionally, some techniques, such as *CONDENSE*[23], take an information theoretic approach and attempt to eliminate nodes that do not belong to a unique structure[24].

2.4 Hierarchical Summarization Algorithms

Few attempts have been made at a hierarchical visualization of large graphs. The *SNAP*[25] and *k-SNAP*[25] graph summarization operators are one such attempt which allow the creation of graph summaries based on user defined attributes and even support drill-down and roll-up interactions. However, unlike Indra, they don't support panning sub-graphs and are limited to storing and processing graphs on a single machine. Many hierarchical summarization algorithms, such as the *Slice Tree*[26] algorithm, perform hierarchical summarization on graph attributes with the intention of enabling easy storage and fast database-esque queries[27, 28]. However, such algorithms often discard information related directly to nodes and edges and consequently are not suitable for graph visualization.

2.5 Graph Analysis

Domain-specific techniques have been developed for sub-setting[29, 30]. Graph network analyses have been used in to visualize super-spreaders and disease dynamics[31]. This work targets domain independent approaches to sub-setting and visualizing large graphs. *Mitra et al* [32] explore server-side caching and dynamic management of rendering resolutions over spatiotemporal phenomena. We view our work to be synergistic with efforts that leverage caching for rendering phenomena.

2.6 Graph Visualization

There exist a number of tools of visualizing graphs and networks. However, due to the enormous variety of datasets, objects, and concepts that can be abstracted as a graph, it may be impossible to create a single tool that can handle every use-case. Here we examine some of the most popular graph visualization tools and their specific use-cases.

2.6.1 GraphViz. GraphViz[33] is a popular, open-source graph visualization tool with scripting API's for many popular programming languages like Java and Python. It supports a number of different layouts, including spring-model layouts, and contains tools for basic visual interactivity. GraphViz's DOT format is accepted by a number of graph-related tools, including Gephi and NetworkX[34], making it a standard tool for graph visualization. However, unlike Indra, GraphViz does not support drill-down/roll-up operations and lacks sophisticated tools for visually navigating sub-graphs, making it unsuitable for viewing large, hierarchically indexed graphs.

2.6.2 Gephi. Gephi[35], like GraphViz, is an open-source tool for graph visualization, however it places a much greater emphasis on graph manipulation and exploration. It provides an exceptional user interface (UI) which allows users to highlight, hide, and organize graph visualizations. It also comes with a suite of graph analysis tools (average path length, clustering coefficient, etc) which are easily accessible to non-programmers. Consequently, Gephi is flexible and powerful enough to provide support for most graph visualization use-cases. However, Gephi runs on a single machine and does not provide tools for topology summarization. This limits the size of the graphs Gephi can visualize to what a single machine can handle and results in poor performance when visualizing exceptionally large graphs.

2.6.3 Cytoscape. Cytoscape[36] is a similar tool to Gephi which specializes in visualizing biological networks. Unlike Gephi, Cytoscape has a JavaScript library which provides interactive tools for graph visualization and analysis in a web browser. It provides a large variety of different graph layouts and support for use on mobile devices. However, like GraphViz, it lacks support for drill-down/roll-up operations, the ability to navigate across sub-graphs, and the ability to scale horizontally to multiple machines.

3 METHODOLOGY

OLAP style analytics allow users to explore the data with varying resolution and progressive coverage. To retrieve sub-graphs effectively, Indra provides a hierarchical sub-graph retrieval query using the viewpoint path indexing scheme.

3.1 Hierarchical Sub-Graph Retrieval Query (RQ1)

The resolution of the data representation is often defined differently based on the context. In Indra, we define the resolution of graph data representation as the distance from the most representative sub-graph. Consider a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. We define a sub-graph G_R as the most representative sub-graph of G and it contains the most important vertices (super-nodes) in the graph G. Section 3.2.1 explains how Indra defines the importance of nodes and computes G_R. The sub-graph G_R provides the highest level overview of the graph G with the lowest resolution of the data representation.

To explore less important vertices and edges, users can issue a hierarchical sub-graph retrieval query with 2 main input parameters: the *vantage vertex* (v) in G_R, and the desired level of resolution (r). The vantage vertex is the vertex that is closely related to the user's interest. The output of this query, G_{query} is a subset of G and it includes all vertices that are reachable from v with distance r. As a G_{query} shares greater r, the resolution of the given sub-graph has a higher resolution based on the vantage vertex, v. In Indra, r is measured as the number of hops from the sub-graph to the representative node v. The cost r is extensible with the weights or other metrics.

3.2 Viewpoint Path: Hierarchical Graph Indexing Scheme (RQ2, RQ3)

Understanding a voluminous graph is challenging because of the excessive amount of unstructured data and its complexity. Narrowing down the scope of the graph to a manageable size is critical to performing timely analyses. However, existing summarization techniques result in unavoidable information loss of meaningful nodes and edges. To circumvent this, we have designed a hybrid scheme that combines summarization with a reachability-oriented graph indexing scheme - the viewpoint path. Our approach reduces

the memory footprint in the visualization application by means of aligning the query output with the user's visual analytics activities.

As depicted in Figure 1, the original graph G is segmented into a sub-graph of super-nodes (G_R) and a set of sub-graphs comprising a super-node and its reachable nodes (G_S={G_{S1}, G_{S2}, G_{S3}, ...}, where G_{S1},G_{S2},G_{S3} ... are subsets of G). This pivots the view of the graph G to a fractal style hierarchical structure with the topmost level view of super-nodes and downstream their sub-graphs. Based on the algorithm selecting the super-nodes, some vertices and their associated edges may be ignored if they are not reachable from any of the super-nodes. A non-super-node may belong to multiple sub-graphs if there are multiple reachable super-nodes. A user can specify one of these super-nodes and then descend to view a sub-graph that represents only a portion of the original graph. This allows users to both drill-down to a specific location of interest and roll-up to view an arbitrarily large portion of the graph without triggering excessive network communications or entailing significant memory consumption.

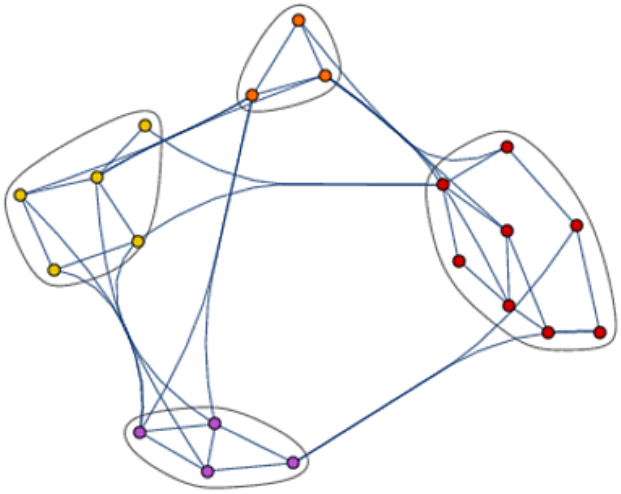

Figure 1: An illustration of sub-graphs contained within super-nodes. In INDRA each node in the sub-graphs is also a super-node containing its own sub-graph.

3.2.1 Initial Graph Summarization. To generate an initial representative sub-graph with super-nodes (G_R), INDRA relies on existing topology summarization algorithms. Since the importance of a node is subjective based on the angles of the analysis, INDRA allows users to select/examine the initial indexing scheme algorithm.

INDRA supports the initial indexing scheme algorithms listed below. We have evaluated the overall performance of these algorithms in section 5.

3.2.2 Degree-Indexing. The Degree-Indexing (DI) algorithm [15] is an importance-based graph indexing algorithm. First, DI assigns importance of each vertex based on the degree or other vertex values based on the context of analysis (e.g. the shortest path from the vertex to another chosen vertex, the number of shortest paths which pass through the vertex, or some other attribute of the vertex). Then, a set of nodes with the highest importance values are selected. Since DI captures the importance of each vertex, it is applicable for social network type graph analyses.

3.2.3 KeepAll. Like DI, the KeepAll Algorithm [15] is an importance-based graph indexing algorithm. KeepAll identifies a subset of key vertices; the shortest paths between the subset of key vertices are calculated and all the nodes which lie on them are labeled as important. We used the degree of a vertex as the importance metric for KeepAll and DI. In contrast to DI, KeepAll preserves the closely connected vertices (to the initial important vertices) as well. Obviously, KeepAll requires additional computations to perform the shortest path algorithm.

3.2.4 Jaccard-Indexing. The Jaccard-Indexing (JI) algorithm [20] focuses on structural equivalence. It classifies the network nodes into categories by their positions in the network. Because JI preserves the context and topology details, it is a suitable to understanding flow patterns (e.g. traffic analysis). First, the neighboring vertices for each node are collected and stored as a set of vertex IDs. Then, the Jaccard Similarity (1) between the set at each vertex and its immediate neighbors is computed and summed.

$$Jaccard(X, Y) = |X \cap Y|/|X \cup Y| \tag{1}$$

By summing the Jaccard Similarity, nodes with a high number of similarly structured neighboring nodes will be assigned a high value. Finally, N nodes with the highest values are labeled as important. Intuitively, nodes that make many of their neighbors redundant are preserved in the indexing scheme.

3.2.5 Generating Viewpoint Path Index. Once the initial important subset (G_R), is determined, INDRA computes viewpoint paths, which provides a uniquely identified location of each node in any given sub-graph. Elements of the array are the IDs of nodes related to the path from a super-node of the sub-graph to the current node. The length of the viewpoint path is configurable; this limits the number of layers one can drill-down into a graph before reaching the viewpoint path bottom-level.

Listing 1 and 2 shows pseudo code of the iterative computation of the hierarchical indexing scheme. When updating the viewpoint path for each node, it should be noted that the final element in the adjacent node's sub-array will be the adjacent node's ID. As a result, each node takes the partial viewpoint path from its most valuable neighbor and propagates that neighbor's ID to the end, assigning itself to that neighbors' super-node at the current level of viewpoint path. It also takes that neighbor's importance as its own. The constant C in line 7 defines the threshold at which a node switches super-nodes. If the difference between a node's current super-node importance and its neighboring super-node's importance is greater than C, it will make the switch. In the case of a binary importance value, such as in figure 2, C has no effect.

Listing 1: The Hierarchical Indexing Scheme Algorithm

```
1:  level = 0
2:  recent_update = True
3:  while recent_update do
4:    recent_update = False
5:    for node in nodes
6:      for adj_node in adjacent_nodes(node)
7:        if adj_node.value > node.value + C
8:          node.update(adj_node, level)
9:          recent_update = True
10:   level = max(level++, max_level)
```

Listing 2: The Viewpoint Path Update Function

```
1:  Function update(adj_node, level):
2:    this.viewpoint_path[0:level] =
          adj_node.viewpoint_path[0:level]
3:    this.value = adj_node.value
```

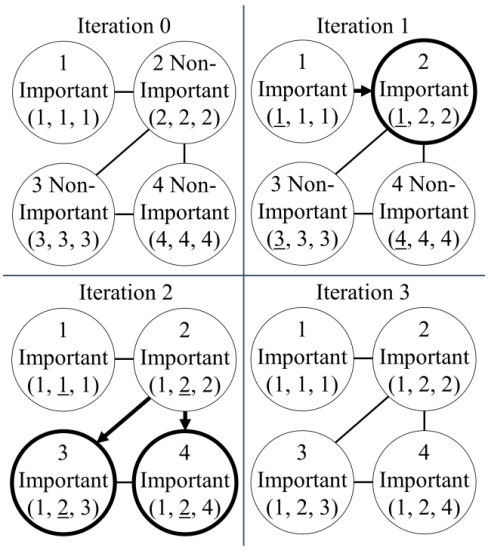

Figure 2: A simple illustration of the hierarchical indexing scheme algorithm. Node attributes consist of an ID, an importance value, and a viewpoint path.

Figure 2 depicts a simple example of the indexing scheme with the viewpoint path length of 3. Note that the viewpoint path for each node is initialized with the node's ID. To complete indexing, 3 iterative computations must be performed. In iteration 3, no node is updated and the algorithm terminates and each viewpoint path contains the id (1) from the super-node.

For a graph with n vertices and a viewpoint path length of m, the iterative indexing scheme entails a computational complexity of $O(mn)$, which is computationally intensive especially for large graphs. To address this issue, Indra leverages Apache Spark [37] and its Pregel [38] APIs, which provides a massively parallel data processing environment and selectively performs computation on nodes only if updates are needed. We have observed that using these frameworks with our optimizations considerably reduced the run time (benchmarks included in section 5).

4 SYSTEM ARCHITECTURE AND CAPABILITIES

As illustrated in figure 3, Indra consists of a client application and a back-end server running on a commodity machine cluster. The client application tracks the user's location, provides operations to navigate the graph, and translates user's actions into data retrieval queries. The back-end server performs graph indexing using the viewpoint path algorithm and evaluates user's queries. The server and client communicate using an HTTP connection. The key requirement of this architecture is to provide sub-second latencies that facilitates real-time interactivity during explorations over voluminous graph datasets.

4.1 Client Application

Indra's client is a web-based application that consists of roll-up controls (at the top), graph layout (in the middle), and distribution displays (at the bottom). Roll-up controls enable the user to step up either one level in the hierarchical graph path or jump directly to the top of the hierarchy. The graph layout shows the current sub-graph and an abstraction of all adjoining sub-graphs. It also displays the ID of the sub-graph near the top of the GUI. The distribution displays provide some metadata about the current sub-graph and support user interactivity. A snapshot of Indra's visual interface is provided in figure 4.

4.1.1 Rendering Views. The client application leverages D3.js [39] to render the graph and results of the built-in analytics features. D3.js is a widely used open-source JavaScript library for data visualization. The graph layout in the center of the GUI consists of two nested force-directed graphs (often referred to as spring-model layouts). Spring-model layouts achieve a well-distributed uniform view of a graph via a mechanical simulation in which edges pull nodes together while nodes repel each other. The simulation is run until an equilibrium, a state in which no nodes are changing their location, is reached. Spring-model layouts do not require specific knowledge of graph theory and often result in graphs with near-uniform edge length, even node distribution, and semi-symmetrical layouts

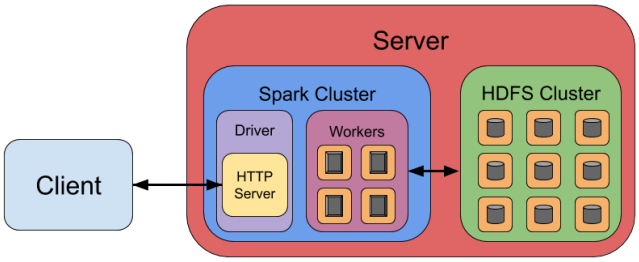

Figure 3: An illustration of Indra's system architecture. The Apache Spark cluster creates and caches an indexed graph using data from an HDFS cluster. The client sends queries to an HTTP server residing in the driver node of the Spark cluster, which generates results from the cached graph.

[40]. In INDRA, the abstracted adjoining sub-graphs are displayed as a force-directed graph with the current sub-graph displayed as another force-directed graph within the central node of the abstraction. INDRA utilizes D3.js to create the distribution displays that provide real-time interactivity during the data rendering.

4.2 Server

INDRA's server consists of an Apache Spark cluster and a HTTP server. Spark is the key computing component for generating a hierarchical index of the graph using viewpoint paths. The HTTP server resides on the driver node of the Spark cluster and coordinates the cluster's activities and collects data from the cluster in response to queries sent by client applications.

4.2.1 Distributed Computing Environments. Low latency is key for interactive analytics that chains steps of analytics and user's cognitive understanding [41]. Interactive exploration of a large dataset, or simply reading the data, often takes minutes or hours. To mitigate this issue and provide sub-second response times, INDRA employs a distributed Apache Spark [37] cluster.

Apache Spark is an open-source, distributed data-processing engine that uses clusters of machines to partition the data across the main memory of multiple machines and orchestrate data processing in parallel. In our implementation, Spark reads data from a Hadoop Distributed File System (HDFS)[42] cluster, which is a distributed file system designed for large datasets with fault-tolerance.

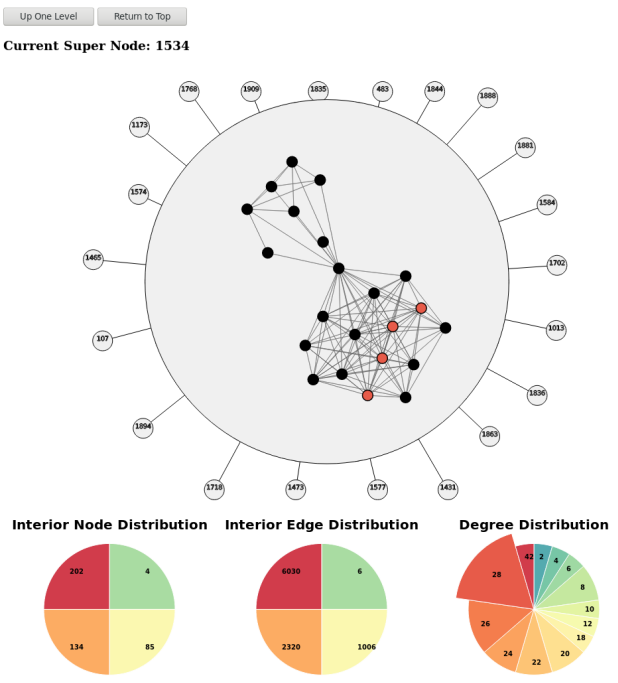

Figure 4: An example of INDRA's visual interface. The current sub-graph is shown in the center, with pie-charts providing metadata about the sub-graph at the bottom. Controls for rolling-up are provided at the top of the interface. The sub-graphs that can be panned to are represented as larger nodes surrounding the sub-graph.

We leverage Spark's existing library, GraphX [43], that is designed for performing graph computations over large-scale graphs. To perform our graph indexing scheme (described in section 3.2), INDRA used the implementation of Dijkstra's shortest-path algorithm supported by GraphX. To compute viewpoint path indexing, we have leveraged the Pregel [38] implementation included in the GraphX library. We have observed that the Bulk Synchronous Parallel model of Pregel was well-aligned with our iterative computing requirements. The benchmarks of the computing using Spark and its libraries are included in section 5.

4.2.2 Performing the Viewpoint Path Algorithm (RQ1, RQ2). INDRA indexes the graphs in two stages: initial indexing scheme and calculating the viewpoint path. In the initial indexing scheme stage, the user-defined indexing scheme algorithm is applied to the graph. The result of this indexing scheme must comprise a subset of the initial graph's nodes and an importance value associated with each (G_R). This step is computed off-line.

To calculate the viewpoint path for each node, INDRA takes the desired length of the path, the original graph, and the indexed graph (G_R) as inputs and generates a viewpoint path for each vertex. The results are cached in memory to provide rapid query evaluation.

4.2.3 Sub-Graph Query Evaluation (RQ3). INDRA's sub-graph query allows the client application to retrieve a subset of vertices and edges that share the same distance from the vantage vertex. The user's actions such as drill-down or roll-up are translated to a sub-graph query by the client application. As described in Section 3, INDRA specifies the sub-graph with the most representative vertices as the highest-level overview (level-0 resolution). The current view of a sub-graph (at the resolution level of n) displays a set of vertices with their viewpoint path containing the same sequence of indices from the 0^{th} to the $(n-1)^{th}$ elements. For the actions exploring an immediately higher resolution (e.g. drill-down), the server will return a set of vertices with the viewpoint path containing the identical sequence of the 0^{th} to n^{th} elements, where the n^{th} element is the vertex ID of the current vantage vertex. INDRA evaluates this query over the memory resident viewpoint paths computed off-line. All edges associated to the selected vertices are included in the query output.

4.3 Interactive Visualization Features (RQ4)

INDRA provides several interactive features that are designed to enable easy exploration of large graphs, including linked-view metadata about the sub-graph currently being viewed, the ability to drill-down to a specific sub-sub-graph or roll-up to the current super-graph, and the ability, unique to INDRA, to see and traverse across adjoining sub-graphs.

4.3.1 Multi-Linked Views. Multi-linked views describe a system in which multiple visual displays are used to show a variety of information about a single object or system. The displays are also linked, such that a modification to one automatically triggers an equivalent modification in another. In INDRA, we display both a visualization of the graph and a series of pie-charts providing additional information about the nodes displayed. The views are bound

to each other by a visualization technique, brushing and linking; highlighting or selecting any portion of the pie-charts will highlight the corresponding nodes in the graph visualization and update the distributions shown in the pie-charts. Conversely, changing the sub-graph being visualized will update the pie-charts with information about the new sub-graph.

4.3.2 Sub-Graph Analysis. INDRA provides pie-charts illustrating the distribution of node degrees, of interior nodes (the nodes within each super-node), and of interior edges (the edges com- pressed within each super-node) within the current sub-graph. As depicted in Figure 4, mousing-over a section of a pie-chart will highlight all the nodes that fall within that section with the corresponding color. Note that the other distribution displays have also been updated to match the highlighted portion. Clicking on the section will keep the nodes highlighted even after mousing-off. Multiple sections of multiple pie-charts can be selected simultaneously with nodes being highlighted in a ring-nested fashion. This allows users to quickly identify all the super-nodes which meet a certain set of criteria *i.e.* all super-nodes which have a degree of five and contain 30 interior edges.

4.3.3 Drill-Down and Roll-Up. INDRA allows users to drill-down and roll-up through visualizations. These operations are best described as changing the horizontal scope of the visualization. Drilling-down narrows the scope to sub-graphs comprising fewer total nodes, while rolling-up broadens the scope to sub-graphs comprising more total nodes. They are comparable to zooming-in and zooming-out, however while zooming merely magnifies or minimizes a portion of the visualization, drill-downs and roll-ups actually change the information being displayed. In INDRA this entails showing a different sub-graph at a different level of resolution.

4.3.4 Navigating Sub-Graphs. When navigating graphs that cannot be viewed in their entirety it is essential that a user always has a clear idea of the location of the sub-graph currently being viewed with respect to the graph as a whole. To that end, INDRA attempts to provide a simple and intuitive interface for navigation. At any given time, the sub-graph representing the current location is displayed in the center of the interface. Users can get information about lower-level sub-graphs by hovering over their corresponding super-nodes and drill-down to them by clicking on the super-node. Users can roll-up to either the encapsulating super-node or the very top level of the visualization by clicking one of the buttons near the top of the interface. INDRA also displays an abstraction of all the adjoining sub-graphs as a network around the edges of the interface. Users can pan to an adjoining sub-graphs by clicking on the corresponding node.

4.3.5 Panning Operations. Panning in INDRA is the process of moving across sub-graphs on a single layer and can be thought of as changing the vertical scope of the visualization. Panning does not change the viewpoint path level or significantly impact the portion of the graph being displayed, it simply changes the current view from one sub-graph to an adjoining sub-graph. In order to reduce the complexity of both the visualization and the navigation controls, panning is restricted to adjoining sub-graphs; it is not possible to jump directly to a sub-graph disconnected from the one currently being visualized. This is due to the exceptionally large number of sub-graphs at lower viewpoint path levels. However, such jumps can be made by rolling-up one level and selecting the desired sub-graph from the new visualization.

5 EVALUATION

We evaluate INDRA on several factors, including the time required to index with the viewpoint path algorithm, the quality of the generated indexing scheme, and the server's throughput and latency in responding to queries. We use two 1,000,000 node datasets in our indexing scheme time and throughput/latency evaluations: a social network graph collected from Pokec, a Slovakian social network exhibiting small-world properties and containing 30,000,000 edges, and a graph generated from Pennsylvania's road network which exhibits very sparse connectivity with only 1,000,000 edges. Our indexing scheme quality evaluation was performed with the DBLP (a database of computer science articles, papers, and other publications) collaboration graph consisting of 300,000 nodes and 1,000,000 edges, and containing labeled communities.

Our benchmarks were performed on a Spark cluster consisting of 70 machines, each with 4 2.60GHz CPUs and 10GB of memory. Sample graphs were generated by selecting nodes and their connecting edges from the datasets by randomly assigning a unique ID to each vertex and removing all vertices with an ID less than the sample size.

5.1 Hierarchical Indexing Scheme Time

We compare INDRA's hierarchical indexing scheme time across initial indexing schemes generated by the Degree-Indexing, KeepAll, and Jaccard-Indexing algorithms in **figure 5**. The output of each initial indexing scheme is a graph consisting of 100 important vertices and a variable number of edges.

In every case the indexing scheme on the social network dataset is significantly faster than the indexing scheme on the road dataset. This is due to the fact that the upper-bound on the number of iterations INDRA's indexing scheme algorithm must perform is the distance between the most important node and the node furthest from it. Because social networks exhibit small-world properties the average path-length between any two nodes is exceptionally low. Consequently, the maximum number of iterations needed to index the social-network graph is much lower.

In **figure 5** it can be seen that the required indexing scheme time increases sub-linearly; doubling the size of the graph does not double the indexing scheme time. Remarkably, the indexing scheme time on the social network graph appears to increase *logarithmically*. When the size of the graph increases by three orders of magnitude the indexing scheme time increases by approximately one. This bodes exceptionally well for INDRA's performance on graphs consisting of hundreds of millions to billions of nodes, and suggests that the bottle-neck will be storage space and not computation time.

Finally, it is also apparent that the hierarchical indexing scheme time is not independent of the initial indexing scheme, as the indexing scheme initialized with KeepAll completes twice as quickly as Degree-Indexing and Jaccard-Indexing. This is most likely the result of the Degree and Jaccard Indexings providing a subset of nodes distributed widely across the graph while the KeepAll algorithm

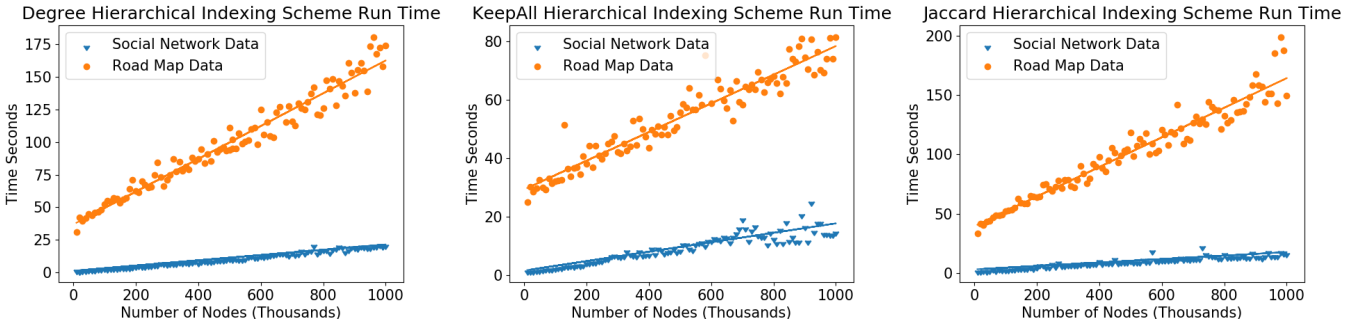

Figure 5: INDRA's hierarchical indexing scheme run-time based on the initial indexing scheme. The social network graph always indexes much faster due to its small-world properties. Starting with KeepAll yields the fastest indexing scheme time and Jaccard-Indexing the slowest.

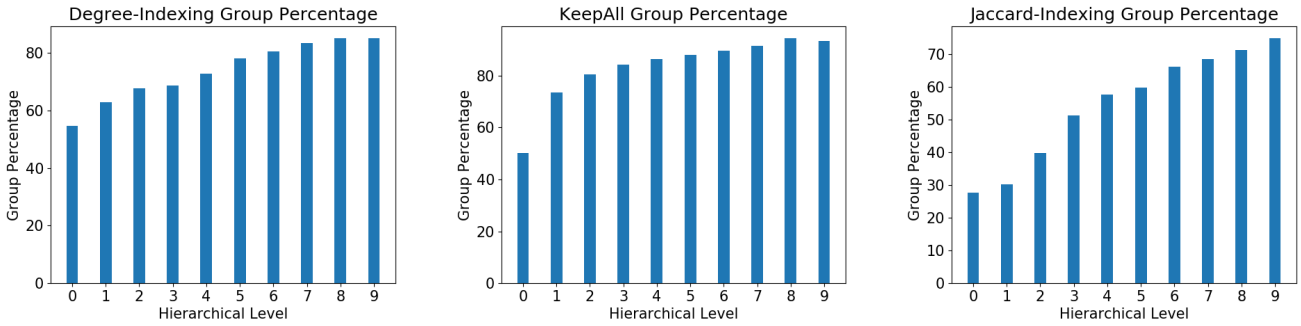

Figure 6: The average percentage of nodes in each sub-graph which belong to the same ground-truth community at each level of indexing. Desirable qualities are a low percentage at the top levels and a high percentage at the lower levels. Based on this measure, starting with a Jaccard-Indexing appears to yield to best results on the collaboration graph.

provides a connected subset of nodes. The tightly clustered nodes from KeepAll have fewer total neighbors and consequently prompt fewer computations during each iteration. The explanation for why initializing with Degree-Indexing appears to be slightly slower than with Jaccard-Indexing is similar, as the important nodes selected by Degree-Indexing are biased towards nodes with many neighbors.

5.2 Quality of Summarization

There is no standard method to evaluate the quality of a hierarchical graph summarization. Consequently, we choose to measure the quality of INDRA's hierarchical indexing scheme by computing the proportion of nodes in each sub-graph which belong to a ground-truth community. The assumption behind this measure is that sub-graphs with a higher resolution should represent a portion of a single community with most super-nodes in the sub-graph belonging to that community. Conversely, sub-graphs with a low resolution should represent interactions between communities and contain a diverse set of super-nodes belonging to many different communities. Under these assumptions, a high quality indexing scheme should have a relatively low proportion of nodes in the same ground-truth community in low-resolution sub-graphs and a high proportion of nodes in the same community in high resolution sub-graphs.

We perform this evaluation with the DBLP collaboration graph. The nodes represent authors, the edges connect two authors who have published at least one paper together, and the ground-truth communities are publishing venues. It is possible for an author to have published in multiple venues and belong to multiple ground-truth communities. The ratio we compute for each sub-graph is the number of super-nodes belonging to the most represented community in the sub-graph against the total number of super-nodes in the sub-graph. We compute the average across every sub-graph at each viewpoint path level. The results are shown in **Figure 6**.

Based on our established criteria, starting with Jaccard-Indexing produces the best hierarchical indexing scheme. At the top level less than $1/3^{rd}$ of the vertices belong to the same ground-truth community, while at the bottom level almost $4/5^{ths}$ of vertices in each sub-graph do. Degree-Indexing reaches a higher proportion of member vertices at low levels than Jaccard-Indexing, however it lacks diversity at high levels. KeepAll performs well at both low levels and the highest level, but there is very little diversity in its intermediate levels (although this may be desirable for some use-cases).

These results are not surprising as Jaccard-Indexing is designed for collaboration and interaction graphs. However, it does illustrate the manner in which the initial indexing scheme algorithm used for INDRA influences the properties of its hierarchical indexing scheme.

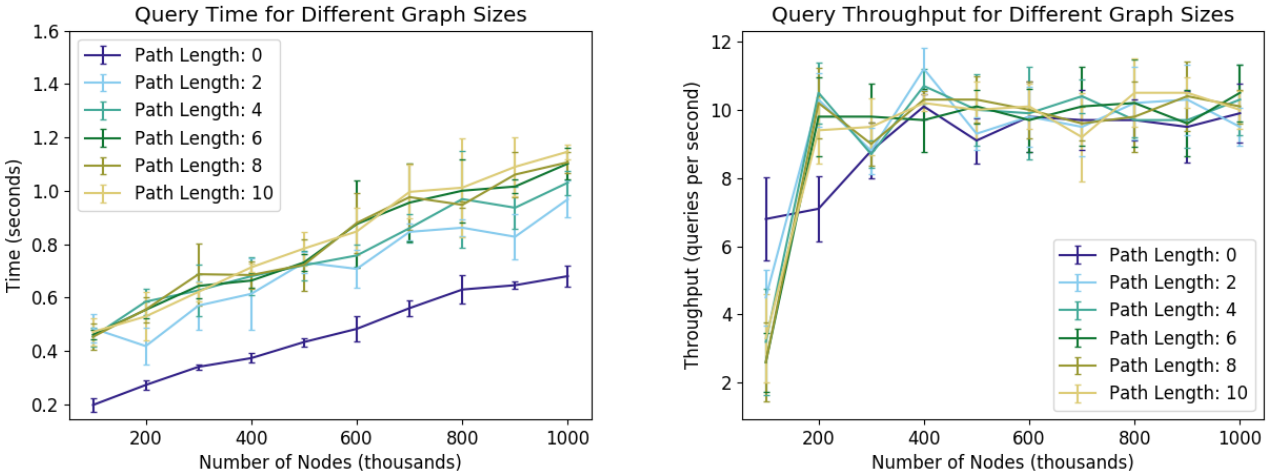

Figure 7: Indra's latency and throughput querying graphs of different sizes, averaged over twenty 10-second runs. Querying with a viewpoint path length of zero requires no comparisons and is very fast. The difference in query time between viewpoint paths of lengths greater than zero is visible, but small. Throughput is independent of path length for graphs larger than 200,000 nodes.

It also demonstrates how Indra successfully reduces the diversity of sub-graphs at each successive level of the hierarchical indexing scheme as the scope of the visualization narrows from interactions between communities to interactions within communities.

5.3 Interactive Latency

Speed in an interactive application is paramount, so it is essential that Indra is capable of processing queries with a very low latency. Additionally, Indra's server based nature makes it possible for multiple clients to simultaneously explore the same dataset. To make this feasible a high throughput is required to handle many queries at the same time without adversely impacting performance. We analyze Indra's latency and throughput in **figure 7**.

It should be noted that the latency is heavily dependent on the number of processors in the cluster, however it can be seen that Indra scales well with an increasing number of nodes. Doubling the size of the graph increases the query time by approximately 50%. Queries with a path length of zero are exceptionally fast as they merely filter for the initial indexing scheme and don't need to compare viewpoint paths. Queries with a viewpoint path are slightly slower, however the length of the viewpoint path has a relatively small effect on latency. The difference in latency between queries with a viewpoint path of length two and length ten averages less than $1/5^{th}$ of a second.

Figure 7 demonstrates the remarkable fact that Indra's throughput is largely independent of the size of the graph being queried, remaining at approximately 10 queries per second for graphs larger than 200,000 nodes. This suggests that a single Indra server can accommodate at least 10 simultaneous clients and probably more, as clients are unlikely to be performing queries every second. It's unclear why the throughput is lower for small graph sizes, however it is most plausibly because the graph is more sparsely distributed across the cluster, incurring higher overheads.

6 CONCLUSIONS AND FUTURE WORK

In visual analytics applications, having a structure that assists a user's navigation is critical. This is especially true when exploring voluminous graph datasets. Indra's viewpoint path indexing scheme provides a semantically pivoted hierarchical view to a non-hierarchical graph. The highest-level overview consists of the most semantically important vertices and supports maximum details of less important graph components. Users can specify the initial indexing scheme algorithm based on the objectives of their analysis (RQ1 and RQ2).

To provide the interactive responsiveness to visual analytics applications, data rendering, retrieval query evaluation, and batch computation must be seamlessly streamlined to reduce the end-to-end latency. The compute-intensive indexing phase is performed completely off-line and data retrieval relies on a distributed query evaluation over memory resident indices. Indra orchestrates data retrievals for multiple views without redundancy to reduce latency (RQ 3).

Finally, to facilitate comprehensive exploration of large datasets, a visual analytics application must provide multiple views for the same subset and orchestrate these views in real-time. Indra provides multiple analytics views and resolutions of data. The dashboard consists of graph view, charts, and sub-graph views. It also allows users to drill-down and roll-up the current view (RQ4).

As part of our future work we plan to explore automatic identification/updates of data summarization based on a user's exploration patterns. We will also extend our visualization features to support user-customized computations such as calculating the diameter or number of triangles within a user-specified sub-graph.

ACKNOWLEDGMENTS

This research was supported by grants the US National Science Foundation [OAC-1931363, ACI-1553685], the US Department of

Homeland Security [D15PC00279], the Advanced Research Projects Agency- Energy(ARPA-E), and a Cochran Family Professorship. We thank David Lee for his assistance during initial discussions.

REFERENCES

[1] Facebook. Facebook Newsroom stats, 2019.

[2] Twitter. Twitter Q1 2019 Earnings Report. https://s22.q4cdn.com/826641620/files/doc_financials/2019/q1/Q1-2019-Slide-Presentation.pdf, March 2019.

[3] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 587–596, New York, NY, USA, 2011. ACM.

[4] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, Sep 2007.

[5] K. M. Borgwardt and H. P. Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, Nov 2005.

[6] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Compression picks item sets that matter. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006*, pages 585–592, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[7] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008.

[8] Deepayan Chakrabarti, Yiping Zhan, Daniel Blandford, Christos Faloutsos, and Guy Blelloch. Netmine: Mining tools for large graphs. 01 2004.

[9] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. ACM.

[10] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In Mohammed J. Zaki, Jeffrey Xu Yu, B. Ravindran, and Vikram Pudi, editors, *Advances in Knowledge Discovery and Data Mining*, pages 435–448, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 433–444, New York, NY, USA, 2008. ACM.

[12] Xifeng Yan and Jiawei Han. gspan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, Dec 2002.

[13] Cody Dunne and Ben Shneiderman. Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3247–3256, New York, NY, USA, 2013. ACM.

[14] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. The State of the Art in Visualizing Group Structures in Graphs. In R. Borgo, F. Ganovelli, and I. Viola, editors, *Eurographics Conference on Visualization (EuroVis) - STARs*. The Eurographics Association, 2015.

[15] Anna C. Gilbert and Kirill Levchenko. Compressing network graphs. In *Workshop on Link Analysis and Group Detection*, Seattle, WA, USA, 2004. ACM.

[16] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *KDD '11 Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973, San Diego, California, USA, 2011. ACM.

[17] Fang Zhou, Sebastien Malher, and Hannu Toivonen. Network simplification with minimal loss of connectivity. In *2010 IEEE International Conference on Data Mining*, pages 659–668, Sydney, NSW, 2010. IEEE.

[18] Daniel Hennessey, Daniel Brooks, Alex Fridman, and David Breen. A simplification algorithm for visualizing the structure of complex graphs. In *International Conference on Information Visualisation (IV)*, pages 616–625, London, 2008. IEEE.

[19] Jie Sun, Erik M. Bollt, and Daniel ben-Avraham. Graph compression—save information by exploiting redundancy. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(6):06001, Jun 2008.

[20] Lei Shi, Qi Liac, Xiaohua Sun, Yarui Chen, and Chuang Lin. Scalable network traffic visualization using compressed graphs. In *IEEE International Conference on Big Data*, pages 606–612, Silicon Valley, CA, 2013. IEEE.

[21] Tomas Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. In *STOC '91 Proceedings of the twenty-third annual ACM symposium on Theory of Computing*, pages 123–133, New Orleans, Louisiana, USA, 1991. ACM.

[22] Ryan A. Rossi and Rong Zhou. Graphzip: a clique-based sparse graph compression method. In *Journal of Big Data 2018*, pages 5–10. Springer Nature, 2018.

[23] Yike Liu, Tara Safavi, Neil Shah, and Danai Koutra. Reducing large graphs to small supergraphs: a unified approach. In *Social Network Analysis and Mining*, pages 8–17. Springer Vienna, 2017.

[24] Lixia Zhang and Jianliang Gao. Incremental graph pattern matching algorithm for big graph data. In *Scientific Programming Volume 2018*, page 8 pages. Hindawi, 2018.

[25] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580, Vancouver, Canada, 2008. ACM.

[26] Arlei Silva, Petko Bogdanov, and Ambuj K. Singh. Hierarchical in-network attribute compression via importance sampling. In *2015 IEEE 31st International Conference on Data Engineering (ICDE)*, pages 951–962, Seoul, South Korea, 2015. IEEE.

[27] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. Query preserving graph compression. In *SIGMOD '12 Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 157–168, Scottsdale, Arizona, USA, 2012. ACM.

[28] Anurag Khandelwal, Zongheng Yang, Evan Ye, Rachit Agarwal, and Ion Stoica. Zipg: A memory-efficient graph store for interactive queries. In *SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1149–1164, Chicago, Illinois, USA, 2017. ACM.

[29] Yogesh L. Simmhan, Sangmi Lee Pallickara, Nithya N. Vijayakumar, and Beth Plale. Data management in dynamic environment-driven computational science. In Patrick W. Gaffney and James C. T. Pool, editors, *Grid-Based Problem Solving Environments*, pages 317–333, Boston, MA, 2007. Springer US.

[30] Sangmi Lee Pallickara, Shrideep Pallickara, and Milija Zupanski. Towards efficient data search and subsetting of large-scale atmospheric datasets. *Future Gener. Comput. Syst.*, 28(1):112–118, January 2012.

[31] N. Shah, H. Shah, M. Malensek, S. L. Pallickara, and S. Pallickara. Network analysis for identifying and characterizing disease outbreak influence from voluminous epidemiology data. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1222–1231, Dec 2016.

[32] S. Mitra, P. Khandelwal, S. Pallickara, and S. L. Pallickara. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. In *IEEE International Conference on Cluster Computing (CLUSTER)*, Albuquerque, NM, USA., 2019.

[33] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, and Lucent Technologies. Graphviz - open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[34] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA, 2008.

[35] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks. *International AAAI Conference on Weblogs and Social Media*, 2009.

[36] Shannon P., Markiel A., Ozier O., Baliga NS., Wang JT., Ramage D., Amin N., Schwikowski B, and Ideker T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research 2003 Nov; 13(11):2498-504*, 2003.

[37] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, Boston, MA, 2010. USENIX.

[38] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, Indianapolis, Indiana, USA, 2010. ACM.

[39] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. In *IEEE Transactions on Visualization and Computer Graphics Volume 17 Issue 12, December 2011*, pages 2301–2309, Piscataway, NJ, USA, 2011. IEEE.

[40] Peter Eades and Xuemin Lin. Spring algorithms and symmetry. In *Theoretical Computer Science Volume 240, Issue 2*, pages 379–405. Elsevier, 2000.

[41] John A Hoxmeier and Chris Dicesare. System response time and user satisfaction: An experimental study of browser-based applications. *Proceedings of the Association of Information Systems Americas Conference*, 01 2000.

[42] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *MMSST '10 Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, Washington, DC, USA, 2010. IEEE.

[43] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. Graphx: graph processing in a distributed dataflow framework. In *OSDI'14 Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*, pages 599–613, Broomfield, CO, 2014. USENIX.