Large-scale graph visual analytics

By

Fangyan Zhang

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

October 2017

Large-scale graph visual analytics

By

Fangyan Zhang

Approved:

Song Zhang
(Major Professor)

J. Edward Swan II
(Co-advisor)

Andy D. Perkins
(Committee Member)

Pak Chung Wong
(Committee Member)

T. J. Jankun-Kelly
(Graduate Coordinator)

Jason Keith
Dean
The James Worth Bagley
College of Engineering

Name: Fangyan Zhang

Date of Degree: October 20, 2017

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Song Zhang

Title of Study: Large-scale graph visual analytics

Pages of Study: 122

Candidate for Degree of Doctor of Philosophy

Large-scale graph analysis and visualization is becoming a more challenging task, due to the increasing amount of graph data. This dissertation focuses on methods to ease the task of exploring large-scale graphs through graph sampling and visualization. Graph sampling aims to reduce the complexity of graph drawing, while preserving properties of the original graph, allowing analysis of the smaller sample which yields the characteristics similar to those of the original graph. Graph visualization is an effective and intuitive approach to observing structures within graph data. For large-scale graphs, graph sampling and visualization are straightforward strategies to gain insights into common issues that are often encountered.

This dissertation evaluates commonly used graph sampling methods through a combined visual and statistical comparison of graphs sampled at various rates based on random graphs, small-world graphs, scale-free graphs, and real-world graphs. This benchmark study can be used as a guideline in choosing the appropriate method for a particular

graph sampling task. In addition, this thesis proposes three types of distributed sampling algorithms and develops a sampling package on Spark. Compared with traditional/non-distributed graph sampling approaches, the scalable distributed sampling approaches are as reliable as the traditional/non-distributed graph sampling techniques, and they bring much needed improvement to sampling efficiency, especially with regards to topology-based sampling. This benchmark study in traditional/non-distributed graph sampling is also applicable to distributed graph sampling as well.

A contribution to the area of graph visualization is also made through the presentation of a scalable graph visualization system-BGS (Big Graph Surfer) that creates hierarchical structure from an original graph and provides interactive navigation along the hierarchy by expanding or collapsing clusters when visualizing large-scale graphs. A distributed computing framework-Spark provides the backend for BGS on clustering and visualization. This architecture makes it capable of visualizing a graph up to 1 billion nodes or edges in real-time. In addition, BGS provides a series of hierarchy and graph exploration methods, such as hierarchy view, hierarchy navigation, hierarchy search, graph view, graph navigation, graph search, and other useful interactions. These functionalities facilitate the exploration of very large-scale graphs. Evaluation of BGS is performed through application to several representative of large-scale graph datasets and comparison with other existing graph visualization tools in scalability, usability, and flexibility.

The dissertation concludes with a summarization of the contributions and their improvement on large-scale graph analysis and visualization, and a discussion about possible future work on this research field.

Key words: large-scale graph, graph sampling, graph property, distributed graph sampling methods, sampling methods evaluation, graph visualization, graph clustering, graph hierarchy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Graph, as a standard data model, is widely used in various application domains to represent entities and their relationships, such as social network [70, 33, 87], Internet network [71, 25], citation network [39, 16], and biological network [40, 82], which define the relation between two objects. In mathematics, graphs are used to represent the pairwise relation between nodes. All the nodes and links comprise a whole graph. Real world systems use the concept to show relations.

Figure 1.1: Zachary's karate club network

For example, Figure 1.1 is a simple Zachary's karate club network [59] where vertices represent members of a karate club and links between two vertices denote interactions outside the club. The club network is an example of how the graph can intuitively represent relations in application.

As graph size grows, however, a lot of questions arise in graph research field, such as, how can we abstract useful information from the graph? How can we effectively explore structures or topology of the graph? How can we detect important objects or connections within the graph? How can we reveal the information of one vertex and its neighbors promptly? How can we obtain a good representative from the graph and infer useful information from the representative? Those problems become extremely difficult to be addressed with the increase of graph size.

To cope with the above-mentioned problems, graph analysis and visualization has evolved into a very active area of research over the last several decades. This thesis focuses on exploring large-scale graphs through graph sampling and visualization in order to comprehend large-scale graphs well. Graph sampling aims to reduce the complexity of graph drawing while preserving properties of the original graph, allowing analysis of the small sample to yield the characteristics similar to those of the original graph. Graph visualization is an effective and intuitive approach to observe structures within graph data. For large-scale graphs, graph sampling and visualization are straightforward strategies to gain insights into common issues that we have encountered. Specifically, We evaluate commonly used graph sampling methods through a combined visual and statistical comparison of graphs sampled at various rates to uncover their preference in preserving graph

properties. This study offers us a guideline in choosing the appropriate method for a particular graph sampling task. In addition, We develop a number of distributed graph sampling methods which are very suitable for usage in obtaining a representative from large-scale graphs. Finally, We design and develop a scalable graph visualization system-BGS (Big Graph Surfer) that allows clustering, hierarchy generation, interactive navigation on a large-scale graph. This distributed architecture of the system realizes linear scalability on the number of machines in large-scale graph visualization.

1.1.1 Graph Sampling

In the past few years, graphs have been growing explosively in numerous research fields. It was reported that Facebook had 1.6 billion active users during a month in late 2015 [104]. There are 3 billion emails created every day worldwide [6].

Effectively understanding huge graphs becomes more and more difficult with the increase of graph size. It is often imperative to sample a small subgraph from a large-scale graph. This is necessary for several reasons. First, sometimes visualizing a large-scale graph seems to be impossible because of limits of screen size. A good small representative subgraph can help us understand the topological structure of the original graph. The findings from the subgraph allow us to make more accurate estimations about graph features on the original graph. Second, graph property calculation is an essential approach to analyze graph features, but the calculation of graph properties on a large-scale graph can be costly or unaffordable. If an accurate estimation graph properties can be made from a representative of original graph, then substantial computational cost may be avoided. Third,

for some large-scale graphs, it is not practical to obtain complete datasets because of graph size or other various reasons. For example, total Facebook graph has more than 250TB of HTML data [41]. It is impractical to obtain complete data considering the time-cost.

For the above reasons, sampling algorithms aims to reduce the complexity of graph drawing while preserving properties of the original graph, allowing analysis of the small sample to yield the characteristics similar to those of the original graph.

Given a graph $G = (V, E)$, V and E represent vertices and edges respectively. Sampling techniques try to get a subgraph $Gs = (V_s, E_s)$, where $V_s \subset V$, $E_s \subset E$. Prior sampling approaches explore networks from two aspects [6]: making accurate estimations of the original graph [9, 93]; selecting a representative sample whose topology is similar to the original [56].

Many sampling techniques have been proposed aiming to sample a representative subgraph from the original graph[55, 66, 28], but there has not been a systematic empirical comparison of existing methods. Practical questions often arise regarding which sampling method one should use for a particular application. To answer these questions, we need to identify graph properties and metrics that facilitate a fair and conclusive comparison of sampling approaches. In turn, we need to use these metrics to ascertain which sampling methods are most suitable for estimating specific graph properties. This study can be used as a guideline in choosing the appropriate method for a particular graph sampling task.

1.1.2 Distributed Graph Sampling Methods

In previous research, many sampling techniques have been proposed aiming to sample a representative subgraph from the original graph. Those sampling methods can be classified in three categories: node sampling [66, 8], edge sampling [8, 32] and topology-based sampling [63, 84, 99]. Unfortunately, these traditional/non-distributed sampling methods, especially topology-based sampling, are not practical for large-scale graphs (more than 1 million vertices) because of efficiency issue or memory issue. For example, sampling on Email graph (vertices: 265, 214; edges: 420, 045) takes more than 24 hours while using topology-based sampling. Most prior sampling research only worked on graphs less than 1 million vertices. With the increase of graph size, scalable sampling methods are in urgent demand in graph research.

To solve the above issue, we design and develop a number of scalable sampling methods (called distributed graph sampling methods) based on a MapReduce framework-Spark, and make comprehensive evaluation by comparing traditional/non-distributed sampling approaches with distributed graph sampling methods in visual and statistical perspectives. This study provides us a channel to sample large-scale graphs efficiently without losing accuracy.

1.1.3 Large-Scale Graph Visualization

Graph visualization is an intuitive and fundamental technique to understand relations within graph data. Until now, many visualization techniques and systems have been developed in a variety of domains. However, as graphs grow exponentially in size, we find

existing visualization systems have more and more difficulty visualizing such large-scale graphs in application.

When visualizing large-scale graphs, there are several fundamental issues that impair graph visualization. Also, all the issues are getting worse and worse with the increase of graph size. Those issues are stated as follows.

Memory. Large-scale graph datasets stored on disk may be hundreds of Gigabytes or even larger, for example, Friendster graph has around 30 Gigabytes [67]. To visualize such large graph, the first step is to load it into main memory. However, it is challenging to do this job because of the limits of RAM capacity in single machine. Even though external memory algorithms can be used to manipulate the graph, the substantial difference between CPU speed and external memory makes the operations very ineffective [1].

Display. The second issue is caused by screen size. As we know, the amount of information that screen can display is dependent upon its number of pixels in 2D visualization. Although an entire large-scale graph can be visualized, it is difficult for us to discern vertices, edges, and internal structure because visual clutters would exist in visualization. Layout [72, 91] can increase the scalability of graph visualization to some degree, but there still exists an upper bound for graph size. Thus, the size limit of the display should be considered in graph visualization.

Layout. Layout techniques investigate how to arrange vertices and edges in aesthetic criteria. For example, force-directed algorithm, a classic layout algorithm, finalizes the vertices position by iteratively calculating repulsive forces between all pairs of vertices and attractive force between connected vertices [73, 53]. Unfortunately, the computation

of vertices position is very expensive for layout algorithms. The complexity of force-directed layout algorithm is $O(n^3)$. As graph size increases, the computational challenge for layout calculation becomes more and more serious. It also under-mines interaction in graph visualization.

Interaction. Interaction is regarded as a further step to analyze graph data through an array of operations while visual representations alone cannot convey information clearly. Those operations include zooming in/out on graphs, navigating on graphs, dragging and moving of vertices, highlighting vertices, expanding or collapsing clusters, etc. When a large-scale graph is too large to fit on the screen, interactions are important and necessary to explore the graph. Since most interactions are involved in layout calculation, how to interact with a large-scale graph smoothly in visualization is also a big issue.

To solve these issues, we propose a new graph visualization system called BGS (Big Graph Surfer) intended to visualize very large-scale graphs by generating hierarchical structure from original graphs and providing a series of graph exploration methods, such as hierarchy view, hierarchy navigation, hierarchy search, graph view, graph navigation, graph search, and other useful interactions. The fundamental goal of BGS is to visualize very large-scale graphs and interact with such graphs smoothly.

1.2 Objective

The research question that we address in the thesis is as follows:

How can we help users gain insights from large-scale graph with billions of nodes or edges using graph visual analytics?

We approach the research question from graph sampling and graph visualization. These techniques involved in both topics provide several feasible means to explore and analyze large-scale graphs. For graph sampling, an evaluation is conducted on existing prevalent sampling methods via a combined visual and statistical comparison on various graph datasets, which provide a basis for the development of distributed graph sampling methods. We develop a number of distributed sampling methods which allow us to sample large-scale graphs effectively according to some requirements. For graph visualization, We develop a visualization system-BGS which is not designed to answer all graph questions through the visualization, but instead explore large-scale graphs visually in several aspects. First, We try to design a scalable architecture which is able to handle billion-scale graphs. Second, we opt for the significant ways of interacting in graph visualization. All the interactions I select are particularly suitable for large-scale graphs. Afterwards, to evaluate effectiveness of the visualization system, We perform several case studies on a few graph datasets. Finally, we provide some guidelines for explore large-scale graphs when using BGS.

1.3 Contributions

In this dissertation, We present several techniques to tackle the research question. In graph sampling, the thesis evaluates commonly used graph sampling methods to reveal their preference in preserving graph properties, and develop a number of distributed graph sampling methods which fit for very large-scale graphs. These contributions are as follows:

1. A benchmark for evaluating graph sampling methods is built with both visual and statistical properties, which provides a basis for analyzing distributed graph sampling methods.

2. A large-scale graph sampling package including nine scalable graph sampling methods is developed based on Spark, which can be incorporated into GraphX for graph research.
3. A visual and statistical comparison is made between traditional/non-distributed sampling methods and distributed graph sampling methods.
4. Nine distributed graph sampling methods are analyzed from visual and statistical perspective, which provides a guideline in choosing the appropriate method for graph exploration.

In graph visualization, We propose a scalable graph visualization tool that allows generating hierarchical structure by clustering and interactive navigation of large-scale graphs when visualizing large-scale graphs. These contributions are as follows:

1. A scalable graph visualization system-BGS is presented, which can visualize large-scale graphs with billion-scale vertices or edges.
2. An efficient hierarchy construction algorithm based on a tuned Louvain clustering is proposed, which can be easily implemented in distributed computing system.
3. Two well-designed view modes, two hierarchy and graph expansion modes, and two visualization modes provided in BGS allow us to explore hierarchy and graph based on users' needs and visualization efficiency.
4. BGS supports direct search on hierarchy view and graph view by vertices attribute(s) or edges attribute(s), which helps users identify interesting vertices or edges promptly.

1.4 Organization

The above topics are effective approaches to explore large-scale graphs. I will introduce each topic in following sections.

Chapter 2 presents an evaluation on commonly used graph sampling methods through a combined visual and statistical comparison of graphs sampled at various rates. The evaluation is conducted on three graph models: random graphs, small-world graphs, and scale-free graphs. This benchmark study finds out how a sampling method is dependent on the graph model, the size of the graph, and the desired statistical property.

Chapter 3 presents three types of distributed sampling algorithms and develops a sampling package on Spark. To evaluate the effectiveness of these distributed sampling techniques, they are applied to three graph datasets and compare them with traditional sampling approaches in reliability, efficiency, and scalability.

Chapter 4 presents scalable graph visualization system-BGS that allows generating hierarchy and interactive navigation along the hierarchy. BGS provides a series of graph exploration methods that brings us one approach with greater convenience to analyze very large-scale graphs. The case studies in this chapter help us identify BGSs scalability, usability, and flexibility compared with existing graph visualization tools.

Finally, chapter 5 concludes the thesis with a talk about interesting future work on large-scale graphs.

CHAPTER 2

GRAPH SAMPLING FOR VISUAL ANALYTICS

2.1 Introduction

Graph sampling is necessary in large-scale graph research for various reasons, such as, graph visualization, graph property calculation, graph data acquisition etc. While numerous graph sampling techniques have been proposed, there is still a lack of a systematic empirical comparisons of existing methods. Users often get confused while choosing a sampling method in a particular application. To solve the problem, a study on a fair and conclusive comparisons of sampling approaches should be made to identify their statistical and visual preference, and ascertain which sampling methods are most suitable for estimating specific graph properties. To reflect the diversity of real-world graphs in this study, we choose three commonly seen graph types: random graphs, small-world graphs, and scale-free graphs.

Although these three graph models are widely discussed in graph research, many real-world graphs are too complex to be sufficiently modeled by any current research approaches. We designed a benchmark for comparing sampling methods for artificial random graphs, artificial small-world graphs, artificial scale-free graphs, and real-world graphs. Our comparison considers two complementary aspects: 1) how effectively the method preserves the graphs visual properties and 2) how well it preserves the graphs statisti-

cal properties. We conducted our study on directed and undirected graphs separately and used a number of statistical properties for comparison. To properly compare graph sampling methods for visualization, we fixed the graph layout in both the original and sampled graphs. The visual and statistical comparison provided criteria for selecting sampling methods in application.

The main contributions of our work are as follows:

- We implemented twelve graph sampling techniques in the benchmark
- We built a benchmark for evaluating graph sampling methods with both visual and statistical properties
- We studied a number of graph data sets with the benchmark and analyzed the results

2.2 Related Work

Existing graph sampling algorithms can be classified into three types: node sampling, edge sampling, and traversal-based sampling [55, 8, 66, 7]. Node sampling constructs subgraphs based on sampling vertices, often uniformly. In some cases, node sampling methods integrate traversal-based sampling in order to use graph topology information, such as random walk sampling. The metropolis algorithm [56] is a modified version of node sampling. It replaces some sampled vertices with other vertices, which often leads to sampled graph properties that are consistent with the original. Similarly, edge sampling builds a subgraph by randomly sampling edges. Traversal-based sampling creates subgraphs based on the topological information from the original graph. These methods do not sample vertices or edges directly but instead select vertices using traversal-based algorithms. Breadth-first

[63], random walk [32, 107], and snowball sampling [32] are commonly used traversal-based sampling algorithms that select vertices based on the topological information of the graph.

One purpose of sampling is to simplify the graph for better visualization. With millions or billions of vertices or edges, it is challenging to clearly visualize all of them. Even when the entire graph can be displayed, graph visibility and usability are issues [96]. Numerous techniques have been proposed to approach graph visualization, such as clustering [96, 86, 49, 110], sampling [83, 66], and special layout [29, 29, 85, 17]. These techniques aim to reduce the overlap between vertices and edges. Sampling approaches improve visualization by sampling the original graph, resulting in fewer vertices and edges. Layout techniques explore vertex and edge arrangements when displaying graphs. Many layout techniques have been proposed, such as Tree layout [98, 88], 3D layout [31, 22], hyperbolic layout [74], and force-directed layout [37]. Clustering reduces vertex and edge overlap by replacing clusters with vertices. Two graph clustering techniques are vertex clustering [103, 75] and edge clustering [72, 27, 54].

2.3 Visual and Statistical Benchmark

To build visual and statistical benchmark for sampling methods, several graph models and their typical degree distribution will be introduced, and additionally eight undirected graph properties and nine directed graph properties will be presented. These properties are used for comparing twelve widely used sampling methods. Those sampling methods

are also discussed in this section. Finally, we will talk about eight datasets used in our experiment.

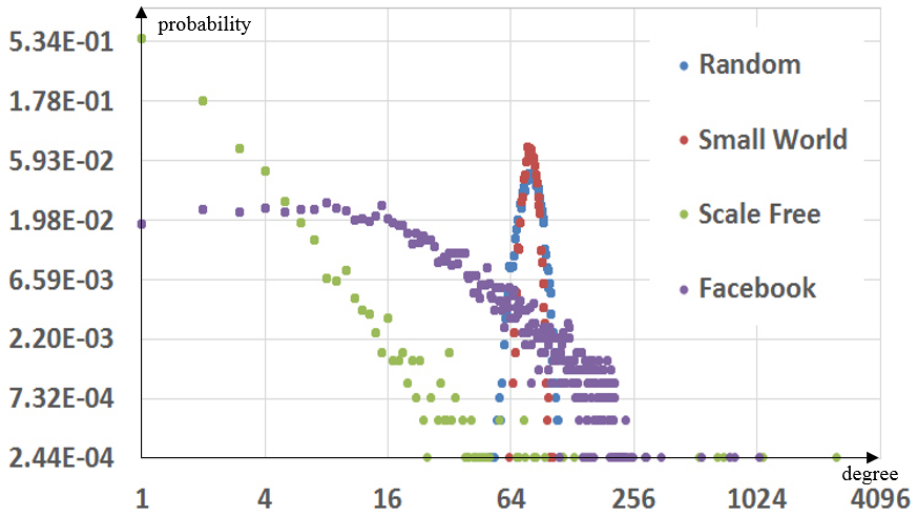
2.3.1 Graph Models

We hypothesize that graph type is one of the main factors affecting sampling methods performance. Thus, we simulate three types of graphs in this study. The key motivation is to develop graph models that fit many real-world graphs. Towards this end, we use the random graph model, the small-world graph model, and the scale-free graph model as well as real-world graphs.

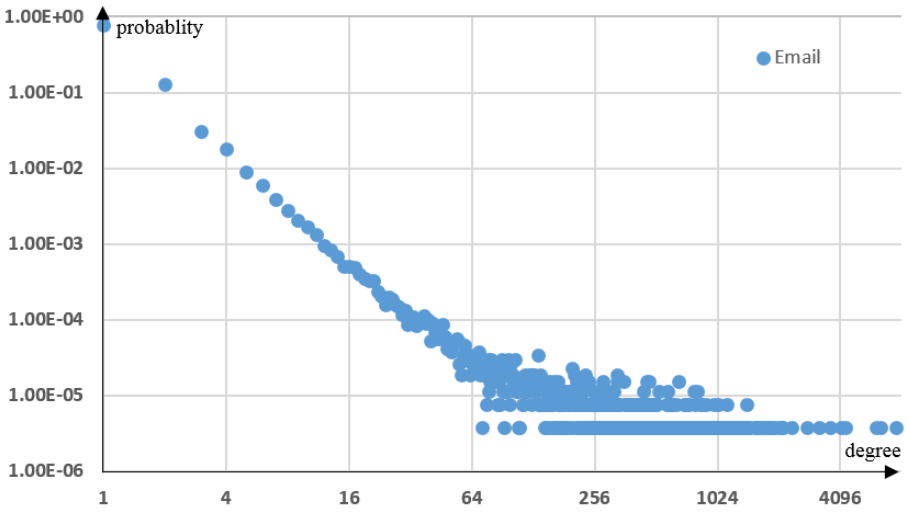
The degree distributions of the three graph models and a real social network graph are illustrated in Figure 2.1a. The degree distribution of an email graph (an email graph is a type of graph data we used in the chapter; see details in Graph Datasets) is shown in Figure 2.1b. They are drawn separately because of their different scales in degree distribution. We find that the real social network graph is a complex graph that is different from any theoretical models, although it shows some similarities to a small-world graph. We apply sampling methods to the three graph models and the real-world graph. The details of the data are further discussed later in the Graph Datasets section.

2.3.2 Graph Properties

We use eight graph properties for comparing undirected graphs and nine graph properties for comparing directed graphs. For undirected graphs, we use the degree distribution (DD), average neighbor degree distribution (ANDD), degree centrality distribution (DCD), node betweenness centrality distribution (NBCD), edge betweenness centrality distribution



(a) Degree distribution of random graph model (blue), small-world graph model (red), scale-free graph model (green), and real social graph (magenta) with Log 2 axis.



(b) Degree distribution of email graph (email graph is a type of graph data we used in the chapter. See details in Graph Datasets)

Figure 2.1: Degree distributions of five graph models

(EBCD), local clustering coefficient distribution (LCCD), closeness centrality (CCD), and eigenvector centrality distribution (EVCD). For directed graphs, we use in-degree distribution (InDD), out-degree distribution (OutDD), in degree centrality distribution (InCD), out degree centrality distribution (OutCD), ANDD, NBCD, EBCD, CCD, and EVCD.

Degree distribution. A vertexs degree is the number of edges connected to that vertex. The degree distribution is the probability distribution of a vertexs degree. For directed graphs, because a vertex has incoming and outgoing edges, the graph has in degree distribution and out degree distribution.

Degree centrality. Degree centrality describes the importance of vertices by using the degree metric of the graph. The degree centrality for a vertex v is the fraction of vertices it is connected to. Given a graph $G = (V, E)$, n vertices, for a vertex v, degree centrality can be represented as in Ref. [46]:

$$C_{(d)}(v) = \frac{degree(v)}{n-1}$$

Average neighbor degree. This property returns the average degree of the neighborhood of each vertex. It is represented as follows [14]:

$$A_{(d)}(v) = \sum_{i=1}^{n(v)} \frac{d_{(t)}}{n(v)}$$

where $n(v)$ are the numbers of neighbors of vertex v and $d(i)$ is the degree of vertex i which is connected to vertex v.

Betweenness centrality [21, 4]. Betweenness centrality indicates the probability of the vertex acting as a bridge along the shortest path between two other vertices. Betweenness has two categories: vertex betweenness centrality and edge betweenness centrality. Vertex

betweenness centrality is an indicator that shows a vertexs centrality in a graph, which refers to how many shortest paths from all vertices to all others pass through that vertex [21]. If a vertex has a high probability to be chosen as a bridge in shortest path between two other vertices, then it would have a high betweenness. Betweenness centrality of a vertex is defined as:

$$C_{(B)}(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} represents the number of shortest geodesic paths from s to t, and σ_{st} is the number of shortest paths from s to t via v.

Likewise, edge betweenness centrality is the probability of an edge acting as a bridge between two other edges in their shortest path. Edge betweenness centrality is defined in the formula:

$$C_{(B)}(e) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{(s,t)}(v)}{\sigma_{(s,t)}}$$

where $\sigma_{(s,t)}$ represents the number of shortest geodesic paths from s to t, and $\sigma_{(s,t)}(e)$ is the number of shortest paths from s to t through e.

Clustering coefficient. Clustering coefficient is a measure of how vertices in a graph cluster together. It includes the global clustering coefficient and the local clustering coefficient. The former, which is a graph property, shows the overall indication of the clustering in the graph. The latter is a vertex property indicating the embeddedness of the vertex.

Global clustering coefficient is defined based on triangles. This coefficient measures the clustering in the whole graph. The global clustering coefficient can be represented as in Ref. [76]:

$$C_g = \frac{3 \times N_{triangles}}{N_{triplets}}$$

$N_{triangles}$ represents the number of triangles in the graph. $N_{triplets}$ stands for the number of connected triplets of vertices. A triplet consists of three graph vertices. If two of them are connected, then the triplet is called an "open triplet". If three of them are connected, then it is called a "closed triplet". Local clustering coefficient is a metric to evaluate how close a vertexs neighbors are to each other. If kv denotes the number of neighbors of v, and e_v is the number of connected pairs between all neighbors of v, then the local clustering coefficient can be defined in undirected graphs as in Ref. [101]:

$$C_v = \frac{2e_v}{k_v(k_v - 1)}$$

in directed graphs as:

$$C_v = \frac{e_v}{k_v(k_v - 1)}$$

Closeness centrality [78]. Closeness centrality describes how central the vertex is in the graph. It is defined as the reciprocal of the sum of the distances from all other vertices that the vertex is connected to.

$$C(v) = \frac{1}{\sum_{t \in V} dist(v, t)}$$

Eigenvector centrality [64]. Eigenvector centrality is a measure of the weight of a vertex in a graph. Each vertex is assigned a value based on the concept that connections to

high-scoring vertices contribute more to the score of the vertex than equal connections to low-scoring vertices.

If A is the adjacency matrix of the graph and x is an initial eigenvector, then there will be a positive solution to the following equation:

$$Ax = \lambda x$$

There will be a positive solution with the final eigenvector from the formula after using a power method based on PerronFrobenius theorem [80]. λ is also the largest eigenvalue associated with the eigenvector of the adjacency matrix.

In this chapter, we use Graph-tool [79] to analyze the above graph properties because of its fast speed.

2.3.3 Graph Sampling Methods

Within the benchmark, we implement twelve widely used sampling methods. These sampling methods include random node (RN), random node-edge (RNE), random node-neighbor (RNN), random edge (RE), induced edge (IE), breadth-first (BF), depth-first (DF), random first (RF), snowball (SB), random walk (RW), random walk with escape (RWE), and forest fire (FF) sampling. For all these sampling methods, sampling rates are defined as the ratio between the edges after the sampling and before the sampling.

Node Sampling

- ***Random node sampling*** [8, 7]. Vertices are sampled randomly and uniformly, creating a subgraph of the original graph. Those edges that are connected between those sampled vertices in the original graph enter the sample graph.

- ***Random node-edge sampling*** [55, 7]. This method is based on random node sampling. After vertices are uniformly sampled, edges that are incident to these vertices are uniformly included in the sample graph.

- ***Random node-neighbor sampling*** [66]. This sampling method is similar to random node-edge sampling. Vertices are uniformly random sampled first, but all the edges connected to these vertices in the original graph are sampled into subgraph.

Edge Sampling

- ***Random edge sampling*** [66, 32]. Edges are sampled randomly and uniformly, and then a subgraph is created from those edges.

- ***Induced edge sampling*** [8]. Induced edge sampling includes totally induced edge sampling and partially induced edge sampling. Totally induced edge sampling has two steps. First, it conducts random edge sampling and obtains adjacent vertices from these edges. Second, all edges attached to those vertices are sampled in a subgraph. Partially induced edge sampling performs edge sampling in a single pass in which edges are selected with a probability. Incident vertices are also added to the sampled graph if one edge is selected. In this work, we implement totally induced edge sampling.

Traversal-based Sampling

- ***Breadth-first sampling*** [55, 63]. This sampling method is induced from the graph traversal algorithm breadth-first search. It begins with a random vertex and visits its

neighbors iteratively. For each iteration, the first visited vertex will enter the sample first. A subgraph is created from sampled vertices and those edges that are connected between those sampled vertices in the original graph.

- ***Depth-first sampling*** [92]. This approach derives from the depth-first search algorithm. For each iteration, the first visited vertex will enter the sample last.

- ***Random-first sampling*** [55]. This algorithm is similar to breadth-first sampling and depth-first sampling except that vertices are selected randomly in each iteration.

- ***Snowball sampling*** [32, 18]. This sampling method first picks up a starting vertex at random and puts it in the current vertex set, and then all vertices that are connected to any vertex in the current vertex set are chosen and put into the current vertex set recursively until the required number of vertices is selected.

- ***Random walk sampling*** [92, 42]. This method starts at a seed vertex, and then chooses a vertex uniformly at random from the neighbors of the current vertex. A subgraph is created from the walking paths. Random walk with escape or jump and multiple independent random walkers are proposed based on the classic random walk sampling method.

- ***Random walk with escape or jump sampling*** [66, 84]. This sampling method is the same as random walk except that the current walker vertex jumps to another random vertex with probability p.

- ***Forest fire sampling*** [8]. This sampling approach can be regarded as a probabilistic version of breadth-first sampling. Neighbors are chosen to be added to the subgraph with probability p. The number of vertices to be chosen is a random number taken from a geometric distribution with mean $p_f/(1-p_f)$. (p_f is set to 0.5 in this work.)

We apply these sampling methods to four types of graphs: scale-free graph, random graph, small-world graph, and real-world graphs. These graphs can be undirected or directed.

2.3.4 Graph Datasets

The datasets we used in the chapter are collected from several data sources. The social graphs, citation graphs, email communication graphs, and internet graphs are downloaded from Stanford Network Analysis Platform (SNAP). The small-world and random graphs are created from NetworkX [45] via corresponding graph models.

In NetworkX, a random graph is created using the Erdos-Renyi graph model, introduced by Paul Erdos and Alfred Renyi in 1959 [36]. Vertices in the graph are connected randomly, and each edge exists in the graph with probability p. If a random graph has n vertices, and the edges probability is p, then the probability of a vertex with degree k is

$$P(k) = \binom{n-1}{k} p^k (1-p)^{n-k-1}$$

where n is the number of vertices in graph. The degree distribution of the graph follows a Poisson distribution. Random graph creation [36] is straightforward and simple. Given a random graph with n vertices, an edge existing between any pair of vertices has the probability p, independent of the existing edges in the graph.

Table 2.1: Eight test graph datasets and their properties

Graph Dataset	Graph Type	Model	# Vertices	# Edges
Random	Directed	Model	10,000	100,246
Small-World	Undirected	Model	10,000	21,895
Scale-Free	Directed	Model	10,000	18,838
Email	Directed	Real	265,214	420,045
Citation	Directed	Real	34,546	421,578
Internet	Directed	Real	10,876	39,994
Facebook	Undirected	Real	4,039	88,234
U.S. Flight	Undirected	Real	235	1,297

Small-world graphs are generated using small-world graph models presented by Watts and Strogatz [101]. Small average shortest path length and high clustering coefficient characterize the graph model. The typical distance L between any two vertices is proportional to logarithm of number of graph vertices N.

$$L \propto LogN$$

The generation process of a small-world graph can be described as follows [101]. The initial graph is a ring with n vertices, and each vertex has k edges connected to its nearest neighbors. This is a regular graph in which each vertex has the same number of connection to other vertices. After the regular graph is generated, each edge is rewired to another uniformly vertex with probability p. If p equals 0, then the graph is a one-dimensional

lattice graph or a regular graph. If p is 1, then the graph will become a random graph. From the generation of a small-world graph, we also find that the rewiring process reduces average path length, but the clustering coefficient is generally as high as in regular graphs.

Scale-free graphs are constructed using the scale-free graph model proposed by Baraasi and Albert [51]. In a scale-free graph, vertices asymptotically follow a power-law on degree distribution:

$$P(k) \approx \frac{1}{k^r}(2 < r < 3)$$

where $p(k)$ is the fraction of degree k in graph. r is a constant value usually ranging from 2 to 3.

The generation process of scale-free graph starts from n vertices and no edge connecting them. New vertices and edges are added to the graph in each step. The probability of the new vertex connecting to existing vertices is proportional to their degrees in the graph:

$$P_i = \frac{k_i}{\sum_{j=1}^{n} k_j}$$

where P_i is the probability of vertex i with degree k_i to connect to a new vertex. n is the existing number of vertices. The preferential bias to high degree of vertices is termed preferential attachment. The growth of a graph based on preferential attachment results in a power-law degree distribution. Table 2.1 summarizes the properties of the eight datasets used for the comparison study.

2.3.5 Statistical Comparison

From those graph properties mentioned above, we can obtain graph property distributions of vertices or edges. We evaluate the sampling techniques based on the comparison

of the graph property distributions between sampling methods. A good sampling method should produce a sampled graph with sampling results that approximate the original graph. That is, the probability distributions of the properties of the two graphs should have a short distance between them. Here we use skew divergence (SD) to evaluate the difference between two distributions [65]. Generally, skew divergence is used to measure Kullback-Leibler (KL) divergence between two probability density distributions that do not have continuous support over the range of values. Because graph properties distributions are not continuous, e.g. clustering coefficient, the two probability density distribution should be smoothed before computing KL divergence. We use the same strategy as the Ref. [65, 6] to smooth the distributions:

$$SD(P, Q, \alpha) = KL[\alpha P + (1 - \alpha)Q||\alpha Q + (1 - \alpha)P]$$

To better compare the sampling results, we use the average SD defined in the work, and α is set to 0.99 as in the Ref. [65, 6]. Previous work [65] has proven that SD has better performance to approximate KL divergence on non-smoothed distributions.

In our experiment, the above eight datasets are used in statistical comparison. To compare computing time between sampling methods, we record the execution time for each method.

2.3.6 Visual Comparison

We use Gephi [15] to visually compare sampling methods. Gephi is a well-designed graph visualization tool with many features including, 3D render engine able to display graphs in real-time as well as graph exploration, analysis, and manipulation.The original

graph is drawn first, and vertices are decorated for the sake of comparison between sampling results with the original graph. Then the decorated graph is exported into a file with vertex decorations preserved. When sampling on the decorated graph, vertex attributes are also sampled along with vertices. The layout in the sampling results should have the same layout as the decorated graphi.e., the same vertex in all sampled graphs will occupy the same location as in the decorated graph. Also, the same vertex in all sampled graphs has the same color and label size as in the decorated graph.

The vertex attributes, for example, vertex position and color, are preserved in sampling because we consider that such vertex attributes are significant in visual comparison. In this way, the similarity and difference within or between sampling results can be easily identified.

We preserve vertex attributes instead of edge attributes for two reasons. First, edge attributes are not as important as vertex attributes in visual comparison. Second, the complexity of edge attributes persevering algorithm is $O(|E|)$, but the complexity of vertex attributes persevering is only $O(|V|)$. For a dense graph, $O(|E|) \approx O(|V_2|)$. The edge attributes persevering algorithm may increase sampling complexity.

In this work, there are two datasets of visual comparison provided: U.S. flight graph data and social graph data. For the flight graph, we use the geospatial layout, and for the social graph data, we use the force-directed layout.

2.4 Results and Analysis

2.4.1 Results

We apply the sampling approaches to the U.S. flight graph, social graph, citation graph, internet graph data, email communication graph, random graph, scale-free graph, and small-world graph data. In this work, we conduct experiments using a sampling rate ranging from 10 to 50 percent with a 10 percent interval on all eight graph datasets. For each sampling rate, we perform graph sampling 10 times, and take the average SD value for this sampling rate. All sampling rates are based on the number of edges. We use the average SD value as final results for analyzing each graph-metric. Here we do not list all individual results in this work.

The line charts in Figure 2.2 show the SD divergence between the sampling result and the original graph for each sampling method on statistical properties.

The vertical axis in the line chart is the SD value between the sampling result and the original graph ranging from 0 to 1. A smaller value indicates more consistency between the sampling results and the original graph. The horizontal axis lists the graph properties. Each line in the chart represents one sampling method. Its value indicates the sampling methods performance. From the benchmark, a user who is working on a particular type of graph can identify which sampling method performs the best for each graph property for that particular type of graph.

In addition, when conducting sampling on the graph dataset, we record the execution time for each sampling method. For example, the computing time for Facebook data is shown in Figure 2.3.

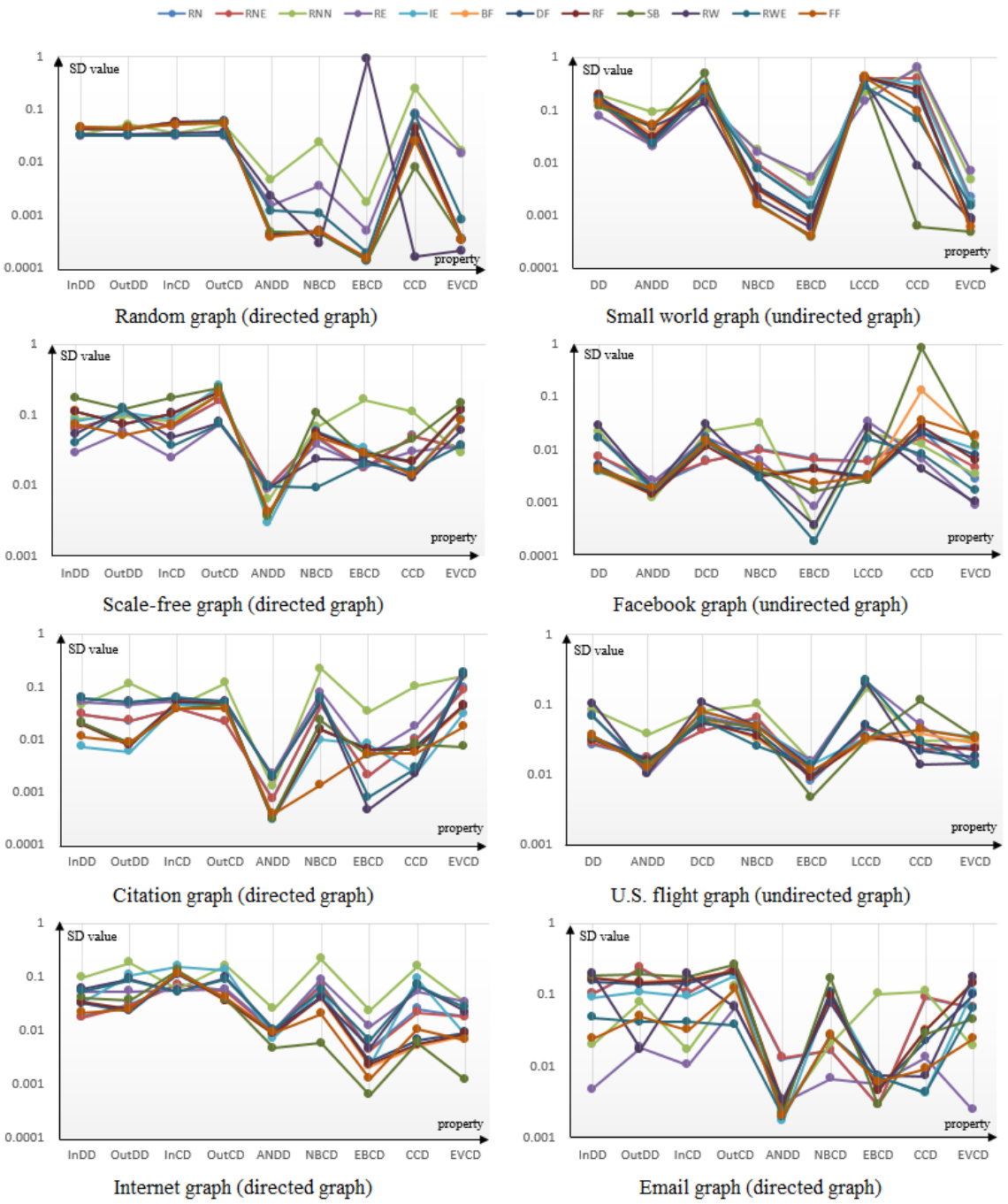


Figure 2.2: Average result of the statistical comparisons between sampling methods with 10 to 50 percent sampling rates. The vertical axis is SD values, horizontal axis represents graph properties, and lines are sampling methods.

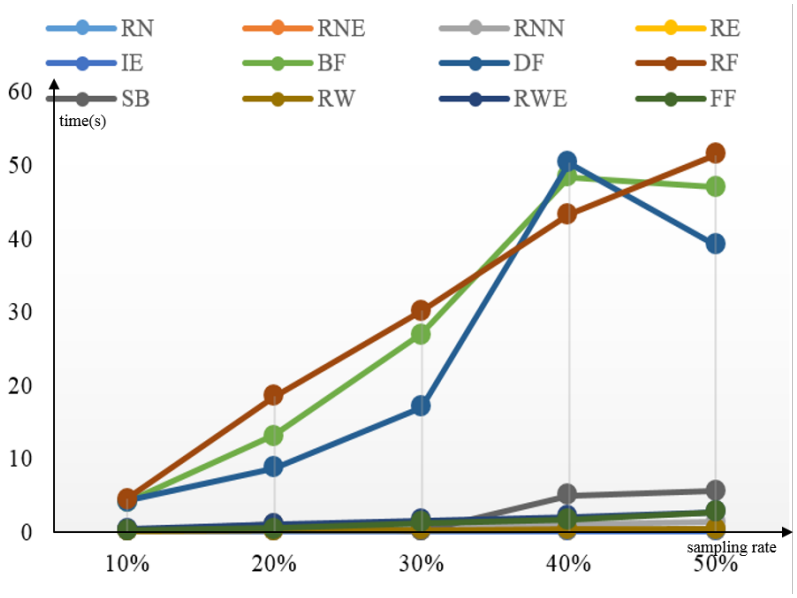
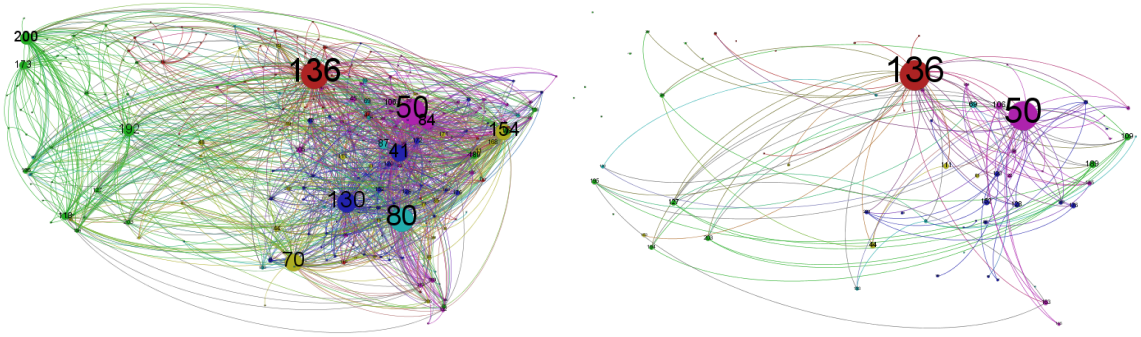
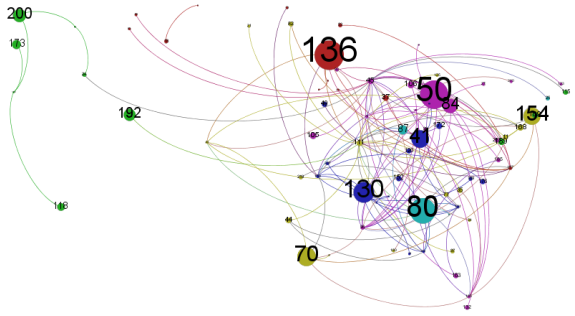


Figure 2.3: Execution time between sampled methods for Facebook graph data. Lines represent sampling methods. X axis represents sampling rate for each sampling method. Y axis represents execution time in seconds.

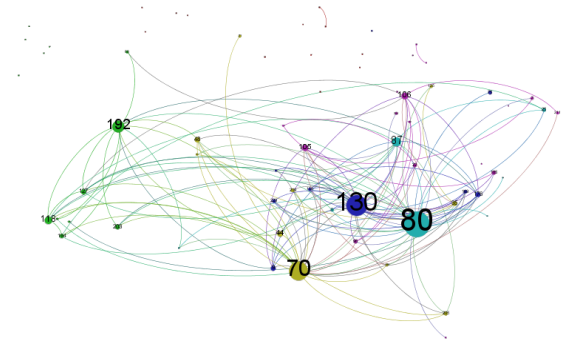


(a) Original (Nodes:235 Edges:1297)

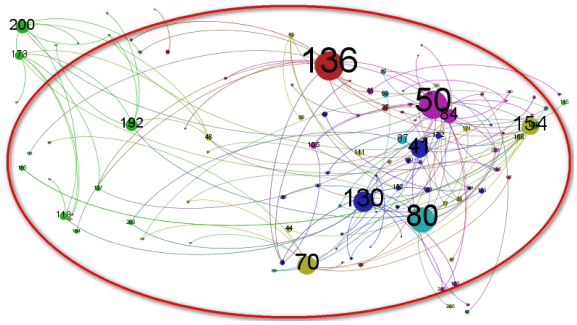
(b) RN (Nodes:68 Edges:127)



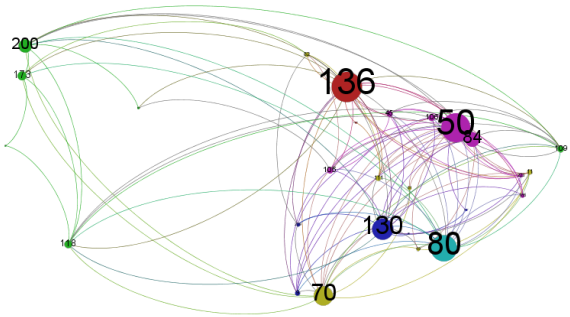
(c) RNN (Nodes:78 Edges:124)



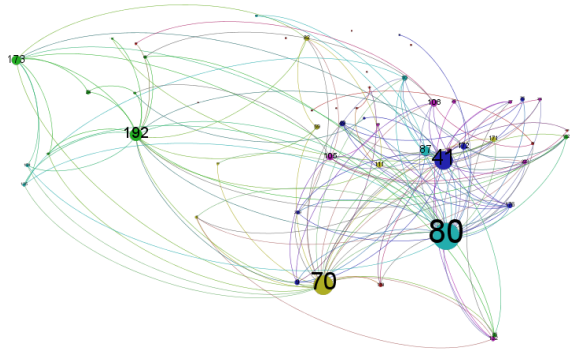
(d) RNE (Nodes:69 Edges:134)



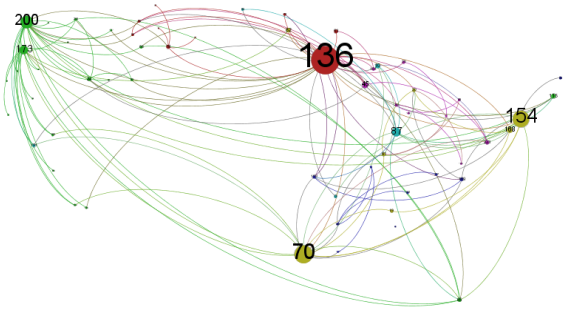
(e) RE (Nodes:110 Edges:129)



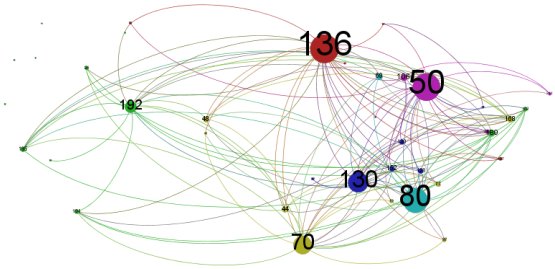
(f) IE (Nodes:27 Edges:125)



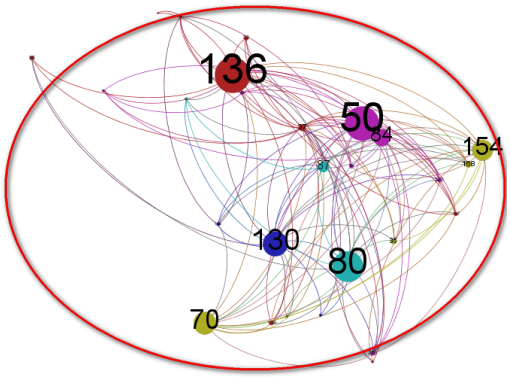
(g) BF (Nodes:53 Edges:146)



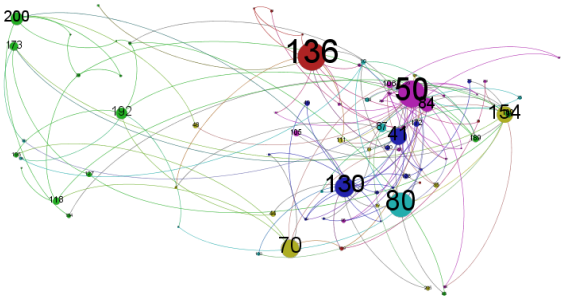
(h) DF (Nodes:60 Edges:131)



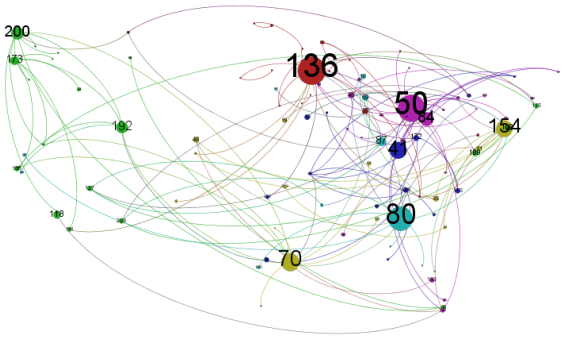
(i) RF (Nodes:38 Edges:132)



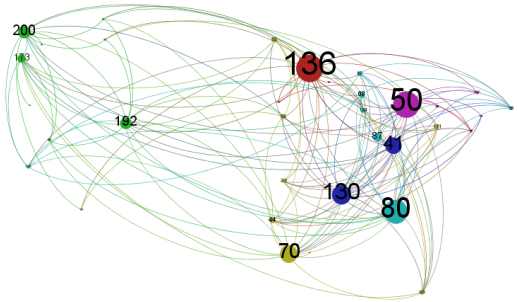
(j) SB (Nodes:27 Edges:133)



(k) RW (Nodes:81 Edges:120)

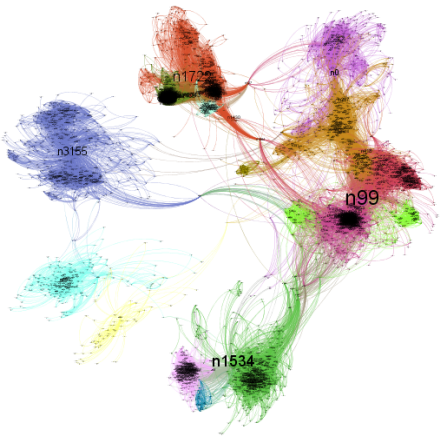


(l) RWE (Nodes:103 Edges:129)

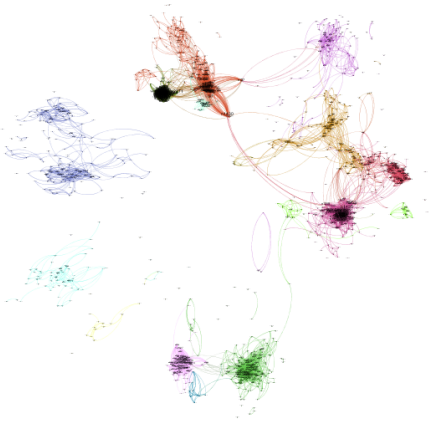


(m) FF (Nodes:35 Edges:158)

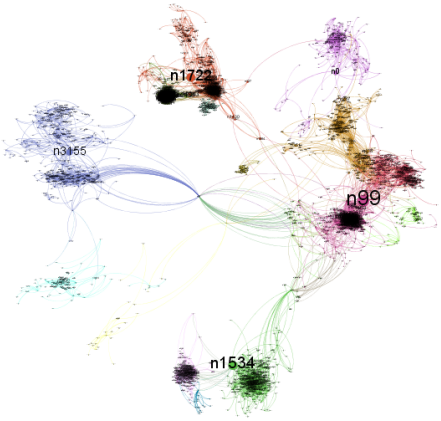
Figure 2.4: Visual comparison between sampling methods for U.S. flight data (undirected graph) with sampling rate 10% on edges. Red Circles in RE and SB show spatial coverage area of sampling results.



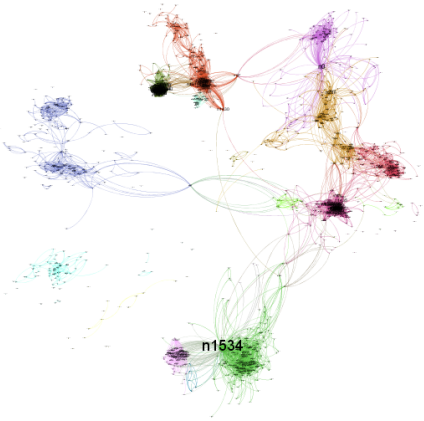
(a) RN (Nodes:4039 Edges:88234)



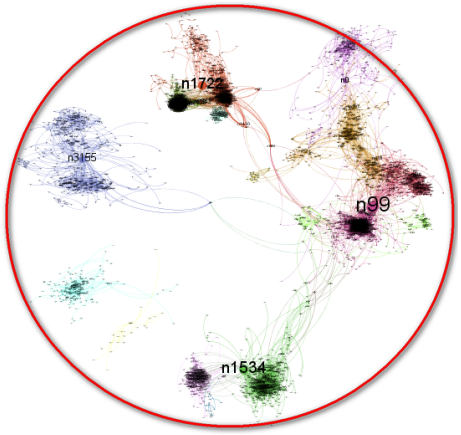
(b) RN (Nodes:1231 Edges:8841)



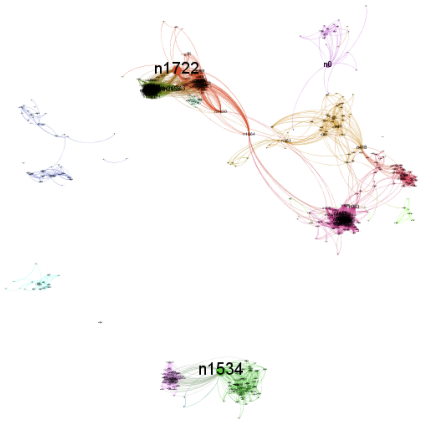
(c) RNN (Nodes:2859 Edges:8396)



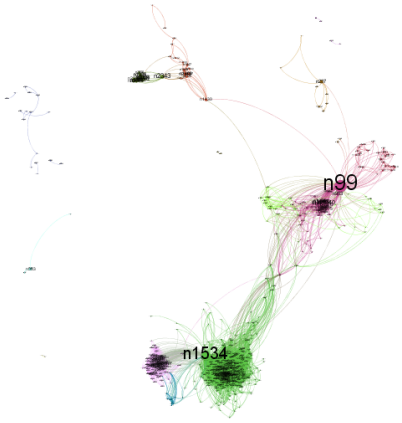
(d) RNE (Nodes:1252 Edges:8891)



(e) RE (Nodes:3289 Edges:8834)

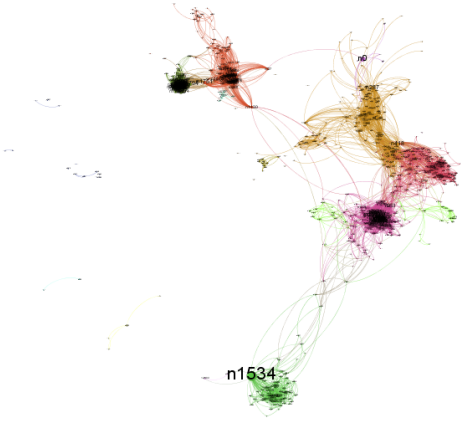


(f) IE (Nodes:583 Edges:8649)

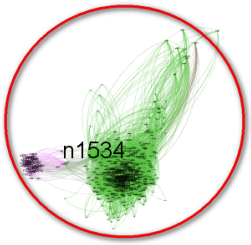


(g) BF (Nodes:684 Edges:8826)

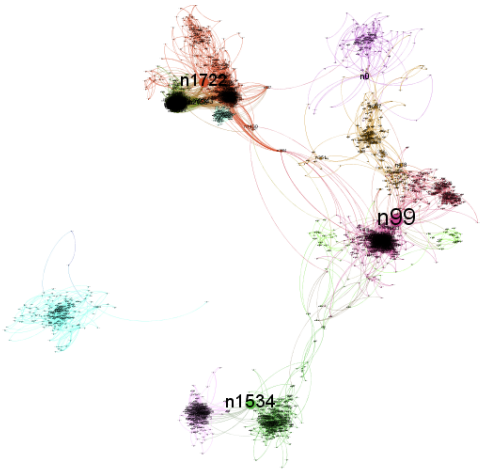
(h) DF (Nodes:749 Edges:8827)



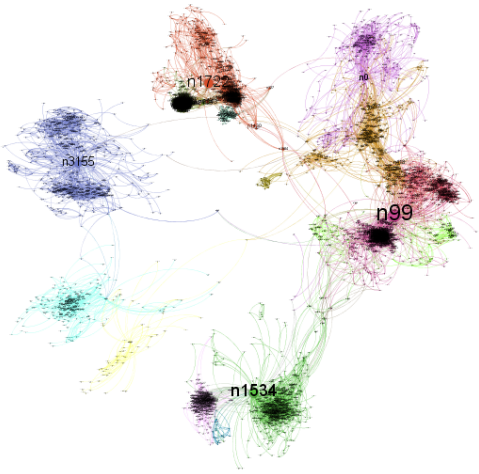
(i) RF (Nodes:882 Edges:8826)



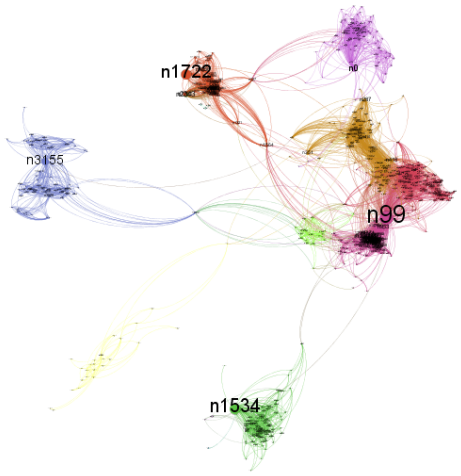
(j) SB (Nodes:491 Edges:8852)



(k) RW (Nodes:2055 Edges:8823)



(l) RWE (Nodes:3610 Edges:8823)



(m) FF (Nodes:744 Edges:8853)

Figure 2.5: Visual comparison between sampling methods for Facebook graph data (undirected graph) with sampling rate 10% on edges. Red Circles in RE and SB show spatial coverage area of sampling results.

Figure 2.4 shows the visual comparison between sampling methods using a 10% sampling results for the U.S. flight graph data. Figure 2.5 shows the same type of comparison on the Facebook graph data. In both visualizations, vertex label size is positively proportional to its degree.

2.4.2 Analysis

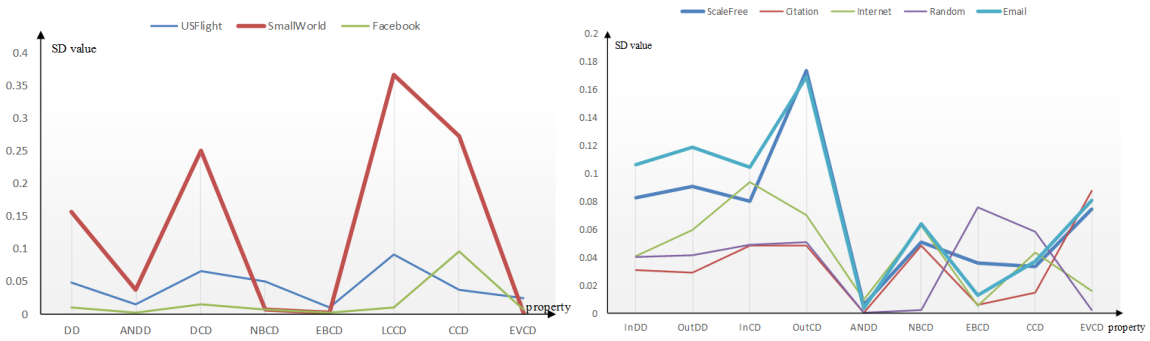
The benchmark allows us to compare sampling methods quantitatively and qualitatively in several aspects. First, the sampling experiment covers U.S. flight graph, social graph, citation graph, internet graph, email communication graph, scale-free graph, random graph, and small-world graph. This approach allows us to analyze the sampling results for different graph data types. Second, for each sampling method, we use nine properties for

directed graph and eight properties for undirected graph. For each graph property, we observe the average SD divergence between the sampling results and the original graph, and then determine whether the sampling methods have stable performance for that graph property. Third, in order to compare the efficiency between sampling methods, we also record the execution time for each sampling algorithm. Finally, we conduct visual comparison between these sampling results. We list the observations from the results below. These findings could potentially provide guidance for users when choosing sampling methods in their applications.

Comparison between graph types

In our experiment, we apply sampling methods to eight graph datasets, including random graph, small-world graph, scale-free graph, and five real-world graphs. We summarize the sampling results for each graph type by averaging the results from different sampling methods and then compare them. Figure 2.6 is summarized from Figure 2.2 by averaging the SD values of all sampling method. Because directed graph and undirected graph have different properties, we summarize undirected graphs (Figure 2.6a) and directed graphs (Figure 2.6b) separately. From Figure 2.6a, we observe that the sampling result of the small-world graph (red line) deviates significantly from both the Facebook graph (green line) and the U.S. flight graph (dark blue line), but the Email graph (cyan line) has a very similar pattern to the scale-free graph (dark blue line). Further, by observing both Figure 2.1 and Figure 2.6, we find that two graphs with similar degree distributions, e.g., email graph and scale-free graph, share similar sampling results. Degree distribution is an important characteristic of the graph type. From these observations, it is obvious that graph

type has significant influence on sampling results and should be considered when sampling graphs.



(a) Summarized sampling results for undirected graph. (b) Summarized sampling results for directed graph.)

Figure 2.6: A summary of Figure 2.2 by averaging the results from different sampling methods.The vertical axis stands for the average SD values, horizontal axis represents average graph properties for all sampling results, and each line shows the results of one dataset.

Comparison between graph properties

We analyze how sampling methods perform on each graph property. As shown in Figure 2.2, all sampling methods not only fluctuate from property to property but also scatter along each property vertically. Hence, sampling methods are dependent on graph properties.

In addition, after reviewing sampling methods performance on undirected graph and directed graph respectively, we find that only a few methods act consistently well on certain graph properties across graph types. Random walk sampling works well on closeness cen-

trality distribution in undirected graphs, and induced-edge sampling behaves consistently well on average neighbor degree distribution of directed graph. Furthermore, for a certain graph data, some methods preserve certain graph properties very well. For instance, in email graph, scale-free graph, and random graph, random edge sampling performs consistently well in in degree distribution, out degree distribution, in degree centrality distribution, and out degree centrality distribution. In Internet graph, snowball sampling performs well in out degree centrality distribution, average neighbor degree distribution, node betweenness centrality distribution, edge betweenness centrality distribution, local clustering coefficient distribution, closeness centrality, and eigenvector centrality distribution. These observations indicate that graph property should be considered when choosing sampling methods in application.

Comparison of execution time

To compare the efficiency of the sampling methods, we record the time for all methods during the sampling process. In Figure 2.3, we find that the sampling process generally takes more time as data size increases with the sampling rate. Random sampling methods, such as random node sampling, random edges sampling, random node-neighbor, random node edge, and induced-edge are not sensitive to sampling rate. However, for traversal-based sampling methods, such as breadth-first, depth-first, random walk sampling, etc., the execution time grows rapidly with the sampling rate. For random sampling, such as, the complexities of random node sampling and random edge sampling are O (n) and O (m), respectively, where n is the number of vertices and m stands for the number of edges. For traversal-based sampling, the complexity of breadth-first sampling is O (n+m), which

explains why random sampling is less sensitive to sampling rate than traversal-based sampling.

Visual comparison

Using visual comparison, we can find out which sampling method preserves the visual cues of a graph. We provide two criteria for visual comparison and analyze how well each sampling method perform.

First, because the locations of vertices in the graph layout are fixed in our visualization, we define the spatial coverage as one criteria of visual comparison between sampling results. The sampling method that produces similar spatial coverage to that of the original graph is considered good. From the visualization of the sampling results, we find that random sampling methods have better spatial coverage than traversal-based sampling, in particular for a small sampling rate. For example, in Figure 2.4 and Figure 2.5, random edge sampling results (red circle) of the U.S. flight and Facebook graphs cover the major spatial area of the original graph. However, traversal-based sampling, such as snowball sampling, does not produce good spatial coverage. That is because traversal-based sampling cannot sample far-ranging vertices or edges as efficiently as random sampling methods for a small sampling rate. For example, the snowball sampling result (red circle) in Figure 2.5j only covers a small local area.

Second, clustering is an important task in graph research. We define another visual comparison criterion in sampling as the ability to preserve the size, shape, and number of clusters. In this regard, we observe that edge-related sampling methods (e.g., random edge) are better than node sampling and traversal-based sampling when the sampling rate is

small. For example, random edge sampling in Figure 2.4 and Figure 2.5 is able to preserve most clusters at 10% sampling rate while random node sampling cannot. The reason is that edge-related sampling methods are biased towards high-degree vertices. They are more likely than node sampling to sample large clusters in a graph.

Table 2.2: Top three sampling methods for each graph property for undirected graphs.The sampling methods in red indicate the corresponding SD values are greater than 0.1

Small World Graph							
DD	**ANDD**	**DCD**	**NBCD**	**EBCD**	**LCCD**	**CCD**	**EVCD**
RE	RE	RW	FF	SB	RE	SB	SB
SB	RN	RNN	SB	FF	RNN	RW	FF
FF	RNE	RE	RW	RW	RWE	RWE	BF
US Flight graph							
DD	**ANDD**	**DCD**	**NBCD**	**EBCD**	**LCCD**	**CCD**	**EVCD**
RN	RE	RNE	RWE	SB	BF	RW	RWE
RNE	RW	RN	BF	RN	SB	DF	RW
RF	RWE	DF	RF	RF	FF	RNE	RE
Facebook Graph							
DD	**ANDD**	**DCD**	**NBCD**	**EBCD**	**LCCD**	**CCD**	**EVCD**
IE	RNN	RNE	RWE	RWE	BF	RW	RE
SB	RF	RN	RW	RNN	SB	RE	RW
BF	BF	RF	RF	RW	IE	RWE	RWE

Summary of good sampling methods for each graph property

Table 2.3: Top three sampling methods for each graph property for directed graphs

Scale-Free Graph								
InDD	**OutDD**	**InCD**	**OutCD**	**ANDD**	**NBCD**	**EBCD**	**CCD**	**EVCD**
RE	FF	RE	RE	IE	RWE	RN	RW	RNN
RWE	RE	RWE	RWE	SB	RW	RNE	FF	RN
RW	RF	RW	RW	DF	RE	RE	IE	RNE
Email Graph								
InDD	**OutDD**	**InCD**	**OutCD**	**ANDD**	**NBCD**	**EBCD**	**CCD**	**EVCD**
RE	RW	RE	RWE	IE	RE	RN	IE	RE
RNN	RE	RNN	RW	RWE	RN	RNE	RWE	RNN
FF	RWE	FF	RE	FF	RNE	SB	RW	FF
Citation Graph								
InDD	**OutDD**	**InCD**	**OutCD**	**ANDD**	**NBCD**	**EBCD**	**CCD**	**EVCD**
IE	IE	FF	RNE	IE	FF	RW	RW	SB
FF	RF	SB	RN	RF	IE	RWE	IE	FF
BF	BF	RN	FF	BF	RF	RNE	RWE	IE
Random Graph								
InDD	**OutDD**	**InCD**	**OutCD**	**ANDD**	**NBCD**	**EBCD**	**CCD**	**EVCD**
RE	RE	RE	RE	FF	RW	DF	RW	RW
RWE	RWE	RW	RWE	IE	DF	SB	SB	DF
RNN	RW	RNN	RW	RF	SB	BF	FF	BF
Internet Graph								
InDD	**OutDD**	**InCD**	**OutCD**	**ANDD**	**NBCD**	**EBCD**	**CCD**	**EVCD**
RNE	DF	RWE	BF	SB	SB	SB	BF	SB
RN	BF	RW	RF	IE	FF	FF	RF	FF
FF	RF	RE	SB	RF	BF	BF	SB	IE

We hope that the sampling results can guide users to choose a suitable sampling method for their applications. Several factors need to be considered, including graph type, graph properties, sampling efficiency, and visual requirements in sampling results. In Figure 2.2, we list the top three sampling methods for each graph property and each graph type according to the average SD distance between the sampling results and the original. If the average SD values are greater than 0.1, we mark the sampling methods in red. From Table 2.2 and Table 2.3, appropriate sampling methods can be chosen based on graph types and graph properties.

In Table 2.2 and Table 2.3, we have several observations. First, for small-world graph, few sampling methods achieve good results in degree centrality and local clustering coefficient. Second, similar graphs, for example, Email graph and scale-free graph, share common choices on sampling methods. Third, induced edge sampling and snowball sampling are good candidates on most graph properties for citation graph and Internet graph, respectively.

Based on the above analysis, consideration of all factors will allow users to make more informed choices on sampling methods. For example, if the execution time is important, it is more reasonable to use random sampling methods for very large graphs than traversal-based sampling methods. For a particular graph type and graph property, we can refer to the comparison results to get a good sampling method. For example, to preserve node betweenness centrality distribution and in degree distribution on large scale-free graph, random edge sampling is a good choice because the sampling results of random edge show good agreement with the original graph for these two graph properties. If a number of

factors need to be considered, we have to sort the priority list of these factors first, and then choose appropriate sampling methods.

2.5 Discussion and Observations

We explored twelve sampling methods and applied those sampling methods to random graph, small-world graph, scale-free graph, and real-world graph. The graph data range from 235 to 265,214 vertices and from 1,297 to 421,578 edges. Eight undirected graph properties and nine directed graph properties are used to evaluate those sampling methods. Our visual and statistical benchmark evaluates sampling methods for their effectiveness in preserving both the quantitative statistical properties and qualitative visual properties of the original graph. The initial analysis indicates that the ranking of these graph sampling methods is dependent on a list of factors, including graph type, desired statistical property, sampling efficiency, and visual requirements. For example, in a time-sensitive task, users will need to consider the computation time for sampling. If one graph property is particularly important in sampling results, users could choose sampling methods according to their rank in Table 2.2 and Table 2.3.

Furthermore, the visual comparison of the sampling methods gives users an intuitive understanding of the differences among them. The consistent graph layout in the benchmark facilitates the visual comparison and identification of features for each sampling method. In addition, two visual comparison criteria are defined to help users compare sampling methods.

Finally, the results provide insight into the effectiveness of each sampling method in preserving statistical properties. Graph type, graph properties, sampling efficiency, and visual requirements in sampling results are the four key factors when choose sampling methods. The result could help users choose which method to use for a particular application.

CHAPTER 3

DISTRIBUTED GRAPH SAMPLING METHODS

In the previous chapter, we built a benchmark for evaluating traditional/non-distributed graph sampling methods with both visual and statistical properties. In this chapter, we will discuss the similarities and differences between non-distributed sampling methods and distributed sampling methods, and elaborate on the merits of distributed sampling methods in large-scale graph exploration.

3.1 Introduction

Many sampling techniques have been proposed aiming to sample a representative subgraph from the original graph. Those sampling methods can be classified in three categories: node sampling [66, 8], edge sampling [8, 32] and topology-based sampling [63, 84]. Unfortunately, many of these non-distributed sampling methods, especially topology-based sampling, are not practical for large-scale graphs (more than 1 million vertices) because of efficiency issue or memory issue. For example, in our previous research [109], we found that sampling on an Email graph (vertices: 265, 214; edges: 420, 045) took more than 24 hours while using topology-based sampling. Most prior sampling research only worked on graphs with less than 1 million vertices [6, 66, 100]. With the increase of graph size, scalable sampling methods are in urgent demand in graph research.

In this work, we aim to design and develop a number of scalable topology-based sampling methods based on a MapReduce framework-Spark [108], and make comprehensive evaluations between non-distributed sampling approaches and our new methods. For completeness, scalable node sampling and scalable edge sampling techniques are also included in our work. These scalable sampling techniques can greatly improve sampling efficiency without losing sampling accuracy.

To improve the reliability of these distributed sampling algorithms, we apply both non-distributed sampling methods and new sampling methods to three datasets, and make statistical and visual comparison between them.

The main contributions of our work are as follows:

- We designed and developed nine scalable graph sampling methods based on Spark.
- We made visual and statistical comparison between non-distributed sampling methods and distributed graph sampling methods.
- We implemented a large-scale graph sampling package which can be incorporated into GraphX for graph research.
- We analyzed the nine distributed graph sampling methods from visual and statistical perspectives, and summarized their merits compared to non-distributed sampling techniques.

3.2 Related Work

Graph sampling has been an interesting field within graph research for many years. Many sampling techniques have been proposed in last few decades. Existing graph sampling algorithms can be classified into three categories [6, 66]: node sampling, edge sampling, and topology-based sampling.

Node sampling is a simple sampling method which creates representative graph by sampling nodes independently and uniformly. For example, in random node sampling [66], vertices are sampled randomly and uniformly, creating a subgraph from the original graph. The edges between the sampled vertices in the original graph are included the sample graph as well. Sometimes, node sampling also considers neighbors of the sampled vertices, for instance, random node-edge sampling [55] and random node-neighbor sampling [66]. In random node-edge sampling, when vertices are uniformly sampled, edges that are incident to these vertices are also uniformly sampled in the sample graph. In random node-neighbor sampling, all the edges that are connected to these vertices in the original graph are sampled into subgraphs. In some cases, node sampling methods integrate topology-based sampling in order to use graph topology information, such as random walk sampling [107]. The metropolis algorithm [56] is a modified version of node sampling. It replaces some sampled vertices with other vertices, which often leads to sampled graph properties that are consistent with the original.

Similarly, edge sampling builds a subgraph by randomly sampling edges. Random edge sampling [32] is one typical edge sampling in which edges are sampled randomly and uniformly, and then a subgraph is created from those edges. Induced edge sampling [8] is

another edge sampling method, which includes totally induced edge sampling and partially induced edge sampling. Totally induced edge sampling has two steps. First, it conducts random edge sampling and obtains adjacent vertices from these edges. Second, all edges attached to those vertices are sampled in a subgraph. Partially induced edge sampling performs edge sampling in a single pass in which edges are selected with a probability. Incident vertices are also added to the sampled graph if one edge is selected. In this work, we implement totally induced edge sampling.

Topology-based sampling is regarded as the state-of-art sampling methods because they have good ability to preserve graph properties [6]. Graph traversal algorithms are often used in these sampling methods. For example, breadth-first sampling [100] creates a subgraph by using breadth first search algorithm. It begins with a random vertex and visits its neighbors iteratively. For each iteration, the first visited vertex will enter the sample first. A subgraph is created from visited vertices and those edges that are connected between those sampled vertices in the original graph. Forest fire sampling [6, 8] can be regarded as a probabilistic version of breadth-first sampling. Neighbors are chosen to be added to the subgraph with probability p. The number of vertices to be chosen is a random number taken from a geometric distribution with mean $p_f/(1p_f)$. Random walk sampling starts at a seed vertex, and then chooses a vertex uniformly at random from the neighbors of the current vertex. A subgraph is created from the walking paths. Random walk with escape or jump [32] and multiple independent random walkers [55] are proposed based on the classic random walk sampling method. Random walk with escape or jump sampling is

more random than random walk sampling since the current walker vertex jumps to another random vertex with probability p.

Spark [90] is an in-memory distributed computing framework that manipulates datasets in memory across distributed machines, which is different from Hadoop MapReduce [30] in that it keeps intermediate data in memory instead of storing it on disk. This strategy makes Spark run up to 100 times faster than Hadoop MapReduce [108]. It allows us create iterative algorithms efficiently because it offers us a MapReduce environment. GraphX [43], a Spark library, takes the advantage of data-parallel and graph-parallel systems in Spark framework. Within GraphX, several graph property calculation algorithms are implemented, such as PageRank, connected component, shortest paths, etc. In this work, we design distributed sampling methods and implement them on Spark.

3.3 Distributed Graph Sampling Methods

In this section, we present three types of novel graph sampling techniques: 1) distributed node sampling, 2) distributed edge sampling, and 3) distributed topology-based sampling. We make analytical comparisons between non-distributed sampling methods vs. our distributed methods. The node sampling includes random node sampling (RN), random node edge sampling (RNE), and random node neighbor sampling (RNN). The edge sampling includes random edge sampling (RE), induced edge sampling (IE), and random hybrid sampling (RH). The topology-based sampling includes breadth first sampling (BF), snowball sampling (SB) and forest fire sampling (FF).

3.3.1 Distributed Node Sampling Methods

Node sampling constructs subgraphs based on sampling vertices from the original graph. In random node sampling, vertices are sampled randomly and uniformly.

Given a graph $G = (V, E)$, where V and E are vertices and edges of G. Let Gs degree sequence be $\{1, 2 \dots i \dots k\}$, the number of degree i is $N(i)$. If node sampling rate is r, then the expected nodes in non-distributed random node sampling can be represented as:

$$E_{non-distributed}(NS(r)) = N(1) * r + N(2) * r + \cdots + N(k) * r = r * \sum_{i=1}^{k} N(i) \quad (3.1)$$

For distributed node sampling algorithms, since graphs are distributed into multiple machines, each machine contains one or more partitions. In distributed node sampling, nodes are sampled from each partition independently. If a graph is distributed into n partitions, then the expected nodes in one partition can be represented as:

$$E_p(NS(r)) = N_p(1) * r + N_p(2) * r + \ + N_p(t) * r$$

where t is maximum degree in partition p, and $t <= k$. If the graph has n partitions, then for overall partitions:

$$E_{distributed}(NS(r)) = E_1(NS(r)) + E_2(NS(r)) + \cdots + E_n(NS(r)) = \sum_{t=1}^{n} E_t(NS(r))$$

Since $N_1(i) + N_2(i) + \cdots + N_n(i) = N(i)$, then

$$\begin{aligned} E_{distributed}(NS(r)) = \sum_{t=1}^{n} E_t(NS(r)) &= N(1) * r + N(2) * r + \cdots + N(k) * r \\ &= \sum_{i=1}^{k} \sum_{t=1}^{n} N_t(i) * r = r * \sum_{i=1}^{k} N(i) \end{aligned} \quad (3.2)$$

From equation 3.1 and 3.2, theoretically, both distributed node sampling and non-distributed node sampling should have the similar results.

3.3.2 Distributed Edge Sampling Methods

Edge sampling builds a subgraph by randomly sampling edges. If $|E|$ is the number of edges, sampling rate is r, then there should be $r * |E|$ edges sampled in sampling results. Suppose the degree original graph is $\{1, 2 \dots i \dots k\}$, the number of degrees i is $N(i)$, the probability of one node with certain degree falling into sample is $[1 - (1 - \frac{i}{|E|})]^{r|E|}$. Thus, the expected nodes in random edge sampling can be described as:

$$\begin{aligned} E(ES(r)) = N(1) * [1 - (1 - \frac{1}{|E|})]^{r|E|} + N(2) * [1 - (1 - \frac{2}{|E|})]^{r|E|} + \\ \cdots + N(k) * [1 - (1 - \frac{k}{|E|})]^{r|E|} = \sum_{t=1}^{k} N(t) * [1 - (1 - \frac{t}{|E|})]^{r|E|} \end{aligned} \tag{3.3}$$

Similarly, in distributed edge sampling algorithms, each machine also contains one or more partitions. In each partition, edges are sampled independently. If one graph is distributed into n partitions, then the expected nodes in one partition can be represented as:

$$E_p(ES(r)) = N_p(1)*[1-(1-\frac{1}{|E|})]^{r|E|}+N_p(2)*[1-(1-\frac{2}{|E|})]^{r|E|}+\cdots+N_p(t)[1-(1-\frac{k}{|E|})]^{r|E|}$$

where t is maximum degree in partition p, and $t \leq k$. If graph has n partitions, then for overall partitions:

$$E_{distributed}(ES(r)) = E_1(ES(r)) + E_2(ES(r)) + \cdots + E_n(ES(r)) = \sum_{t=1}^{n} E_t(ES(r))$$

Since $N_1(i) + N_2(i) + \cdots + N_n(i) = N(i)$, then,

$$
\begin{aligned}
E_{distributed}(ES(r)) = \sum_{t=1}^{n} E_t(ES(r)) = N(1) * [1 - (1 - \frac{1}{|E|})]^{r|E|} + \\
N(2) * [1 - (1 - \frac{2}{|E|})]^{r|E|} + \cdots + N(k) * [1 - (1 - \frac{k}{|E|})]^{r|E|} = \\
\sum_{i=1}^{k} \sum_{t=1}^{n} N_t(i) * [1 - (1 - \frac{i}{|E|})]^{r|E|} \\
= \sum_{i=1}^{k} N(i) * [1 - (1 - \frac{i}{|E|})]^{r|E|}
\end{aligned}
\tag{3.4}
$$

From equation 3.3 and 3.4, theoretically, both distributed edge sampling and non-distributed edge sampling should have the similar results as well.

From the distributed random node sampling and distributed random edge sampling, we can easily demonstrate that distributed random node edge sampling, distributed random node neighbor sampling, distributed induced edge sampling, and distributed random hybrid sampling can produce the similar sampling results as its corresponding non-distributed sampling method.

3.3.3 Distributed topology-based sampling

Since topology-based sampling creates subgraphs based on the topological information of the original graph, instead of sampling vertices or edges directly, which selects nodes or edges in sequential order. The topological visited indices represents the order that one node or edge enter sampling results. In distributed topology-based sampling, there are two challenges in the sampling process. First, the graph is distributed into multiple machines, it is not easy to create visited index in such a graph because it involves frequent communication between partitions, particularly while sampling on multiple partitions at the

same time. Second, it is important to take into consideration that that one graph may have multiple unconnected components. Those components might be partitioned into multiple machines. Which component starts the sampling process has a great influence in sampling results.

To solve the aforementioned challenges, we develop new strategies to implement topology-based sampling on distributed graphs. The topology-based sampling process is divided into two stages: vertex labeling and sampling. Initially, a screening for the quantity of unconnected components the graph contains will be completed. All components have equal probability to act as the seed node, and they are visited in sequential order. We keep a record of the number of vertices for each component during labeling stage. Each vertex has an index number that indicates how many vertices were visited before reaching the current vertex. Thus, the sampling process can be tracked with the label index in the sampling stage. Because our distributed topology-based sampling uses the same principle as non-distributed topology-based sampling, they should produce similar sampling results. Forest fire sampling is a randomized version of breadth first sampling [6], and snowball sampling is similar to breadth first sampling [32]. Here we only provide the pseudocode of distributed breadth first sampling methods.

3.3.4 Impementation of Distributed Sampling Algorithms

The nine distributed sampling methods provided above are implemented using GraphX in Spark. GraphX provides two special versions of resilient distributed datasets (RDD): a VertexRDD and an EdgeRDD, which are used to represent vertex information and edge

Algorithm 1 Distributed Breadth First Sampling

Require: Graph $G = (V, E)$, sampling rate θ

Broadcast (θ)

$G_c \leftarrow$ add one vertex attribute $(index \leftarrow MaxValue)$

$step \leftarrow 1$

$componentsList \leftarrow$ components in G_c

for all each component $c : G_c$ **do**

$startNode \leftarrow$ choose a random node in c

iteratively modify vertex attribute in c ($index \leftarrow step$)

$step \leftarrow step + 1$

end for

for all $s : 1$ to $step$ **do**

check the number of vertices when $index < s$

iteratively modify vertex attribute in c ($index \leftarrow step$)

if the number of vertices is not satisfied **then**

continue

end if

end for

$G_{subgraph} \leftarrow G_c((V, index < s), E)$

information in memory or hard disk. The distributed sampling algorithms are written in Scala language and compiled into a JAR file for distribution. This package and source code will be uploaded to GitHub for public access.

3.4 Experimental Evaluation

Here we present our experiments to evaluate distributed sampling methods from visual and statistical perspectives by making comparisons with non-distributed sampling methods on three graph datasets ranging from 88234 edges to 1,806,067,135 edges (described in Table 1). Specifically, we will introduce some graph properties, and statistical and visual comparison techniques used in the evaluation.

3.4.1 Graph Datasets

We apply our sampling algorithms on the three datasets: Facebook graph, Amazon graph, and Friendster graph that are collected from Stanford Network Analysis Platform (SNAP) [68]. Facebook, as an anonymized graph using an integer number as user id, was collected from the Facebook app [69]. Amazon was created from the Amazon website based on the relation between items that are co-purchased [106]. Friendster is an on-line gaming network created from a social networking site [106]. These graph datasets are summarized in Table 1.

3.4.2 Graph Metrics

For evaluation of distributed sampling methods, several graph metrics are used to assess the quality of sampling results in statistical comparisons. The comparison between sam-

Table 3.1: Three test datasets and their properties

Graph Dataset	Graph Type	Model	# Vertices	# Edges
Facebook	Undirected	Real	4,039	88,234
Amazon	Undirected	Real	334,863	925,872
Friendster	Undirected	Real	65,608,366	1,806,067,135

pling results and original graphs is realized by comparing their graph property distributions in statistics. Here we focus on degree distribution (DD), average neighbor degree distribution (ANDD), triangle distribution (TD), PageRank distribution (PR), and local clustering coefficient distribution (LCCD).

3.4.3 Statistical Comparison

To evaluate distributed sampling methods and non-distributed sampling methods, we appraise their performance by how well the sampled results preserve each of graph properties. A good sampling method should produce a subgraph that approximates the original graph. That is, the probability distributions of the properties of the two graphs should have a short distance between them. Here we use skew divergence (SD) to evaluate the difference between two distributions [65]. Generally, skew divergence is used to measure Kullback-Leibler (KL) divergence between two probability density distributions that do not have continuous support over the range of values. Because graph properties distributions are not continuous, e.g. clustering coefficient, the two probability density distributions should be smoothed before computing KL divergence. We use the same strategy as Ahmed [6] and Lee [65] to smooth the distributions:

$$SD(P, Q, \alpha) = KL[\alpha P + (1 - \alpha)Q||\alpha Q + (1 - \alpha)P]$$

To better compare the sampling results, we use the average SD defined in the work, and is set to 0.99 as in the paper Ahmed [6] and Lee [65]. Previous work [65] has proven that SD has better performance approximating KL divergence on non-smoothed distributions.

3.4.4 Visual Comparison

In addition to making comparisons between sampled subgraphs and the original graph statistically, we also compare them visually by using Gephi [15]. We first draw the original graph and export the decorated graph into a file with vertex decorations preserved (e.g. vertex color, label size, location). When sampling on the decorated graph, vertex attributes are also sampled along with vertices. The layout in the sampling results should have the same layout as in the decorated graphi.e., the same vertex in all sampled graphs will occupy the same location as in the decorated graph. Also, the same vertex in all sampled graphs has the same color and label size as the original graph. We do not preserve the attributes of edges, such as edge color, edge weight, etc. The vertex attributes, for example, vertex position, color, are preserved in subgraph because we consider that such vertex attributes are significant in visual comparison. In this way, the similarity and difference within or between sampling results can be easily identified. Here we provide the visual comparison of Facebook graph in this work.

3.4.5 Results

In our experiment, both non-distributed and distributed sampling methods are applied to Facebook, Amazon, and Friendster graph datasets. We investigate the performance of distributed sampling methods by comparing their abilities to preserve the features of original graph. When sampling these graphs, we set the sampling rate at 15% and 25% based on the number of vertices for each graph. For each sampling rate, there are 5 different runs carried out in this experiment. We take the average SD value as final value for analyzing each graph-metric.

We first compare sampling results visually on Facebook data in Figure 3.2 (For space limits, we only provide visual comparison at 15% sampling rate). The non-distributed sampling results and distributed sampling results are visualized in two columns. Each row in Figure 3.2 represents one sampling method. To allow for better comparison, the original graph is also visualized in Figure 3.1.

We then compare sampling results in statistics on Facebook (Figure 3.3) and Amazon (Figure 3.4) graph, and each comparison includes two sampling rates: 15% and 25%. Since the Friendster graph is too large for non-distributed sampling methods because of unaffordable sampling time cost, we do not make statistical comparisons on this graph data. In each statistical comparison, the line charts (in Figure 3.3 and 3.4) indicate the SD between sampling results and original graphs for each sampling method on each statistical property. The vertical axis in the line chart is the SD value between the sampling result and the original graph ranging from 0 to 1. A smaller value denotes more consistency between the sampling result and the original graph. The horizontal axis lists the graph

properties. Each line in the chart represents one sampling method. Its value indicates the sampling methods performance. From the statistical comparison, we can identify how those sampling methods perform, and similarities or differences between them.

The last but the most important comparison between non-distributed sampling methods and distributed sampling methods is on efficiency shown in Figure 3.5. Charts are presented in two columns representing efficiency comparison at two sampling rates (15% and 25%). For the Friendster graph, the execution time on non-distributed sampling methods is not provided in Figure 3.5 for the same reasons as in statistical comparison.

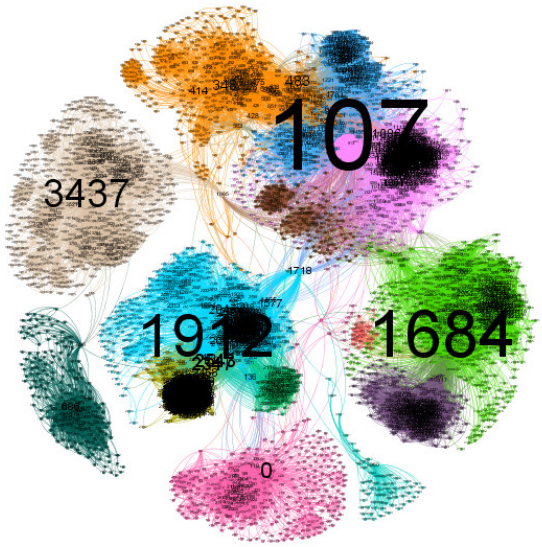
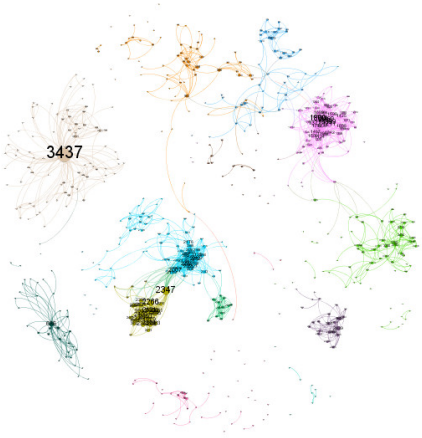
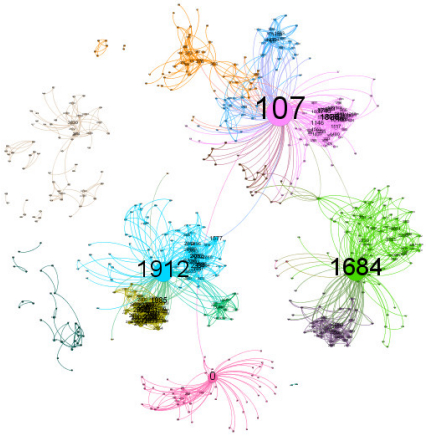


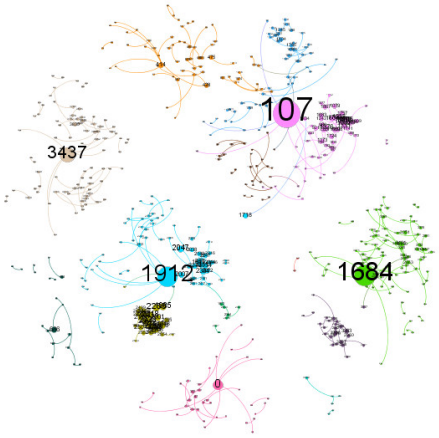
Figure 3.1: Visualization of original Facebook graph. Each group of vertices and edges with unique color represents one clusters. Label size is proportional to vertices degree.



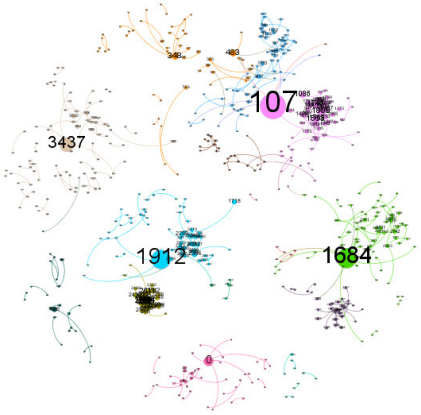
(a) Non-Distributed RN



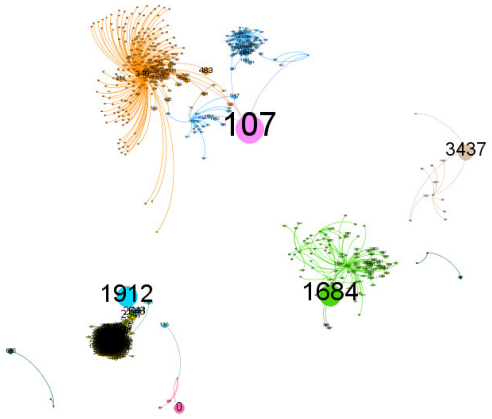
(b) Distributed RN



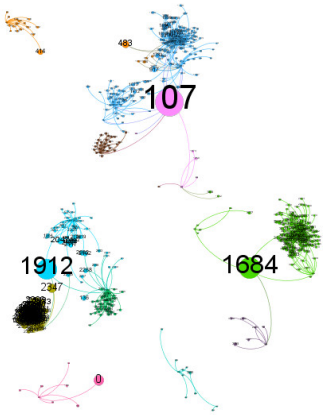
(c) Non-Distributed RNE



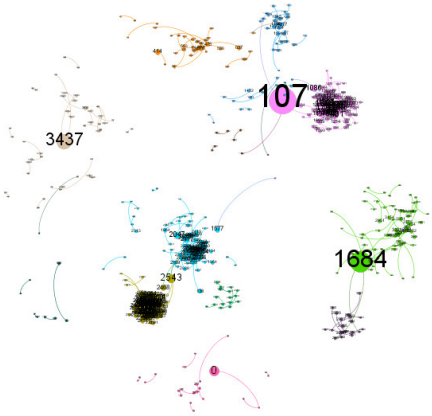
(d) Distributed RNE



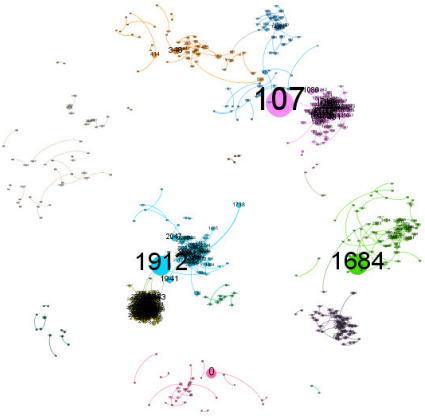
(e) Non-Distributed RNN



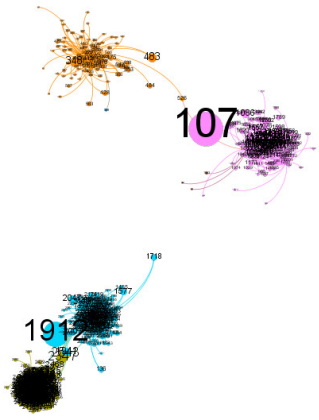
(f) Distributed RNN



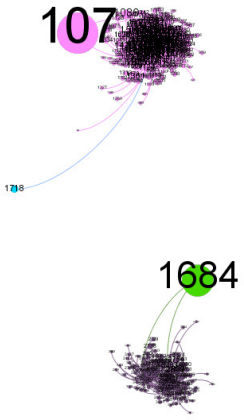
(g) Non-Distributed RE



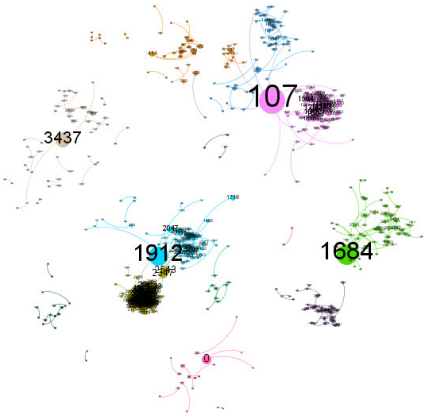
(h) Distributed RE



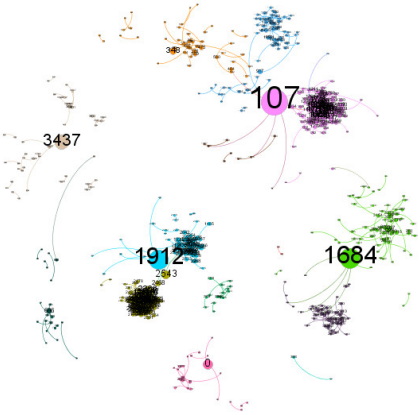
(i) Non-Distributed IE



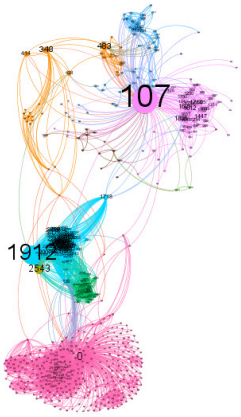
(j) Distributed IE



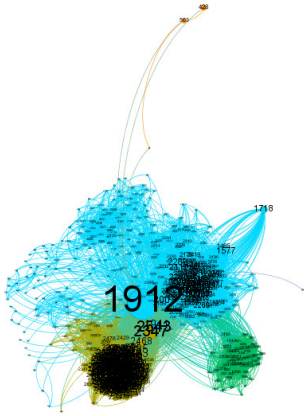
(k) Non-Distributed RH



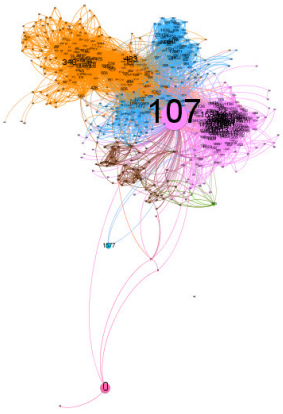
(l) Distributed RH



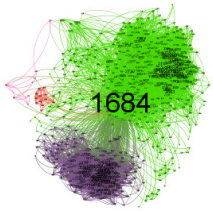
(m) Non-Distributed BF



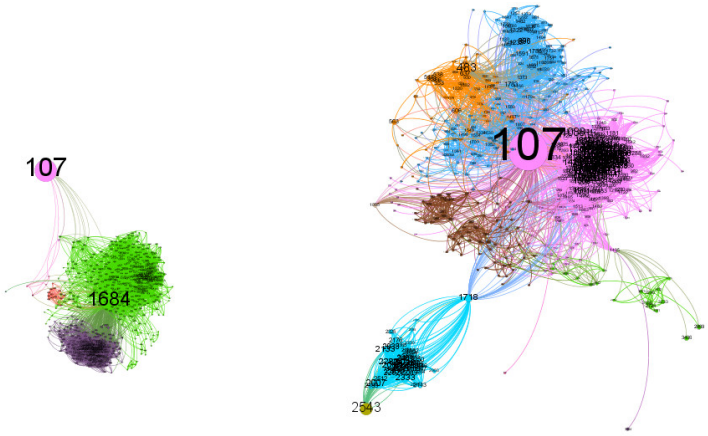
(n) Distributed BF



(o) Non-Distributed SB



(p) Distributed SB



(q) Non-Distributed FF (r) Distributed FF

Figure 3.2: Visual comparison between traditional sampling methods and distributed sampling methods (in columns) at 15% sampling rate on Facebook graph. Each image stands for one sampling result created by corresponding sampling method in row

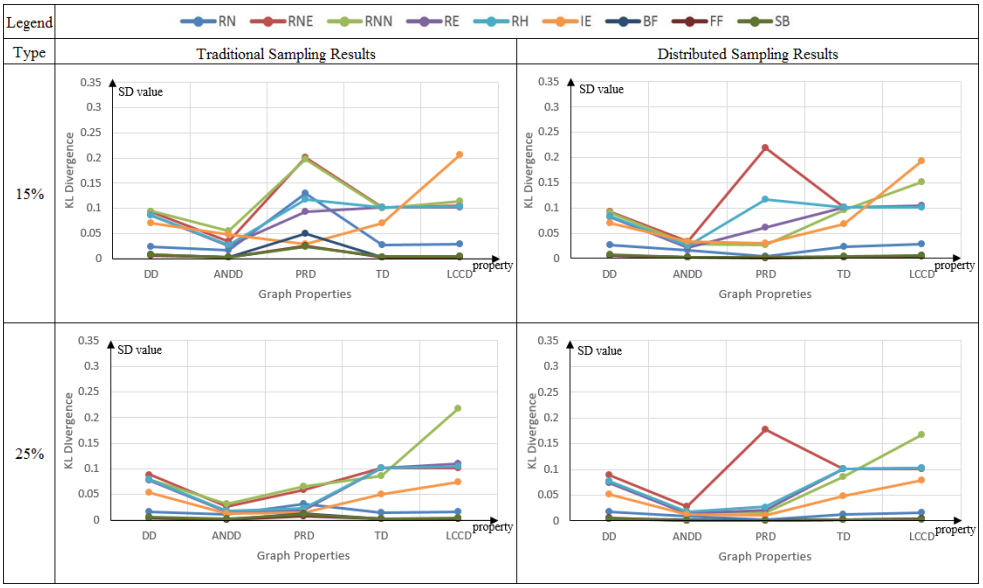


Figure 3.3: Statistical comparisons between traditional sampling methods and distributed sampling methods (in two column) on Facebook graph. The vertical axis is SD values, horizontal axis represents graph properties, and lines represent one sampling methods.

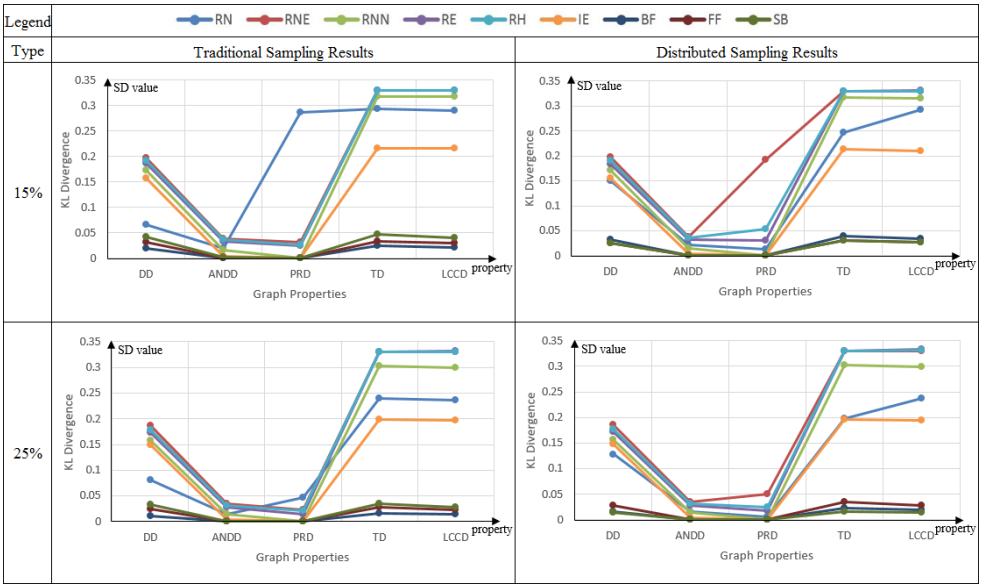


Figure 3.4: Statistical comparisons between traditional sampling methods and distributed sampling methods (in two columns) on Amazon graph. The vertical axis is SD values, horizontal axis represents graph properties, and lines represent one sampling methods.

3.4.6 Analysis

From the experiment results, we can compare sampling methods quantitatively or qualitatively in several aspects.First, from statistical comparison, we can identify how sampling methods preserve graph properties and the similarities and differences between two types of sampling methods. Second, from visual comparison, we discuss the similarities and differences between distributed sampling methods and non-distributed sampling approaches. Third, we compare those sampling methods in efficiency for each dataset. Finally, we summarize characteristics of distributed sampling methods.

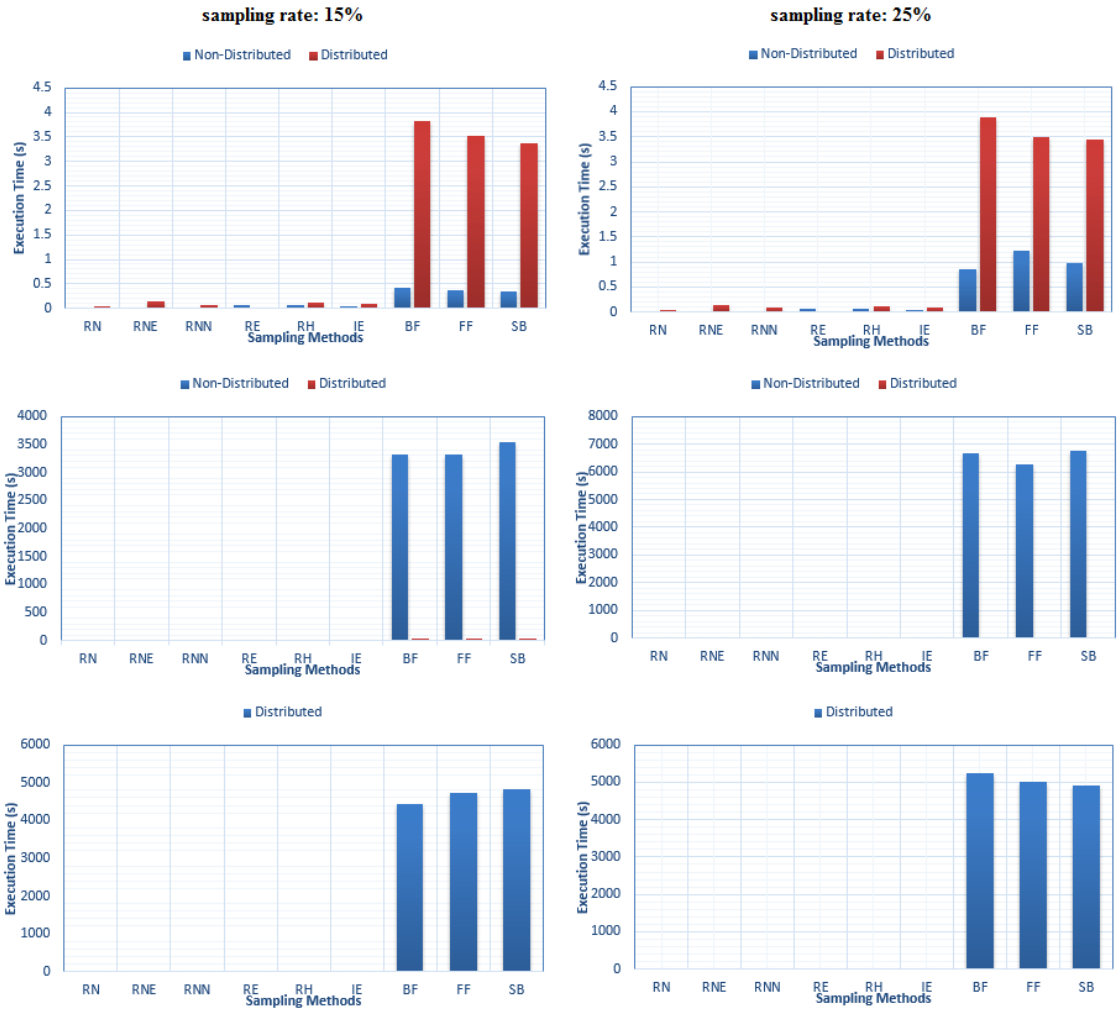


Figure 3.5: Statistical comparisons between traditional sampling methods and distributed sampling methods (in two columns) on Amazon graph. The vertical axis is SD values, horizontal axis represents graph properties, and lines represent one sampling methods.

3.4.6.1 Statistical Comparison

From statistical comparison results in Figure 3.3 and 3.4, we can draw some observations. Generally, distributed sampling methods have the similar performance as the non-distributed in preserving graph properties except for a few sampling methods. In Figure 3.3 and 3.4, most sampling methods show roughly consistent performance between the two sampling types. There are some abnormal cases. For example, in Facebook data (Figure 3.3) at sampling rate 15%, distributed random node neighbor sampling shows better than non-distributed random node neighbor sampling in persevering PageRank. In Amazon data (Figure 3.3) at sampling rate 15%, distributed random node sampling shows better than non-distributed random node sampling in persevering PageRank. Conversely, in Facebook data (Figure 3.3) at sampling rate 25% and Amazon data (Figure 3.4) at sample rate 15%, non-distributed random node-edge sampling method preserves PageRank better than distributed node-edge sampling method. In addition, by observing Figure 3.3 and 3.4, we find that both non-distributed and distributed random sampling methods (e.g. random node sampling, random edge sampling) cannot behave as well as topology-based sampling in preserving degree, triangle, and local clustering coefficient, particularly for large graphs. For instance, in Figure 3.4, random sampling methods shows bad performance on degree distribution, triangle distribution, and local clustering coefficient distribution. Topology-based sampling methods perform consistently well in preserving all graph properties. This observation about non-distributed topology-based sampling methods accords with findings in previous studies [6, 109].

3.4.6.2 Visual Comparison

We visualize sampling results from Facebook graph at sampling rate 15% for both non-distributed and distributed sampling methods, shown in Figure 3.2. From the visual comparison, we can find out the tendency and preference of each sampling method in preserving visual properties.

First, distributed random sampling methods have very similar performance with non-distributed random sampling methods in preserving spatial coverage. For example, by referring Figure 3.1, we can find both kinds of random sampling roughly generate same spatial area in both column. This indicates that distributed random sampling methods have the same preference as non-distributed random sampling methods when sampling graphs. Second, sometimes distributed sampling methods have better ability to preserve the number of clusters. From the visual comparison in Figure 3.2, distributed random node sampling and random node neighbor sampling can sample more clusters than corresponding non-distributed sampling approaches. Third, for topology-based sampling, their ability to preserve spatial coverage and clusters is more random. This is because the seed node is select at random and it has great influence in sampling results. In breadth first sampling, non-distributed sampling seems to be better in preserving spatial coverage and the number clusters. But in forest fire sampling, distributed sampling method shows better than non-distributed sampling in preserving such visual properties. Forth, generally, random sampling methods are more likely to sample broad spatial coverage and more clusters at the same sampling rate with topology-based sampling methods. This conclusion is consistent with previous findings as well [109].

3.4.6.3 Efficiency Comparison

To compare the efficiency for all sampling methods, we record the execution time for all methods during the sampling process (shown in Figure 3.5). From this figure, we can draw some interesting observations. First, for small graphs, non-distributed sampling methods have higher efficiency than distributed sampling methods. However, for large-scale graphs, distributed sampling approaches have remarkable improvement in efficiency (around 100 times) than non-distributed sampling methods, in particular for topology-based sampling (e.g. breadth first sampling, forest fire sampling, snowball sampling). For example, in Figure 3.5, Amazon graph is larger than Facebook graph. Non-distributed sampling methods use less time in Facebook graph at both sampling rates. However, distributed sampling methods are more efficient in the Amazon graph for both sampling rates. For very large-scale graphs, distributed sampling methods have huge advantage over non-distributed sampling methods. For instance, non-distributed topology-based sampling cannot complete the sampling process in an affordable time, but distributed topology-based sampling methods can complete in 2 hours on billion-scale graphs. In addition, for both kinds of sampling approaches, sampling time increases with the increase of sampling rate and graph size, but random sampling (e.g. random node sampling, random edge sampling) is not as sensitive to graph size as topology-based sampling. This observation is also reflected in previous studies [109].

3.4.6.4 Summary of Distributed Sampling Methods

From the above statistical comparison, visual comparison, and efficiency comparison. The merits of distributed sampling methods can be summarized as follows:

- Distributed sampling methods roughly have similar ability in preserving graph properties. They are as reliable as non-distributed sampling methods. In some cases, distributed sampling performs better than non-distributed sampling.

- Generally, distributed sampling methods have similar performance with non-distributed random sampling methods in preserving spatial coverage. On some occasions, distributed sampling methods have better ability to preserve clusters than non-distributed sampling methods.

- Distributed sampling is more efficient than non-distributed sampling on large-scale graphs, which scales with graph size.

- Distributed sampling methods have horizontal scalability. Theoretically, while using distributed sampling methods, more machines can handle larger graphs.

3.5 Conclusion

We designed and developed nine distributed sampling methods and made statistical comparisons, visual comparisons, and efficiency comparisons with non-distributed sampling methods by applying those sampling methods to Facebook graph, Amazon graph, and Friendster graph.

In the statistical comparison, there were five graph properties used to evaluate sampling methods in preserving such quantitative statistical properties. We found distributed sampling methods have good reliability compared with non-distributed sampling methods. This observation offers us a reasonable justification to apply such sampling methods to large graphs in application. We can also use such sampling methods to make accurate estimations about graph properties on original graphs.

In visual comparison, we analyzed distributed sampling methods and non-distributed sampling methods on their ability to preserve spatial coverage and size, shape, and number of clusters. Such comparisons provide us an intuitive understanding of the similarities and differences among them. From the comparisons, we learned that distributed random sampling approaches have similar or even better performance in preserving spatial coverage and clusters. This observation gives us another justification to use distributed sampling methods in application.

In our efficiency comparison, we found distributed sampling methods have a great advantage over non-distributed sampling methods when sampling on large-scale graphs. This is the most important reason why we designed and develop distributed sampling methods for large-scale graphs.

From the above three comparison, we summarized the assets of distributed sampling methods. The findings in this work could help users choose proper sampling methods in application.

CHAPTER 4

BGS: LARGE-SCALE GRAPH VISUALIZATION SYSTEM

In chapter 2 and chapter 3, we presented a benchmark for evaluating traditional/non-distributed graph sampling methods with both visual and statistical properties and nine distributed graph sampling methods. Also, a comparison was made between non-distributed graph sampling and distributed graph sampling. Those distributed graph sampling methods can help us explore large-scale graph efficiently through a representative of original graph. In this chapter, we will propose a new scalable graph visualization system-BGS (Big Graph Surfer), which allows us to visualize large-scale graphs by the virtue of a series of hierarchy and graph exploration methods.

4.1 Introduction

Graph visualization, as a popular and natural method to analyze graph data, is widely used in many fields. Many graph visualization systems [52, 38, 12, 3, 34], generic or specific, have been proposed across a variety of domains in the last few decades. Those visualization tools are truly effective in exploring relatively small graphs, whereas they are not applicable to very large-scale graphs. There are several issues that greatly impair graph visualization, such as memory issue, display issue, layout issue, and interaction issue.

To solve these issues, we propose a new graph visualization system called BGS (Big Graph Surfer) intended to visualize very large-scale graphs by generating hierarchical structure from original graphs and providing a series of graph exploration methods. According to Shneidermans visualization principle of "Overview fast, zoom and filter, then details-on-demand" [89], BGS provides hierarchy view and graph view that allow us to navigate along the hierarchy by expanding or collapsing clusters, zooming in or zooming out to observe details or overviews, highlighting and focusing on vertices. To realize such manipulations, the basic technique we used is graph hierarchy, which is widely used in many visualization systems [11, 1, 3, 61]. Graph hierarchy was proposed to visualize a graph at multiple layers, which can reduce the number of displayed vertices while preserving structural information. At the same time, graph hierarchy provides us a series of abstractions on original graph data. The meaningful abstractions not only enhance layout performance and rendering, but also reduce visual complexity in visualization.

To construct graph hierarchical structure, clustering [86, 97, 62] is broadly applied by researchers to create hierarchies on graphs, which discovers groups or communities based on a certain semantics and abstracts them recursively. Clustering includes content-based clustering and structure-based clustering. Content-based clustering is one clustering method based on the meaning of attributes, which only works for performing clustering on attributed graphs. Since BGS is designed as a general visualization system, it uses one type of structure-based clustering methods-Louvain clustering technique to build the hierarchy [20].

In term of architecture, BGS is developed on several platforms: Spark [108, 5], R [57], RStudio [94], and Shiny [23]. Spark is a distributed computing framework deployed on supercomputer, which acts as a back-end platform working on graph hierarchy construction, graph filtering, and aggregation etc. Shiny works as front-end to visualize graphs through the Web. R and Rstudio act as intermediate link that is responsible for communication with back-end and front-end. Specifically, they translate the operation requests from front-end to back-end, and send results from back-end to front-end for visualization. This architecture makes our tool very powerful in dealing with large-scale graphs. In our study, BGS can easily handle large graph with billion-scale vertices or edges. Theoretically, adding more computers allows for handling larger graphs.

In addition, BGS provides two visualization modes (Local-Memory mode and Distributed-Memory mode), two expansion modes (Minimum mode and Add-Up mode), and two graph view modes (Regular mode and Edge-Free mode). These visualization modes, expansion modes, and view modes can produce several different combination modes. All these combinations modes are helpful in dealing with different occasions. This is a unique feature for our system.

In summary, the main contributions of our visualization tool are as follows:

- The architecture of BGS brings significant increase on graph visualization scalability, which makes BGS capable of visualizing graphs with billion-scale vertices or edges.

- BGS uses an efficient clustering technique in hierarchy construction-Louvain clustering, which is the optimal combination of speed and accuracy, and implements it in distributed computing system.

- BGS provides two visualization modes, two expansion modes, and two graph view modes. These techniques allow us to explore hierarchies and graphs based on users' needs and visualization efficiency..

- BGS supports direct search on hierarchy view and graph view by vertices attribute(s) or edges attribute(s), which helps users identify interesting vertices or edges promptly.

This chapter is organized as follows. In next section, we will introduce the background of graph visualization and discuss some related work in previous research. Then, we present our graph visualization system, including graph clustering techniques, architecture, hierarchical structure, and visualization design, followed by case study and evaluation on several graph datasets. Afterwards, we discuss scalability and usability of the system. Finally, we conclude the work and introduce future work in the last section.

4.2 Related Work

A variety of graph visualization systems have been proposed, such as ASK-GraphView [3], CGV [95], TeGViz [60], GraphVizdb [19], Network Explorer [44], Vizster [50], ZAME [35], Matrix Zoom [2], etc. In this section, we analyze these visualization tools, discuss their strengths and weaknesses, and talk about how BGS takes advantage of their merits and overcomes their drawbacks in architecture, graph representation, graph exploration, interaction etc.

In architecture, GraphVizdb uses database-MySQL as server for storing graph data and WebUI as client for visualization interface. TeGViz uses a distributed system as server and adjacent matrix to represent graphs. BGS has a similar client-server mode to GraphVizdb and TeGViz. This mode can greatly increase graph visualization scalability. BGS uses a distributed system as server for graph data manipulation and WebUI as client for graph visualization. This architecture takes the advantage of high efficiency in distributed system and flexibility in WebUI.

In graph representation, TeGViz, Matrix Zoom, and ZAME are developed using adjacency matrix in graph visualization. Compared to node-link diagram, adjacent matrix has one major disadvantage in generating hierarchy from original graph because clustering on adjacent matrix cannot be sophisticated. In addition, users may have difficulty in understanding graph structures in adjacent matrix as in node-link representation since matrix representation is not intuitive when showing structural information. For example, neighbors are not displayed close to each other in adjacent matrix. Third, considering that hierarchy is brought into BGS, only a small subgraph is visualized in most cases, and node-link can effectively display sparse graphs when they have less than million-scale vertices. Thus, we choose node-link representation in BGS system instead of adjacency matrix to represent graphs in visualization.

In graph exploration, ASK-GraphView and Network Explorer are the two visualization tools that are most similar to our BGS. They both focus on exploring a graph interactively by clustering on the graph and navigating along those clusters in top-down manner. The vertices that users are interested are discovered during exploration process. Unfortunately,

on one hand, the hierarchies in ASK-GraphView and Network Explorer are too simple to offer much help. On the other hand, ASK-GraphView and Network Explorer cannot generate crossover links between different layers. The crossover edges are meaningful in attributed graphs because they can show the relation between two nodes at different abstraction layers. Our visualization system provides rich functionality within hierarchy view and supports generation of such crossover edges while expanding or collapsing clusters in graph view. To our knowledge, this is unique feature of BGS.

In interaction, CGV is one of the best interactive graph visualization system because it provides extensive interactions, including dynamic filtering, graph lenses, and some basic interactions, such as zooming, lock/unlock, brushing, expand/collapse clusters etc. Vizster is another interactive visualization software for online social networks, which has some basic interactions, navigation, search, and other functionalities. Such well-designed interactions in above two visualization systems offer great convenience for users to seek graph data. Therefore, we implement most of those interactions and integrate them into BGS.

In summary, by investigating those existing graph visualization systems, we develop a new visualization tool which integrates many state-of-the-art visualization techniques. The BGS can outperform existing visualization systems in scalability, efficiency, and flexibility.

4.3 Methodology

The existing graph visualization systems provide us many techniques to solve various issues in graph visualization. Based on the existing visualization systems, we designed our new visualization software for visualizing large-scale graphs. In this section, we mainly

elaborate new techniques used in BGS and discuss how BGS deals with the issues and challenges in large-scale graph visualization.

4.3.1 Architecture

One major issue in large-scale graphs visualization is the scalability caused by the resource/capacity limits in single machine. To increase the scalability, we attempt to use multiple machines and aim for linear performance gain on the number of machines in graph visualization. Thus, we bring a distributed computing system-Spark into BGS development. Figure 4.1 shows the architecture of BGS. The Spark works on HPC clusters

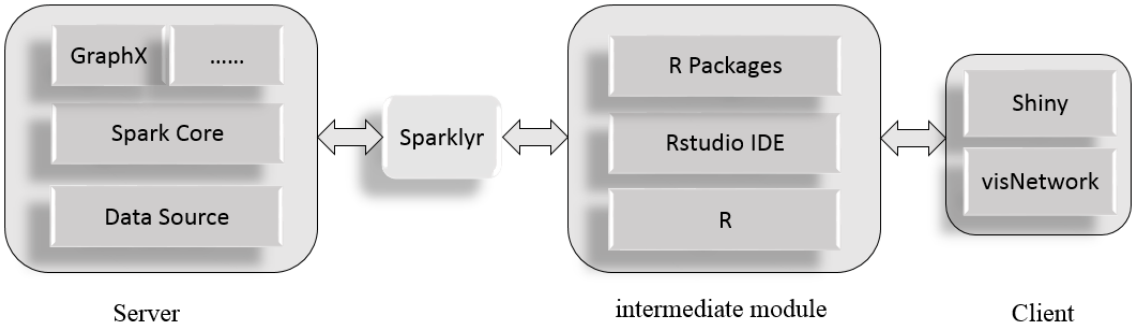


Figure 4.1: Architecture of BGS.

as server (back-end) undertaking heavy computation tasks like clustering, filtering, aggregation etc. Shiny and visNetwork [10] act as the client (front-end) interpreting graph data and displaying graph in WebUI [23]. R and RStudio act as intermediate module that works for the communication between client and server. R is connected to Spark via Sparklyr. Sparklyr is a R package which provides a complete dplyr [102] backend and enable R to manipulate Spark. Shiny and visNetwork both are R packages. The former is a web ap-

plication framework and provides a visualization container for graph, the latter works on graph visualization. Compared to visualization tools running on single machine, BGS has great advantages in scalability because it assigns heavy computation tasks to a distributed computing system which can work in parallel. Also, this architecture allows BGS to utilize all resource across multiple machines, which save huge amount of time to transfer graph data between memory and disk when dealing with large-scale graphs.

4.3.2 Layout

Proper graph layout can help us identify nodes and edges in graph visualization. For example, when visualizing one vertex and its neighbors, star layout is optimal because it can clearly display current node in center and neighbors around; when visualizing several clusters, force-directed layout algorithm is more effective as repulsiveness and attractiveness along connectivity can arrange vertices in a proper distance. BGS provides thirteen different graph layout options which are borrowed from igraph [26, 47, 48]. These layout algorithms can satisfy various layout needs for users. In graph visualization, the graph layouts we provide are computed in real-time, this can be done easily because we set a threshold to control the fan outs of clusters, and only a small subgraph is visualized at any moment at Minimum-Mode in hierarchy view and graph view.

To improve graph layout computational efficiency, we render graphs before stabilization and use straight lines instead of smooth curve for edge shape. Such configuration is helpful in improving interactive experience. In addition, BGS allows users to configure parameters on Barnes-Hut algorithm [13] and forceAtlas2Based algorithm [58] to opti-

mize graph layout. For example, we can adjust arrangement of vertices in visualization by altering gravity value and spring constant value to increase its readability.

4.3.3 Hierarchy

For dense graph or large-scale graphs, some techniques are useful to maintain readability of graph visualization, such as dimensionality reduction [105], layout, and hierarchical abstraction. In BGS, we decide to use hierarchical abstraction for the following reasons. First, since the goal of BGS is to visualize large-scale graphs with billion-scale vertices, hierarchy can greatly reduce the overlaps for very large-scale graphs. In addition, hierarchy support vertical navigation or horizontal navigation by expanding/collapsing clusters to explore the graph. Third, when using hierarchy, only a small subgraph in which users are interested is visualized, which can tremendously reduce expensive layout computation by avoid computing layout on the whole graph. The layout for the small subgraph can be done at rendering stage in real-time.

Generally, hierarchy is generated by clustering on vertices recursively. Clustering techniques can be classified into three categories: divisive algorithms [77, 40] (detect inter-community links and remove them from the network), agglomerative algorithms [81, 97] (merge similar nodes recursively), and optimization methods (maximize an objective function) [20]. Divisive algorithms work from top to bottom by detecting inter-cluster links and removing them recursively. Agglomerative algorithms start from its own singleton cluster, and merge similar clusters recursively. Optimization algorithms usually use modularity value as object function to measure the quality of clustering. They adjust clusters in each

step trying to increase modularity value as high as possible. Network Explorer uses a hierarchical agglomeration algorithm for detecting community structure-NetClustering presented in [24]. Its complexity is $O(nlog_2n)$ for a network with n vertices. ASK-GraphView uses one type of agglomerative algorithm- tuned Markov Cluster Algorithm (MCL) [97] to build hierarchy. Its complexity is $O(|V|^3)$. BGS uses improved Louvain clustering algorithm, which belongs to optimization algorithm. Its complexity is linear with respect to the number of vertices. In addition, Louvain algorithm [20] can be implemented in distributed computing system without much difficulty, which allows us to make clustering on very large-scale graphs.

When using hierarchy in graph visualization, we cannot guarantee vertices are evenly distributed into clusters. It might lead to visualization issue when expanding a cluster which contains lots of vertices. To alleviate the potential issue, we modify the classic Louvain algorithm by controlling its fan out for each cluster, in turn hierarchy depth may increase. Such measures can effectively avoid displaying too many vertices when expanding clusters. This strategy of controlling fan out is also applied in ASK-GraphView when using tuned MCL algorithm. In addition, since our hierarchy generation is done in preprocessing, the tuned Louvain clustering algorithm does not affect visualization efficiency.

As we mentioned before, Louvain clustering algorithm is based on optimization of the modularity value, which uses the modularity value as an object function to measure the quality of clustering. It adjusts clusters step by step aiming to increase the modularity value

as high as possible. Modularity indicates the density of links within clusters as compared to links between clusters. It can be defined as:

$$Q = \frac{1}{2m}\sum_{i,j}[A_{i,j} - \frac{k_i k_j}{2m}]\delta(c_i, c_j) = \frac{1}{2m}\sum_{c}[\sum_{in} - \frac{(\sum tot)^2}{2m}] \tag{4.1}$$

$A_{i,j}$: edge weights between i and j.

k_i: sum of edge weights that come from or go to vertex i

m: $\frac{1}{2}\sum_{i,j} A_{i,j}$

$\delta(c_i, c_j)$: 1 while vertex i and vertex j belong to the same cluster, 0 otherwise.

$\sum_{in}$: sum of weights of edges within cluster c

$\sum_{tot}$: sum of weights of edges of whole cluster c

While adjusting clusters, the change of modularity value can be calculated by the following equation:

$$\Delta Q = [\frac{\sum in + k_{i,in}}{2m} - (\frac{\sum tot + k_i}{2m})^2] - [\frac{\sum in}{2m} - (\frac{\sum tot}{2m})^2 - (\frac{k_i}{2m})^2] \tag{4.2}$$

The classic Louvain clustering algorithm was proposed in [], which can be described in the following steps:

The classic Louvain clustering algorithm was proposed in [10], which can be described in the following steps:

- Initially, each vertex in the network is assigned an individual community. The number of vertices is the same as the number of communities
- Second, for each vertex, tentatively moves it to its neighbor communities and evaluates how the modularity will gain. If the gain is positive, move the current to the

neighbor community where the gain is maximum. If no positive gain exists while moving to its neighbor communities, then it stays in the current community.

- Third, repeat step 2 until no further improvement can be made in modularity value.
- Fourth, compress the network created in step 3. Each community will be replaced by one vertex. The weight of the links between communities can be calculated as the sum of edge between vertices in the corresponding two communities. The edges within the same community become self-loops for the new vertex.
- Fifth, repeat step 2 to step 4, a hierarchy structure can be built by clustering at each level.

We improved the above Louvain clustering algorithm via controlling the number of nodes in each community in the second step.

4.3.4 Graph Data Definition

Our visualization system operates on undirected and directed graphs $G = (V, E)$ where V and E represent the set of vertices and edges respectively. The hierarchy is generated from the original graph G recursively. If each layer of the hierarchy denotes $Gi(Vi, Ei)$, then $G0(V0, E0)$ is $G(V, E)$, and $Gi(Vi, Ei)$ is abstracted from $Gi-1(Vi-1, Ei-1)$. For hierarchy tree, we define the following concepts:

- T: the whole hierarchy tree
- T_i : the subtrees at i^{th} level

- Leaves (T): set of leaves of T. Leaves (T) = V_0 = V.

- Children (T_i): the children of subtree T_i . $Children(T_i) = V_i$, $Children(T_0) = V_0 = V$

Layers G_i describes layer information. Tree T_i defines vertical information. $(T_i, G_i), 0 <= i < h$ consists of the whole hierarchy of the graph, where h is the depth of the hierarchy.

4.3.5 Visualization

After clustering on original graph and generating hierarchy data, BGS will load the hierarchy data into Spark for visualization. In BGS, hierarchy view and graph view both are provided. For hierarchy view, BGS provides hierarchy expansion, hierarchy search, and hierarchy selection. For graph view, we are also allowed to do graph expansion, graph search, and graph selection. In both views, some useful decorations and interactions are presented in BGS, which aid us in graph exploration. The visualization techniques of BGS are summarized in Figure 4.2. The following sections will discuss each functionality in detail.

4.3.5.1 Hierarchy View and Graph View

Hierarchy view is an approach to visualize part of the hierarchy generated from original graph. Hierarchy view only provides vertical links amid clusters or nodes at different layers, instead of horizontal links. Graph view, on the contrary, only offers horizontal links or reduced horizontal links among clusters or nodes. Clus-ters vertical information is absorbed by their children with expansion in graph view. Hierarchy view offers us high

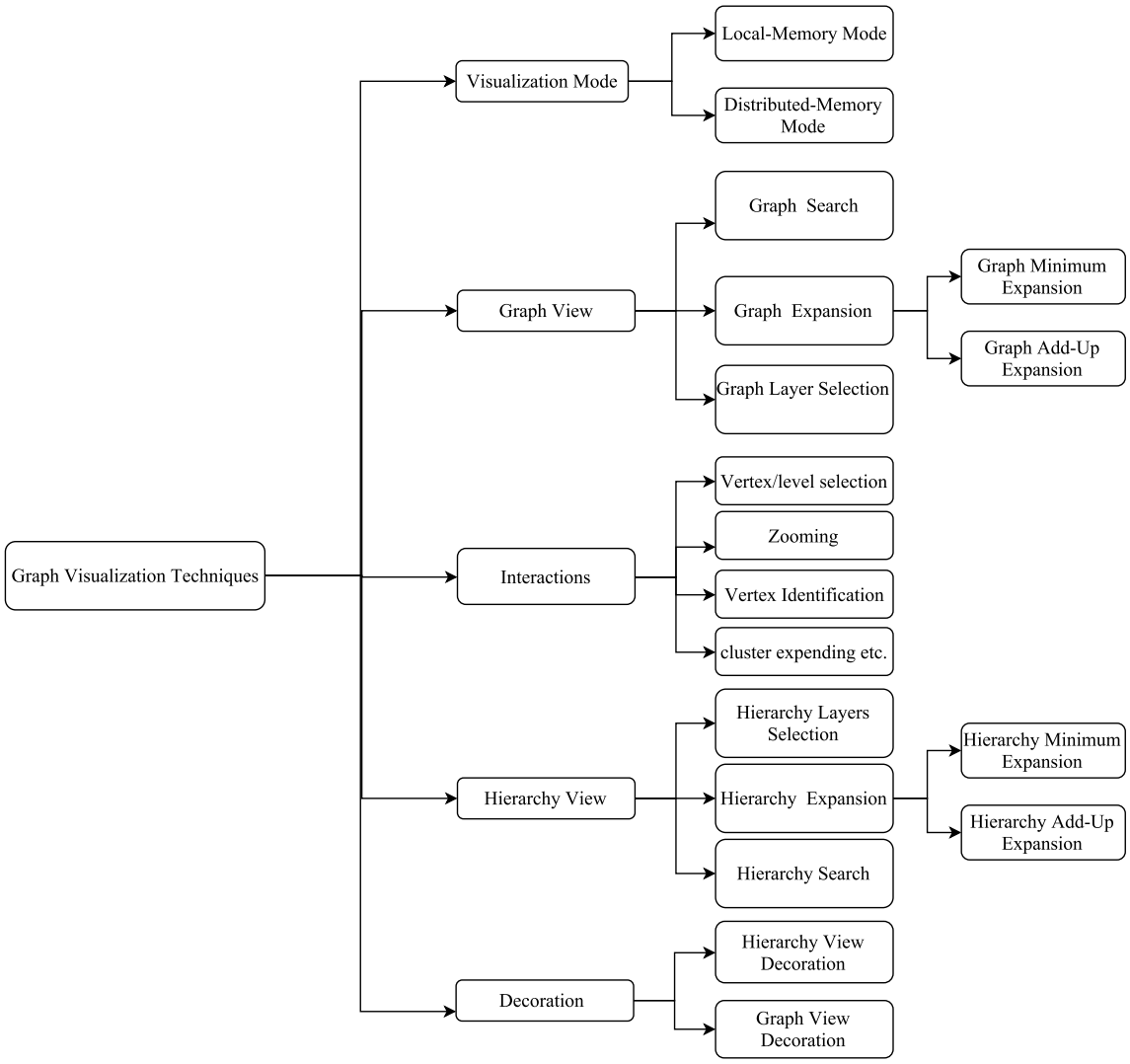


Figure 4.2: Visualization techniques of BGS

level abstractions of the original graph. More im-portantly, hierarchy view can easily locate interesting nodes, which can help users to find the correct clusters to expand to reach the interesting nodes in graph view. Hierarchy view and graph view work together coordinately to display whole graph data.

4.3.5.2 Expansion Mode

In order to satisfy users different demands in graph visualization, we design two expansion modes (Mini-mum mode and Add-Up mode) for hierarchy view and graph view in BGS based on different principles. In Minimum mode, BGS allows users to focus on current expanded clusters or nodes. The previously expanded clusters or nodes will be automatically collapsed into a cluster that is a sibling of the cluster/node or a sibling of its predecessors. In this mode, only one cluster or node is permitted to reach lower layers of the hierarchy at one time, which maintains high efficiency in large-scale graph visualization. In Add-Up mode, BGS allows users to focus on multiple expanded clusters or nodes. The previously expanded clusters or nodes will be preserved instead of collapsed. In this mode, users can ob-serve detailed relations amid multiple clusters or nodes. Minimum mode and Add-Up mode are offered in both hierarchy view and graph view, which can serve users fundamental visualization requirements.

4.3.5.3 Graph View Mode

In addition to two expansion modes, BGS also provides two other modes (Regular mode and Edge-Free mode) in graph view for dealing with sparse or dense graphs. Regular mode is a simple mode, which works for sparse graphs. When visualizing a graph, the links

among vertices are visualized in this mode. Edge-Free mode is designed for very dense graphs. In this mode, the links appear only when users request displaying edges adjacent to one vertex, which can significantly reduce the overlaps within dense graph in visualization.

4.3.5.4 Hierarchy Exploration

Hierarchical structure represents graphs abstraction at different levels, which shows which clusters or nodes belong to which group or cluster. In an attributed graph, the hierarchy may have specific meaning at each level. For example, in the flight graph in Section 4, flights can be regarded as graph edges which connect two different airports. For each flight, it has some related information, such as departure airport, departure city, departure country, departure continent, arrival airport, arrival city, arrival country, and arrival continent. From the fight graph, we can obviously abstract it at four levels: airport level, city level, country level, and continent level. For international flights, we can observe it at country level or even continent level, which shows the connection from one country to another or from one continent to another. For domestic flights, we focus on city level, from one city to another city. From the hierarchical structure, we can easily find graph nodes-airports. Hierarchy exploration includes hierarchy layer/level selection, hierarchy expansion, and hierarchy search.

a) Hierarchy Layers Selection

When exploring graph hierarchical structures, users probably do not want to start with only one top level cluster because it cannot convey much background in-formation for users. BGS deals with such problem by allowing users to set several top levels for obser-

vation at the beginning. If one hierarchy has depth h, and the ini-tial hierarchy has s layers, then the initial hierarchy is $Ti, h - s + 1 < i <= h$ which provides informative context for users to explore the graph hierarchy. Also, the several top levels in the hierarchy will consistently exist with expanding clus-ters. For example, Figure 4.3 shows selecting top two layers in hierarchy view.

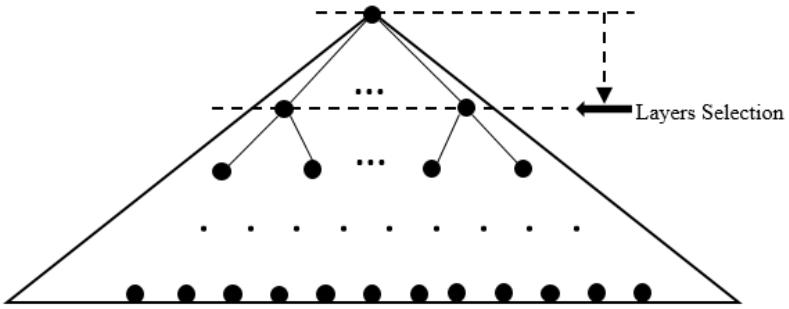


Figure 4.3: Hierarchy layer selection.

b) Hierarchy Expansion

Hierarchy expansion is a major approach to find out where one node or cluster stays in the hierarchy, which provides a top down manner to explore graph hierarchical structure. BGS has two hierarchy expansion modes: Minimum hierarchy expansion and Add-Up hierarchy expansion. In order to illustrate the two modes, one simple graph hierarchy is used in Figure 4.4 to explain the two concepts. Different layers can be differentiated in different colors (red: layer 3; purple: layer 2; green: layer 1; blue: layer 0).

As we mentioned before, there are two hierarchy expansion modes: Minimum hier-archy expansion and Add-Up hierarchy expansion. In Minimum mode, which is demon-

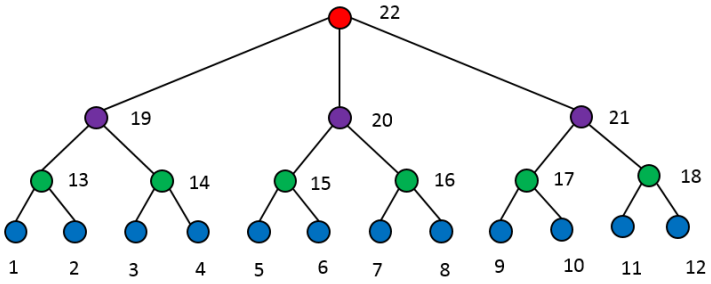


Figure 4.4: An example graph hierarchy.

strated in Figure 4.14a, initial hierarchy layers selection is top 3, when expanding one cluster (node 13), there will be out-going links generated from the cluster to connect its children (edge from 13 to 1, from 13 to 12). Previous expanded cluster (node 16) whose children (node 7 and 8) do not belong to its siblings or its predecessors and their siblings will be collapsed if previous expanded cluster does not belong to initial hierarchy (node 7 and node 8 are collapsed into node 16, node 16 belongs to initial hierarchy). Minimum hierarchy expansion only allows one cluster/node, its siblings, and its predecessors and their siblings in hierarchy to be visualized.

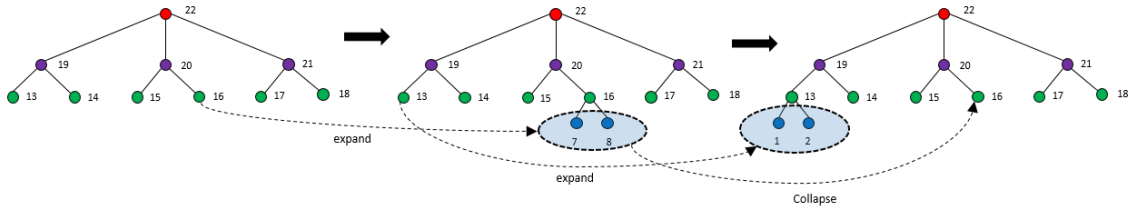


Figure 4.5: Hierarchy expansion at Minimum mode.

In Add-Up mode, for example in Figure 4.10, when expanding one cluster (node 13), just as Minimum mode, it will create out-going links from the cluster to connect its chil-

dren (edge from 13 to 1, from 13 to 2), but previous expanded clusters children (node 7 and node 8) will be always maintained, even though they are not siblings of the currently expanded cluster (node 13), or predecessors and their siblings of the currently expand-ed cluster. Add-Up mode allows multiple clusters/nodes, their siblings, and their predecessors in hierarchy to be visualized.

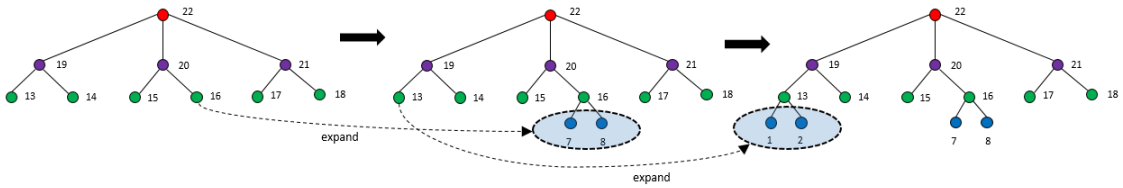
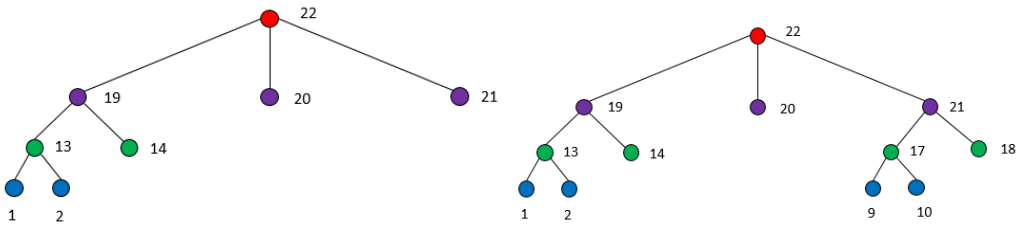


Figure 4.6: Hierarchy expansion in Add-Up mode.

c) Hierarchy Search

Hierarchy search is designed to display one node and its hierarchy path from root. The hierarchy path can tell users where the destination node is, how to identify the node, and which cluster to expand in graph view. When users have no background in hierarchy abstraction, hierarchy search becomes necessary and indispensable to explore a graph. In hierarchy search, hierarchy path is generated based on a node index/attributes. Likewise, hierarchy search also has two modes: Minimum mode and Add-Up mode. Minimum mode only allows one input of node information. At Add-Up mode, users can search arbitrary number of nodes. For example, in Figure 4.7, Figure 4.5 search node 1 or 2 at Minimum mode, Figure 4.7b search node 1 or 2, and 9 or 10 at Add-Up mode.



(a) Hierarchy search in Minimum mode (b) Hierarchy search in Add-Up mode

Figure 4.7: Hierarchy search in two modes

4.3.5.5 Graph Exploration

Graph exploration is a core part of BGS, which provides graph views at different layers. When expanding clusters, there will be links generated across multiple layers, called crossover edges. Crossover edges are extremely important links when we make an abstraction on an original graph. It conveys different meanings with edges in original graph. For example, in flight data, the original graph shows connections between airports. Crossover edges can represent connections between airport to city, airport to country, airport to continent, city to country, city to continent, or country to continent. From crossover edges in graph view, we can straightforwardly answer such questions as: whether we can travel from one airport to another city, country, or continent? Whether we can travel from one city to another country, or continent? Whether we can travel from one country to another continent? All the answers can be found in graph view in the form of crossover edges. Graph Exploration is a crucial aspect of BGS that includes graph layer selection, graph expansion, and graph search.

a) Graph Layer Selection

Initially, BGS starts with the top layer graph G_h (h is the depth of the hierarchy) at graph view. In order to help users quickly identify interesting vertices, users are permitted to select another starting layer G_i to visualize. For example, in Figure 4.8, the third layer is chosen as the starting layer. Based on this layer, users can expand clusters recursively to navigate down layer by layer. Graph Layer Selection is different from hierarchy selection, which selects several top layers, but only one layer is chosen in graph layer selection.

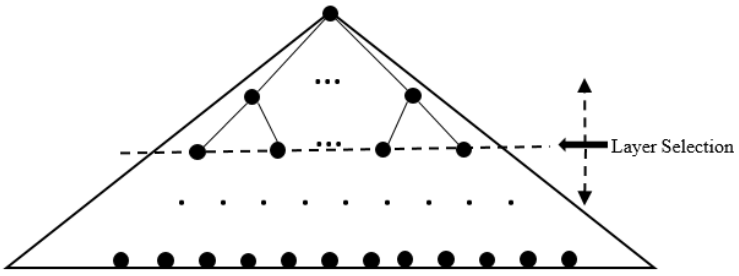


Figure 4.8: Graph layer selection.

b) Graph Expansion

Graph expansion is a fundamental measure to navigate down along the hierarchical structure. During the expansion, users can observe the hierarchy abstractions and their relations in a top-down manner, until they reach the destination node. At this moment, graph view is displaying overall information about the target node, including vertical information and horizontal information. The vertical information refers to the relation between the target node with upper layer clusters, or even down layer nodes/clusters if the target node is not a leaf node. The horizontal information denotes the relation between the target node and its neighbors at the same layer.

In graph expansion, one of the challenging tasks is to determine whether crossover edges exist between the target clusters children and the target clusters neighbors. To check the crossover edges, all pairs of neighbors and children are reduced to the same layer. Like hierarchy expansion, graph expansion has two view modes: Minimum mode and Add-Up mode. In Minimum mode, demonstrated in Figure 4.9, when expanding one cluster (cluster 20), it will be replaced by its children (node 15 and 16). Previously expanded clusters (node 21) whose children (node 17 and 18) do not belong to the new expanded clusters (cluster 20) siblings, or its predecessors and their siblings will be collapsed into one sibling (cluster 21) of the new expanded cluster, or one sibling of the new expanded clusters predecessors. Minimum mode only allows one cluster, its siblings, and its predecessors and their siblings to be visualized.

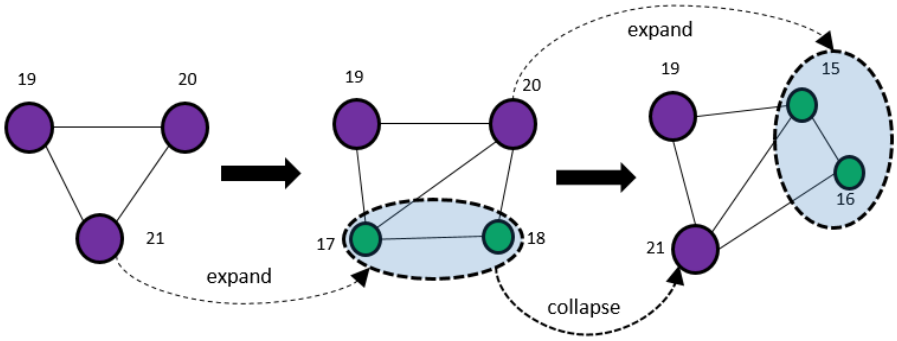


Figure 4.9: Graph expansion in Minimum mode.

In Add-Up mode, demonstrated in Figure 4.10, when expanding one cluster (cluster 20), it will be replaced by its children (node 15 and 16). For previously expanded clusters (cluster 21), their children (node 17 and 18) are always retained, even though they are not

siblings of the newly expanded cluster (cluster 20), or predecessors or the siblings of the newly expanded cluster. Add-Up mode allows multiple clusters/nodes, their siblings, and their predecessors in the hierarchy to be visualized.

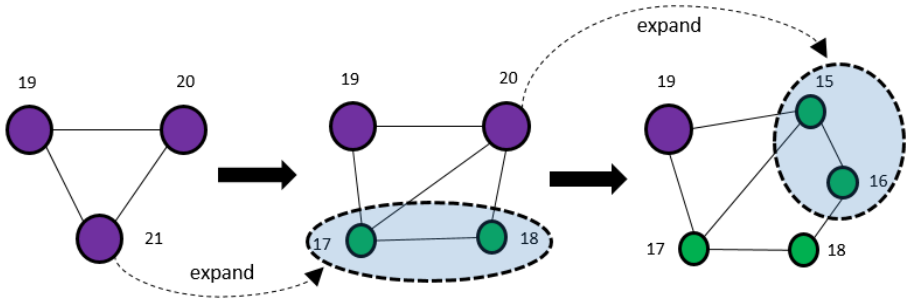


Figure 4.10: Graph expansion in Add-Up mode.

c) Graph View mode

In BGS, we designed two graph view modes: Regular mode and Edge-Free mode. Since Regular mode is simple and easy to understand, here we only discuss Edge-Free mode. When visualizing a dense graph where almost each vertex is connected to all other vertices, it usually causes lots of hairballs in graph view. Users can hardly see any details from the hairballs. To solve this issue, BGS provides Edge-Free mode for dense graphs. In this mode, illustrated in Figure 4.11, initially, only vertices are visualized in graph view, all edges within the vertices are hidden (Figure 4.11 a)). If users want to observe the links adjacent to one vertex, just clicking on the vertex reveals the edges adjacent to that vertex. This is demonstrated in Figure 4.11 with no edges clicked (Figure 4.11 a), vertex 21 selected (Figure 4.11 b) and vertex 20 selected (Figure 4.11 c). Also, previously generated

edges adjacent to previously selected vertices disappear. At any moment, only edges adjacent to one node can be visualized. This measure not only can avoid hairballs and increase readability in graph view, but also improve visualization efficiency.

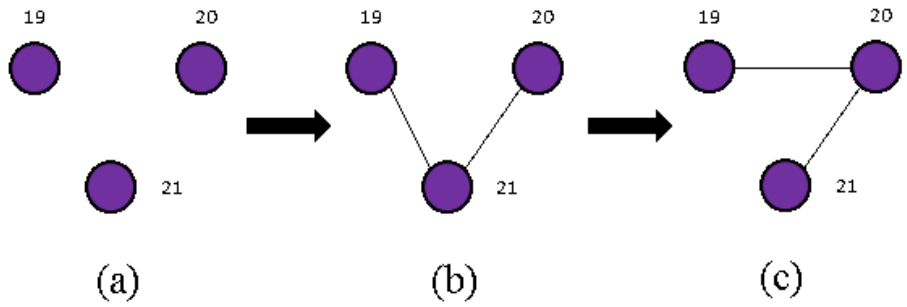
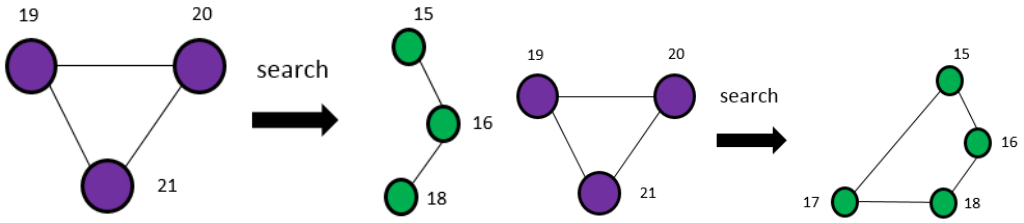


Figure 4.11: Graph View in Edge-Free mode.

d) Graph Search

Graph search in BGS can be regarded as one-step probing of nodes or edges. When visualizing a large-scale graph, we will probably have difficulty in finding target node if we start from starting graph view. Even if users have hierarchy background information, they still have to expand clusters to navigate down to find the target node. BGS supports probing vertex/vertices or edge/edges by index or its/their attributes. If users already have target vertices or edges, the identification of such vertices or edges in large-scale graphs is greatly facilitated. In BGS, users can identify one vertex/edge, or more vertices/edges. BGS will show the target vertices/edges and their neighbors. If two target vertices have common neighbors or two target edges have common vertices, the probing results are connected.

For example, in Figure 4.12, Figure 4.12a search node 16, Figure 4.12b search node 16 and 18.



(a) Graph search on one node case (b) Graph search on two nodes case

Figure 4.12: Graph search for two cases

The above sections talked about Minimum mode and Add-Up mode on hierarchy exploration and graph exploration. The two modes are designed according to different principles. Minimum mode is designed to focus on one target cluster/node, other irrelevant information is to be abstracted at high level. This measure not only can effectively reduce overlap in visualization but also increases visualization efficiency. Add-up mode allows us to focus on two or more clusters or nodes at one time so that users can easily find their relations or make comparisons between such clusters or nodes in hierarchy view or graph view. Edge-free mode is trying to avoid hairballs in graph view, only those edges in which users are interested are shown. These measures not only can effectively reduce overlap in visualization but also increases visualization efficiency.

4.3.5.6 Visualization Mode

When expanding clusters in graph view, one huge computing task is creating crossover edges. If one cluster has n neighbors and m children, there will be $m*n$ potential crossover edges to be generated. Since Spark provides the back-end for BGS, all vertices, edges and hierarchy data are stored in Spark. When generating crossover edges, R needs to send $2 * d * m * n$ request to obtain required graph data (d is the average layer distance between clusters neighbors and children), which causes tremendous communication overheads. To solve this issue, we present two visualization modes on BGS: Local-Memory mode and Distributed-Memory mode.

a) Local-Memory mode

Local-Memory mode is designed for small graphs. When visualizing a small graph, if graph data can be completely loaded into main memory of the local machine, BGS will do this before the graph view rendering. Crossover edge generation is done on local machine. Thereby, great communication overheads can be avoided.

b) Distributed-Memory mode

Local-Memory mode only works well on small graphs, so we designed another visualization mode, Distributed-Memory mode, for large graphs. In this mode, the graph and its hierarchy data are distributed into multiple machines instead of the local machine. To minimize the data requests to Spark, we must first figure out what graph data is really needed. When expanding clusters, only vertices, edges, and hierarchy on the clusters neighbors and children are used in crossover edge generation. Second, we retrieve exact graph data from Spark only once. In this way, the number of data requests can be reduced to $d * m * n + 2$.

This measure makes BGS only keep a small necessary graph data in local memory for rendering, which not only can increase BGSs efficiency but also maintain its visualization scalability.

4.3.5.7 Decoration and Interaction

To increase readability, BGS provides some decoration to modify hierarchy view and graph view and interactions help us explore details in both views. For graph view, BGS allows us to change vertex shape, edge shape, graph layout, etc. For hierarchy view, we can adjust level separation, hierarchy direction, layout etc. These decorations are helpful to increase the readability for both views.

From Shneidermans visualization principle, we can realize the importance of Interaction in graph exploration. According to BGSs visualization characteristics, we provide the following interactions for BGS.

a) Zooming in/out

Zooming, operated by mouse, is very useful interaction to adjust the viewpoint to focus on certain vertices or edges, which is fundamental to graph visualization interaction. Before zooming, we first need to find a focus, then zoom in or out by scrolling with the mouse. In conjunction with dragging and moving nodes, zooming can help us find details in graph view and hierarchy view.

b) Vertex identification

When many nodes are visualized in graph view, it may be not easy to find where the target node is. BGS provides vertex identification by telling the system which vertex users

want to focus, then the viewpoint will move to the target node. In the functionality, users can then tune zoom factors to make the viewpoint a proper distance.

c) Vertex selection and layer selection

Graph selection refers to highlighting one vertex or group of vertices in the visualization interface. Typically, the selected vertex or group of vertices are exhibited in a different color to differentiate with other unselected subgraphs. For example, when visualizing a social network, users can select one person or a group of people in whom users are interested. Only such selected people and their relations become noticeable. This functionality is beneficial for visualization and makes us focus on specific information.

4.4 Case Study

To illustrate BGSs functionalities and scalability, we present three case studies to find out what BGS can achieve. The three graph datasets are Facebook, Friendster, and Flight (see Table 4.1). The Facebook graph is a small social network. Friendster is a large graph with more than 1 billion edges. These three datasets can cover most cases: small graph and large graph, attributed graph and non-attributed graph, where the functionalities in BGS will be evaluated. In both Facebook and Friendster graphs, vertices represent users, edges refer to their friendship between two users. For privacy reasons, vertices are labeled with numbers as identification. Edges are abstracted as pairs of numbers. The flight graph is an attributed graph which represents flights from one airport to another. Each airport has some attributes, for example, airport name, city, country, and continent. The case studies are to verify the usage of BGS in various scenarios. We will observe BGSs performance on

functionalities, scalability, and interactions. BGS can be thoroughly evaluated from these three aspects.

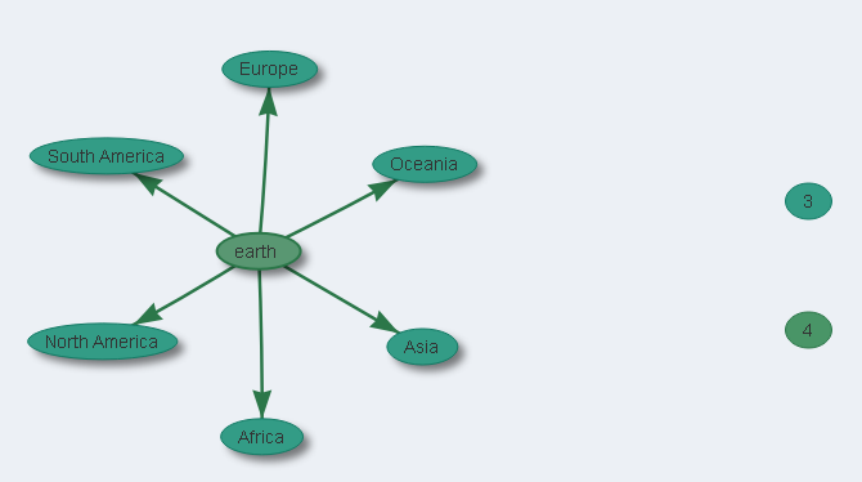
Table 4.1: Graph datasets for visualization

Graph	Vertices	Edges	Attributed	File Size
Facebook	4,039	88,234	No	1MB
Flight	3,125	58,568	Yes	1MB
Friendster	65,608,366	1,806,067,135	No	30GB

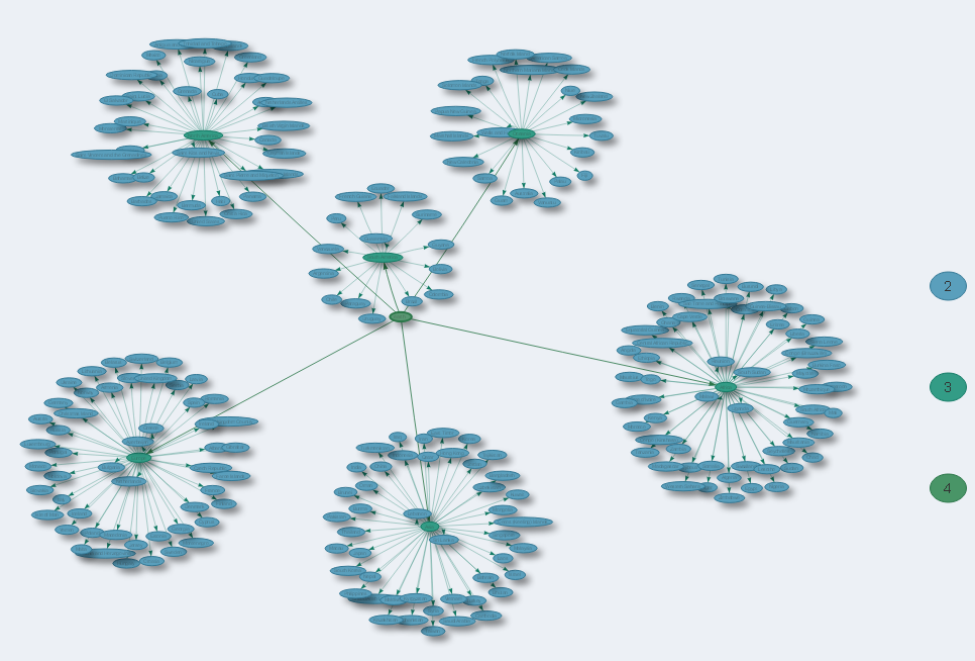
4.4.1 BGS Functionalities

a) Hierarchy layers selection

The hierarchy layers selection refers to allowing users to choose several top layers to visualize in the initial hierarchy structure, which provides the fundamental background information for users. Figure 4.13 is two views of hierarchy layers selection on Flight data. Figure 4.13a shows the top two layers; we can expand hierarchy based on the continent layer to search for country, city, or airport. For example, if the target cluster/node is the United States, then the node of North America should be expanded. If we seek airports at Atlanta, we will continue to expand the node of the United Sates. Figure 4.13b shows the top three layers of the hierarchy, which reaches country layer. In Figure 4.13b there are six clusters representing six continents. From this hierarchy view, users can expand one country to look for cities or airports.



(a) Top 2 layers



(b) Top 3 layers

Figure 4.13: Hierarchy layers selection of Flight Graph

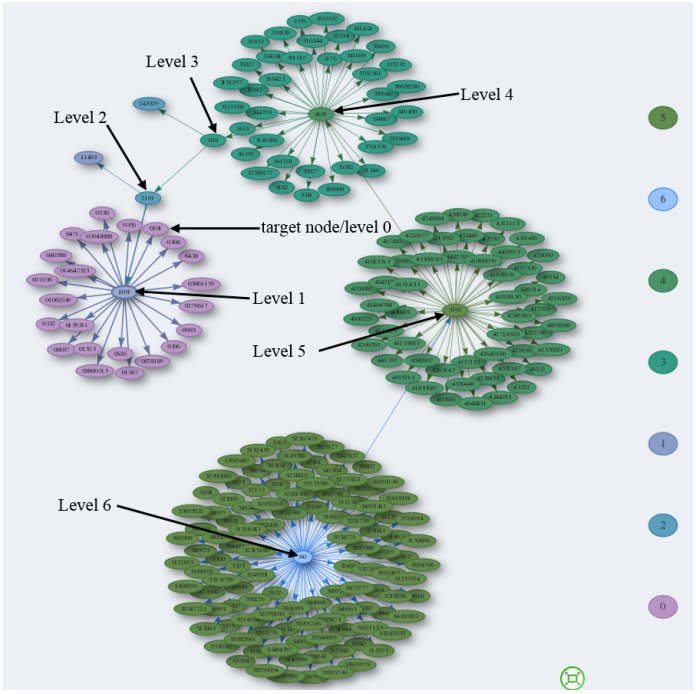
How many layers should be selected in initial hierarchy view depends on users. Fewer or more layers selected both have merits and drawbacks. If more layers are selected in the initial hierarchy view, it can convey more abundant information to users, but it may cause a burden for visualization and lead to many overlaps. If fewer layers are selected in the initial hierarchy view, it can reduce overlaps in visualization, but less information can be found in the initial hierarchy view.

b) Hierarchy View

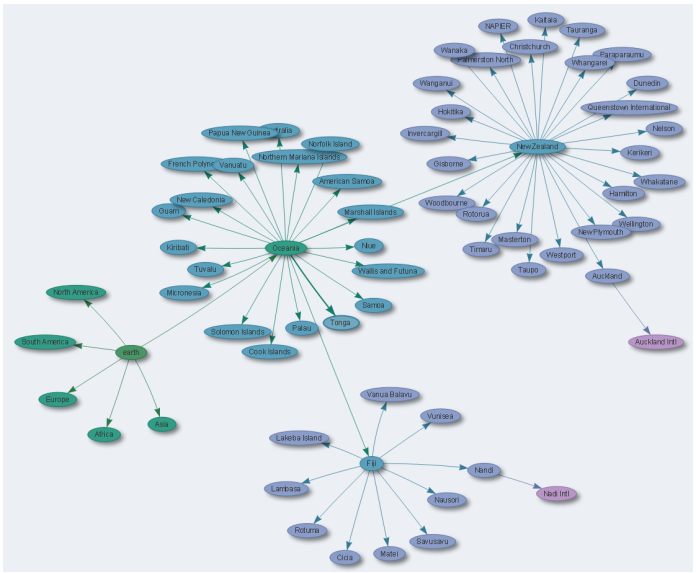
Figure 4.14a is the hierarchy view on Friendster graph in Minimum mode. Figure 4.14b is the hierarchy view on Flight graph in Add-Up mode. The two modes of hierarchy view are used in different scenarios. If users want to search hierarchy for one node, then Minimum mode is a better choice because it only shows the minimized hierarchy, irrelevant hierarchical structures are collapsed, which improves visualization efficiency and reduces overlaps. For example, Figure 4.14a displays the hierarchy of node 101, from level 6 to level 0. Only node 101, its siblings, its predecessors, and their siblings are displayed. If users wish to observe the hierarchy including two target nodes, Add-up mode is more suitable since it can show the combination of two hierarchies, which allows us to explore some insights from the hierarchy easily. For instance, Figure 4.14b is showing hierarchy of Nadi international airport and Auckland international airport. From the hierarchy we know both air-ports belong to different countries but both are located in Oceania.

c) Graph View

Figure 4.15 shows the graph view of Flight data. From the initial graph view, we expand the cluster North America, the cluster Mexico within North America, Mexico City within



(a) Hierarchy view of Friendster Graph in Minimum mode



(b) Hierarchy view of Flight Graph in Add-Up mode

Figure 4.14: Hierarchy view of Flight Graph at Minimum mode and Add-Up mode. In Add-Up mode, leaf Nodes: Nadi International Airport (Nandi, Fiji, Oceania) and Auckland International Airport (Auckland, New Zealand, Oceania)

Mexico, until Licenciado Benito Juarez International Airport is located. Since we use the interaction of vertex selection in the graph view, the node of Licenciado Benito Juarez International Airport and its direct neighbors are highlighted, other irrelevant nodes/clusters become gray. To zoom in on Licenciado Benito Juarez International Airport, the links starting from the airport demonstrate which continents, countries, and cities the airport can reach with non-stop flights, which illustrates the significance of crossover edges in graph view.

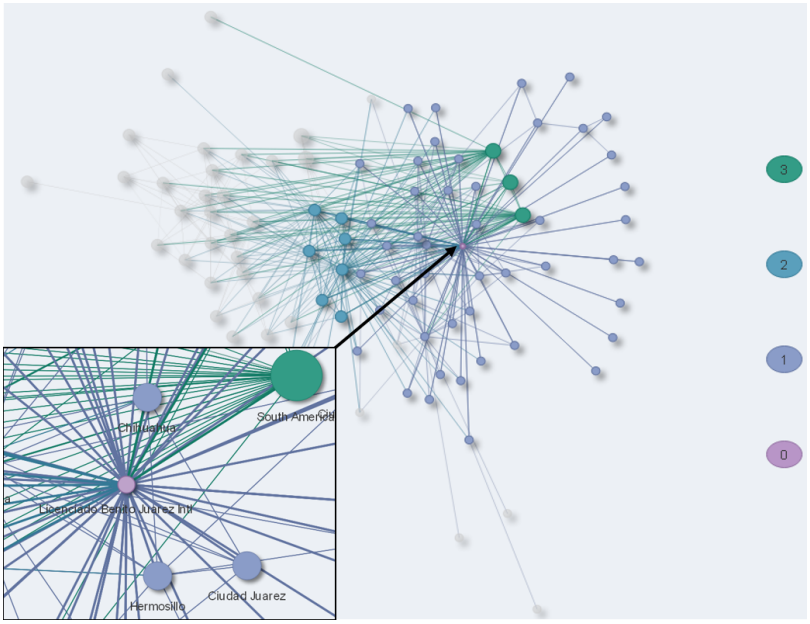


Figure 4.15: Graph view of Licenciado Benito Juarez International Airport (Mexico City) from the Flight graph

d) Graph Search

Graph search in BGS not only includes vertex search but also edge search. In vertex search, for single node search or multiple nodes search, node id or node attributes are

accepted. In edge search, single edge search or multiple edges search, edge id or edge attributes are taken by BGS as well. When doing a search on a single node, it will show one node and its neighbors. When doing a search on multiple nodes, the two nodes and their neighbors and common neighbors are displayed. Also, common neighbors are incorporated in the results. For example, Figure 4.16 shows the search result from Jackson Evers and Birmingham international airport on Flight graph. Their common neighbors are displayed in the middle. Graph edge search works based on the same principle. Graph search is an effective approach to explore original graph in one step. It is important when visualizing a very large graph.

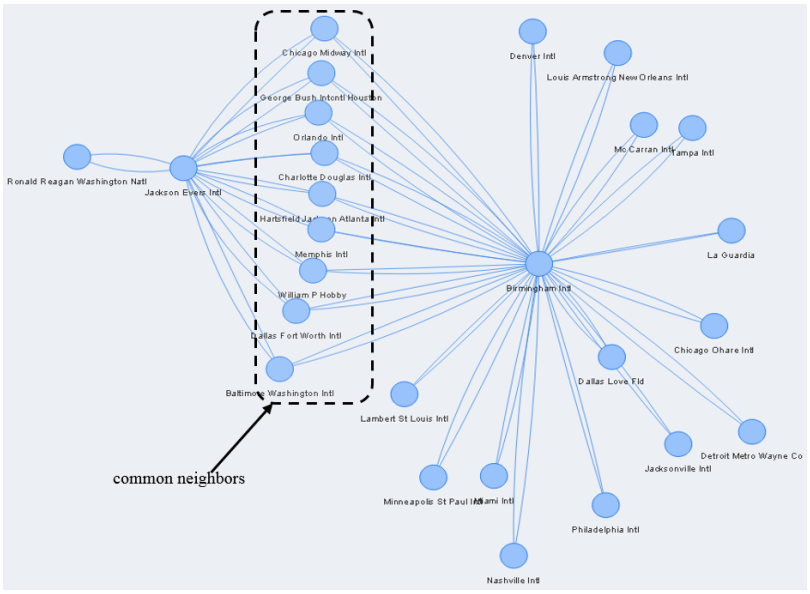


Figure 4.16: Graph search view of Jackson Evers international airport and Birmingham international airport from the Flight graph

e) Graph View Mode

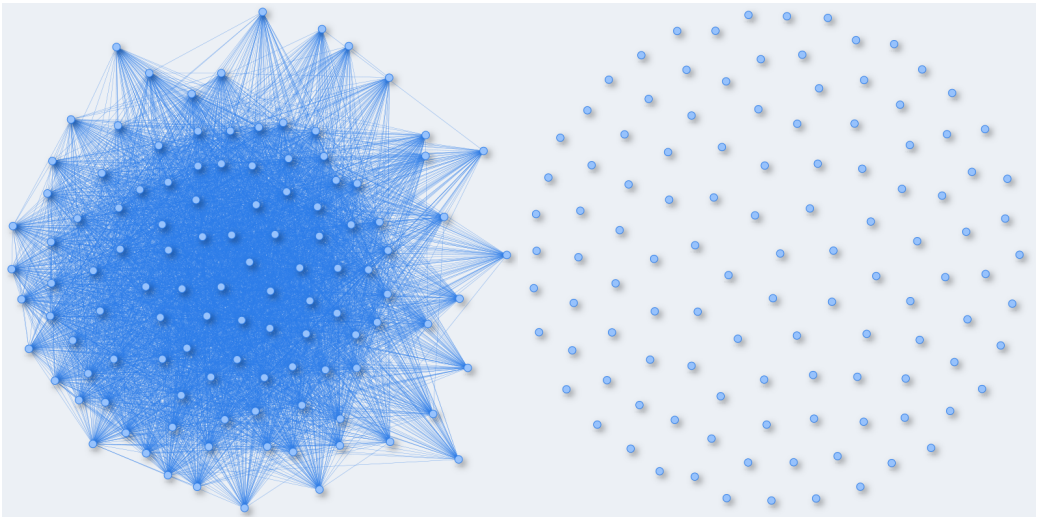
Figure 4.17 shows the graph view of Friendster data at top layer with Regular mode Figure 4.17a and Edge-Free mode (Figure 4.17b and Figure 4.17c). In Regular mode, Figure 4.17a has 113 vertices and 5603 edges. From such dense graph, we can hardly see any details due to the hairballs. Figure 4.17b shows the initial graph view of same graph in Edge-Free mode. When clicking on the node (5101), edges adjacent to the node (5101) are shown in Figure 4.17c. BGS allows us to observe other edges adjacent to one node by clicking the node. Compared Figure 4.17a and 4.17c, Edge-Free mode is obviously very effective in reducing over-laps in graph view and increasing its readability.

f) Graph Layer Selection

Graph layer selection is one method to change the initial graph view. By moving through layers from top to bottom along the hierarchy, more and more vertices are visualized, and background information changes as well. For example, in the flight data, the graph layer can be set at continent level, country level, or city level. Figure 4.18 shows graph layer selection at layer 4 to layer 1.

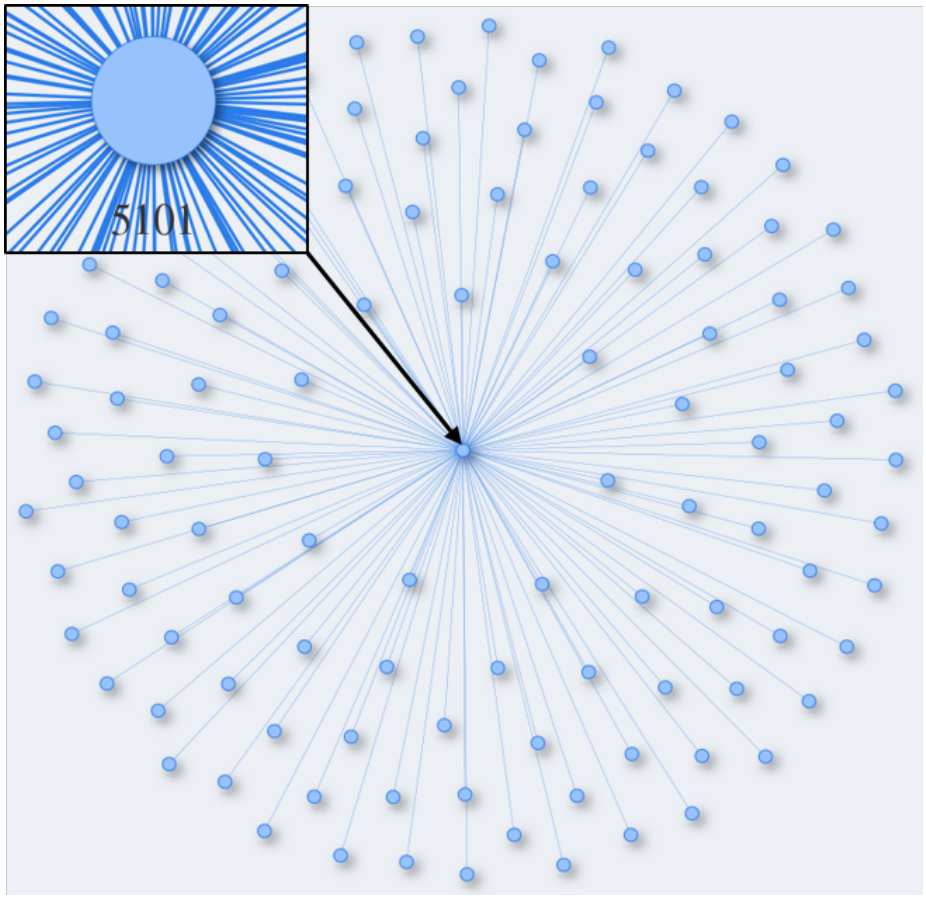
4.4.2 BGS Scalability

From the case studies, our visualization system can easily visualize graphs with billion scale edges. Table 4.2 shows the clustering time, loading time, and visualization time for the three graphs using BGS. Since Facebook and Flight are small graphs, we use Local-Memory mode in order to obtain high efficiency. For Friendster graph, we can only use Distributed-Memory mode because it cannot be loaded into local memory because of large graph size. Compared to ASK-GraphView, BGS has good visualization efficiency due



(a) Regular mode

(b) Initial Edge-Free mode



(c) Edge-Free mode on node 5101

Figure 4.17: Comparison of graph view mode on Friendster

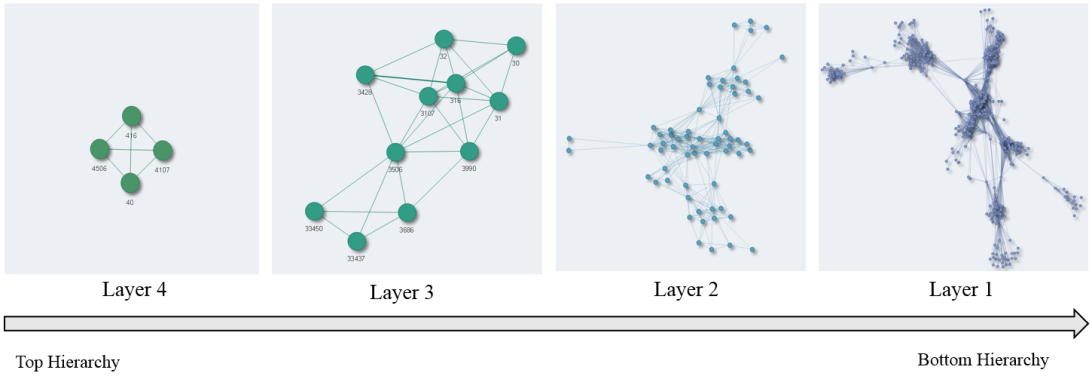


Figure 4.18: Graph layer selection on Facebook graph

to its distributed architecture, which fundamentally differentiates from ASK-GraphView. ASK-GraphView creates separate files in disk when dealing with large-scale graphs. In BGS, however, the whole graph data is kept in distributed memory or local memory, it can quickly retrieve data from Spark and visualize the graph in real time. Thus, BGS has a significant improvement over ASK-GraphView in efficiency. Similarly, BGS is a universal graph visualization tool which has no restrictions on the graphs structure and density. This feature increases BGSs flexibility in application.

4.4.3 Interactions

When using BGS on the Facebook graph and Flight graph, we can instantly interact with BGS, for example, zooming in or out, vertex identification, vertex selection or level selection, clustering expanding etc. For the Friendster graph, we can feel a little delay in interactions with BGS using distributed memory mode for such a large graph. It takes some time to generate crossover edges when expanding clusters.

Table 4.2: Clustering time, loading time, and visualization time for graph datasets

Graph	Clustering	Loading	Visualization Delay	Visualization Mode
Facebook	49 s	1 min	1-3 s	Local Memory
Flight	--	50 s	1-3 s	Local Memory
Friendster	4.2 h	3 min	about 20 s	Distributed Memory

* Flight data is clustered by geographical location

In summary, BGS achieves our desired goal. Specifically, users can visualize graphs through hierarchy view and graph view while using BGS, and get some meaningful insights from the exploration.

4.5 Discussion

Since BGS provides two expansion modes, two graph view modes, and two visualization modes, when we apply BGS to various graphs, there will be eight combination modes. Table 4.3 shows the combination of visualization mode and expansion mode and their characteristics. Generally, Minimum mode has constantly efficiency. In Add-Up mode, efficiency gradually decreases with expanding more clusters. Regular mode works well for sparse graph. Edge-Free mode can avoid hair-balls for dense graph. Local-Memory mode has high efficiency, but is only fit for small graphs. Distributed-Memory mode is fit for large-scale graphs, but its visualization efficiency is not as good as Local-Memory mode. Table 3 can guide users to choose proper combination mode in graph visualization. From this table, they can choose optimal expansion mode and visualization mode according to visualization requirements and graph size. Proper expansion mode, visualization mode,

and graph view mode not only can enhance readability but also improve visualization efficiency

Table 4.3: Combination of visualization mode and view mode

Mode	Minimum mode	Add-Up mode
Local	1) Fit for small-scale graph; 2) Constantly high efficient; 3) Constantly high requirement for local memory;	1) Fit for small scale graph; 2) Relative high efficient; 3) High requirement for local memory;
Distributed	1) Fit for large-scale graph; 2) Constantly relative high efficient; 3) Constantly low requirement for local memory;	1) Fit for large-scale graph; 2) Low efficient; 3) Low requirement for local memory;

From this table, they can choose optimal view mode and visualization mode according to visualization requirements and graph size. Proper view mode and visualization mode not only can enhance readability but also improve visualization efficiency.

4.6 Conclusion

In this chapter, we propose a scalable graph visualization system that aims to visualize large-scale graphs efficiently. BGS is developed on Spark, R, RStudio, and Shiny. Spark is a distributed computing framework which acts as backend in BGS working on cluster-

ing and visualization. This architecture brings BGS great improvement on scalability and efficiency.

In visualization, BGS has hierarchy view and graph view. Hierarchy view shows a series of high level abstractions, which aids users to seek the correct clusters to expand in graph view. In hierarchy view, we can do hierarchy expansion, hierarchy search, and hierarchy selection. Likewise, graph view has graph expansion, graph search, and graph selection. In both views, some useful decorations and interactions are provided.

In addition, BGS has two expansion modes in hierarchy expansion and graph expansion, two graph view modes in graph view, and two visualization modes. We provide a summary of four combination modes in Table 4.3, which offers us a guideline to opt for proper mode in application.

This paper conducts three case studies, which cover the scope of small graph, large graph, attributed graph, and non-attributed graph. The study shows that BGS can satisfy our needs in graph visualization in terms of efficiency and effectiveness

CHAPTER 5

CONCLUSION AND FUTURE WORK

In the past chapters, we have presented an array of techniques aiming to answer the original research questions, namely: How can we obtain a good representative from the graph and infer useful information from the representative? How can we detect important objects or connections within the graph? How can we reveal the information of one vertex and its neighbors promptly? How can we effectively explore structures or topology of the graph? We evaluated commonly used graph sampling methods through a combined visual and statistical comparison of graphs sampled at various rates to uncover their preference in preserving graph properties in chapter 2, and developed three types of distributed graph sampling methods which are fit for large-scale graphs in chapter 3. In addition, a scalable graph visualization system-BGS was designed and developed to interactively visualize large-scale graphs through hierarchy view and graph view in chapter 4. In this chapter, we make a discussion and present concluding remarks on both graph sampling and visualization along with suggestions for further research.

5.1 Conclusion

In prior research, there was no a systematic empirical comparison of existing graph methods. Users were often confused regarding which sampling method should be used

in application. Improper choice on graph sampling methods would lead to inferring unreliable information from the representative. The thesis evaluated commonly used graph sampling methods to reveal their preference in preserving graph properties, and built a benchmark with both visual and statistical properties. The results indicates that the ranking of these evaluated graph sampling methods is dependent on graph type, desired statistical property, sampling efficiency, and visual requirements. A visual and statistical comparison made between non-distributed sampling methods and distributed graph sampling methods demonstrated that the conclusion is also applicable to distributed graph sampling methods. This findings may significantly aid us in large scale graph exploration by choosing an appropriate method. In addition, a sampling package including nine scalable graph sampling methods that work for large-scale graphs was developed on a distributed platform-Spark, which can be incorporated into GraphX for graph research. This library allows researchers to choose proper distributed sampling methods to obtain a desired representative from large-scale graphs based on their requirements.

For graph visualization, this work has set forth to address several of the remaining issues regarding large scale graph visualization. A scalable graph visualization tool was proposed which allows generating hierarchical structure by clustering and interactive navigation when visualizing large-scale graphs. The tuned Louvain clustering in BGS is an efficient hierarchy construction algorithm, which can be easily implemented in distributed computing systems. A series of hierarchy and graph exploration methods, such as hierarchy view, hierarchy navigation, hierarchy search, graph view, graph navigation, graph search, and other useful interactions presented in BGS allows researchers to utilize ap-

proaches with greater convenience to explore very large-scale graphs. The case study of BGS demonstrates its ability to visualize a large-scale graph with billion-scale edges. In addition, graph search on hierarchy view and graph view by vertices attribute(s) or edges attribute(s) in BGS helps users identify interesting vertices or edges effectively.

5.2 Future Work

We have presented a range of solutions to explore large scale graphs, yet there are a number of approaches that we have not tried yet. For example, we can try to analyze how distributed graph sampling methods directly perform on graph type and graph properties. Currently, some conclusions or observations are drawn from the inference of non-distributed graph sampling methods due to them both have very similar performance in preserving graph properties. Although we believe the inference from non-distributed graph sampling methods is reliable, the work on distributed sampling methods is much more straightforward and convincing, and helpful in large-scale graph exploration.

Another attempt on distributed graph sampling is analyzing how it performs on other graph properties, for example, connected component size, path length, etc. This analysis gives us a much more elaborate and complete evaluation of distributed graph sampling methods.

Further research can also focus on our large-scale graph visualization tool-BGS, we can enrich its functionality by designing more view modes in hierarchy view and graph view, and provide more visualization approaches to explore large-scale graphs.

Finally, another improvement can be made on BGSs efficiency and scalability through developing more efficient clustering methods and visualization modes. Generally, clustering on large-scale graphs require more resource (e.g. memory) than visualization. High efficiency clustering techniques definitely enhance BGSs efficiency and scalability.

In summary, although many graph techniques have been developed, graph sampling and graph visualization are still indispensable approaches to understanding large-scale graphs properties and structural information. Possibly, along with the development of relational technologies, some machine learning techniques (e.g. pattern recognition) may be incorporated into graph visualization and bring significant benefits to this research field.

REFERENCES

[1] J. Abello and J. Korn, "MGV: A system for visualizing massive multidigraphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 1, 2002, pp. 21–38.

[2] J. Abello and F. Van Ham, "Matrix zoom: A visual interface to semi-external graphs," *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*. IEEE, 2004, pp. 183–190.

[3] J. Abello, F. Van Ham, and N. Krishnan, "Ask-graphview: A large scale graph visualization system," *IEEE transactions on visualization and computer graphics*, vol. 12, no. 5, 2006, pp. 669–676.

[4] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 US election: divided they blog," *Proceedings of the 3rd international workshop on Link discovery*. ACM, 2005, pp. 36–43.

[5] V. S. Agneeswaran, *Big data analytics beyond hadoop: real-time applications with storm, spark, and more hadoop alternatives*, FT Press, 2014.

[6] N. Ahmed, J. Neville, and R. R. Kompella, "Network sampling via edge-based node selection with graph induction," 2011.

[7] N. K. Ahmed, F. Berchmans, J. Neville, and R. Kompella, "Time-based sampling of social network activity graphs," *Proceedings of the eighth workshop on mining and learning with graphs*. ACM, 2010, pp. 1–9.

[8] N. K. Ahmed, J. Neville, and R. Kompella, "Network sampling: From static to streaming graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 2, 2014, p. 7.

[9] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, "Analysis of topological characteristics of huge online social networking services," *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 835–844.

[10] B. Almende and B. Thieurmel, "visNetwork: Network Visualization usingvis. js Library," *R package version 0.2*, vol. 1, 2016.

[11] D. Archambault, T. Munzner, and D. Auber, "Topolayout: Multilevel graph layout by topological features," *IEEE transactions on visualization and computer graphics*, vol. 13, no. 2, 2007.

[12] D. Auber, "TulipA huge graph visualization framework," *Graph drawing software*, 2004, pp. 105–126.

[13] J. Barnes and P. Hut, "A hierarchical O (N log N) force-calculation algorithm," *nature*, vol. 324, no. 6096, 1986, pp. 446–449.

[14] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 11, 2004, pp. 3747–3752.

[15] M. Bastian, S. Heymann, M. Jacomy, et al., "Gephi: an open source software for exploring and manipulating networks.," *Icwsm*, vol. 8, 2009, pp. 361–362.

[16] V. Batagelj, "Efficient algorithms for citation network analysis," *arXiv preprint cs/0309023*, 2003.

[17] M. Y. Becker and I. Rojas, "A graph layout algorithm for drawing metabolic pathways," *Bioinformatics*, vol. 17, no. 5, 2001, pp. 461–467.

[18] S. Berg, "Snowball samplingI," *Encyclopedia of statistical sciences*, 1988.

[19] N. Bikakis, J. Liagouris, M. Krommyda, G. Papastefanatos, and T. Sellis, "graphVizdb: A scalable platform for interactive large graph visualization," *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 1342–1345.

[20] V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast Unfolding of Community Hierarchies in large network, 2008," *J. Stat. Mech. P*, vol. 1008.

[21] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, 2001, pp. 163–177.

[22] I. Bruß and A. Frick, "Fast interactive 3-D graph visualization," *International Symposium on Graph Drawing*. Springer, 1995, pp. 99–110.

[23] W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson, "Shiny: web application framework for R," *R package version 0.11*, vol. 1, 2015.

[24] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, 2004, p. 066111.

[25] D. E. Comer, *Computer networks and internets*, Prentice Hall Press, 2008.

[26] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal, Complex Systems*, vol. 1695, no. 5, 2006, pp. 1–9.

[27] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li, "Geometry-based edge clustering for graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, 2008, pp. 1277–1284.

[28] K. Dempsey, K. Duraisamy, H. Ali, and S. Bhowmick, "A parallel graph sampling algorithm for analyzing gene correlation networks," *Procedia Computer Science*, vol. 4, 2011, pp. 136–145.

[29] J. Díaz, J. Petit, and M. Serna, "A survey of graph layout problems," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, 2002, pp. 313–356.

[30] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in Hadoop MapReduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, 2012, pp. 2014–2015.

[31] P. Eades, M. E. Houle, and R. Webber, "Finding the best viewpoints for three-dimensional graph drawings," *International Symposium on Graph Drawing*. Springer, 1997, pp. 87–98.

[32] P. Ebbes, Z. Huang, A. Rangaswamy, H. P. Thadakamalla, and O. Unit, "Sampling large-scale social networks: Insights from simulated networks," *18th Annual Workshop on Information Technologies and Systems, Paris, France*, 2008.

[33] N. B. Ellison et al., "Social network sites: Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, 2007, pp. 210–230.

[34] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz-open source graph drawing tools," *International Symposium on Graph Drawing*. Springer, 2001, pp. 483–484.

[35] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete, "ZAME: Interactive large-scale graph visualization," *Visualization Symposium, 2008. PacificVIS'08. IEEE Pacific*. IEEE, 2008, pp. 215–222.

[36] P. Erdos and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, 1960, pp. 17–60.

[37] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and experience*, vol. 21, no. 11, 1991, pp. 1129–1164.

[38] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software Practice and Experience*, vol. 30, no. 11, 2000, pp. 1203–1233.

[39] J. Gehrke, P. Ginsparg, and J. Kleinberg, "Overview of the 2003 KDD Cup," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 2, 2003, pp. 149–151.

[40] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, 2002, pp. 7821–7826.

[41] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in facebook: A case study of unbiased sampling of osns," *Infocom, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[42] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. IEEE, 2004, vol. 1.

[43] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework.," *OSDI*, 2014, vol. 14, pp. 599–613.

[44] J. A. Guerra-Gomez, A. Wilson, J. Liu, D. Davies, P. Jarvis, and E. Bier, "Network Explorer: Design, Implementation, and Real World Deployment of a Large Network Visualization Tool," *Proceedings of the International Working Conference on Advanced Visual Interfaces*. ACM, 2016, pp. 108–111.

[45] A. Hagberg, P. Swart, and D. S Chult, *Exploring network structure, dynamics, and function using NetworkX*, Tech. Rep., Los Alamos National Laboratory (LANL), 2008.

[46] P. Hage and F. Harary, "Eccentricity and centrality in networks," *Social networks*, vol. 17, no. 1, 1995, pp. 57–63.

[47] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu, "iGraph: a framework for comparisons of disk-based graph indexing techniques," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, 2010, pp. 449–459.

[48] W.-S. Han, M.-D. Pham, J. Lee, R. Kasperovics, and J. X. Yu, "iGraph in action: performance analysis of disk-based graph indexing techniques," *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 1241–1242.

[49] E. Hartuv and R. Shamir, "A clustering algorithm based on graph connectivity," *Information processing letters*, vol. 76, no. 4-6, 2000, pp. 175–181.

[50] J. Heer and D. Boyd, "Vizster: Visualizing online social networks," *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*. IEEE, 2005, pp. 32–39.

[51] O. Hein, M. Schwind, and W. König, "Scale-free networks," *Wirtschaftsinformatik*, vol. 48, no. 4, 2006, pp. 267–275.

[52] I. Herman, G. Melançon, and M. S. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE Transactions on visualization and computer graphics*, vol. 6, no. 1, 2000, pp. 24–43.

[53] A. Hinge and D. Auber, "Distributed graph layout with Spark," *Information Visualisation (iV), 2015 19th International Conference on*. IEEE, 2015, pp. 271–276.

[54] D. Holten and J. J. Van Wijk, "Force-Directed Edge Bundling for Graph Visualization," *Computer graphics forum*. Wiley Online Library, 2009, vol. 28, pp. 983–990.

[55] P. Hu and W. C. Lau, "A survey and taxonomy of graph sampling," *arXiv preprint arXiv:1308.5865*, 2013.

[56] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani, "Metropolis algorithms for representative subgraph sampling," *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 283–292.

[57] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, no. 3, 1996, pp. 299–314.

[58] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software," *PloS one*, vol. 9, no. 6, 2014, p. e98679.

[59] A. James and R. Jones, "The social world of karate-do," *Leisure Studies*, vol. 1, no. 3, 1982, pp. 337–354.

[60] B. Jeon, I. Jeon, and U. Kang, "Tegviz: Distributed tera-scale graph generation and visualization," *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1620–1623.

[61] D. Krackhardt, "Graph theoretical dimensions of informal organizations," *Computational organization theory*, vol. 89, no. 112, 1994, pp. 123–140.

[62] B. Kulis, S. Basu, I. Dhillon, and R. Mooney, "Semi-supervised graph clustering: a kernel approach," *Machine learning*, vol. 74, no. 1, 2009, pp. 1–22.

[63] M. Kurant, A. Markopoulou, and P. Thiran, "Towards unbiased BFS sampling," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, 2011, pp. 1799–1809.

[64] A. N. Langville and C. D. Meyer, "A survey of eigenvector methods for web information retrieval," *SIAM review*, vol. 47, no. 1, 2005, pp. 135–161.

[65] L. Lee, "On the effectiveness of the skew divergence for statistical language analysis.," *AISTATS*, 2001.

[66] J. Leskovec and C. Faloutsos, "Sampling from large graphs," *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 631–636.

[67] J. Leskovec and E. Horvitz, "Planetary-scale views on a large instant-messaging network," *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 915–924.

[68] J. Leskovec and A. Krevl, "{SNAP Datasets}:{Stanford} Large Network Dataset Collection," 2015.

[69] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," *Advances in neural information processing systems*, 2012, pp. 539–547.

[70] E. A. Lopez-Rojas, "Social Network Analysis in the dataset US Air 97 with Pajek," *LiU*, , no. 1, 2011, pp. 1–11.

[71] R. Matei, A. Iamnitchi, and P. Foster, "Mapping the Gnutella network," *IEEE Internet Computing*, vol. 6, no. 1, 2002, pp. 50–57.

[72] C. Muelder and K.-L. Ma, "Rapid graph layout using space filling curves," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, 2008.

[73] C. Mueller, D. P. Gregor, and A. Lumsdaine, "Distributed Force-Directed Graph Layout and Visualization.," *EGPGV*, vol. 6, 2006, pp. 83–90.

[74] T. Munzner, "H3: Laying out large directed graphs in 3D hyperbolic space," *Information Visualization, 1997. Proceedings., IEEE Symposium on*. IEEE, 1997, pp. 2–10.

[75] T. Munzner, "Exploring large graphs in 3D hyperbolic space," *IEEE Computer Graphics and Applications*, vol. 18, no. 4, 1998, pp. 18–23.

[76] M. E. Newman, "The structure and function of complex networks," *SIAM review*, vol. 45, no. 2, 2003, pp. 167–256.

[77] M. E. Newman, "Detecting community structure in networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 38, no. 2, 2004, pp. 321–330.

[78] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social networks*, vol. 32, no. 3, 2010, pp. 245–251.

[79] T. P. Peixoto, "The graph-tool python library," *figshare*, 2014.

[80] S. U. Pillai, T. Suel, and S. Cha, "The Perron-Frobenius theorem: some of its applications," *IEEE Signal Processing Magazine*, vol. 22, no. 2, 2005, pp. 62–75.

[81] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *ISCIS*, 2005, vol. 3733, pp. 284–293.

[82] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, 2007, pp. e177–e183.

[83] D. Rafiei, "Effectively visualizing large networks through sampling," *Visualization, 2005. VIS 05. IEEE*. IEEE, 2005, pp. 375–382.

[84] B. Ribeiro and D. Towsley, "Estimating and sampling graphs with multidimensional random walks," *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 390–403.

[85] G. Sander, "Graph layout through the VCG tool," *International Symposium on Graph Drawing*. Springer, 1994, pp. 194–205.

[86] S. E. Schaeffer, "Graph clustering," *Computer science review*, vol. 1, no. 1, 2007, pp. 27–64.

[87] J. Scott, *Social network analysis*, Sage, 2017.

[88] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Transactions on graphics (TOG)*, vol. 11, no. 1, 1992, pp. 92–99.

[89] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE, 1996, pp. 336–343.

[90] A. Spark, "Apache Spark: Lightning-fast cluster computing,", 2016.

[91] M.-A. D. Storey and H. A. Müller, "Graph layout adjustment strategies," *International Symposium on Graph Drawing*. Springer, 1995, pp. 487–499.

[92] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "Sampling techniques for large, dynamic graphs," *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*. IEEE, 2006, pp. 1–6.

[93] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "On unbiased sampling for unstructured peer-to-peer networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 2, 2009, pp. 377–390.

[94] R. Team, "RStudio: integrated development for R," *RStudio, Inc., Boston, MA URL http://www. rstudio. com*, 2015.

[95] C. Tominski, J. Abello, and H. Schumann, "CGVAn interactive graph visualization system," *Computers & Graphics*, vol. 33, no. 6, 2009, pp. 660–678.

[96] N. Tsapanos, A. Tefas, N. Nikolaidis, and I. Pitas, "Large graph clustering using DCT-based graph clustering," *Computational Intelligence in Big Data (CIBD), 2014 IEEE Symposium on*. IEEE, 2014, pp. 1–4.

[97] S. M. Van Dongen, *Graph clustering by flow simulation*, doctoral dissertation, 2001.

[98] J. Q. Walker, "A node-positioning algorithm for general trees," *Software: Practice and Experience*, vol. 20, no. 7, 1990, pp. 685–705.

[99] T. Wang, Y. Chen, Z. Zhang, P. Sun, B. Deng, and X. Li, "Unbiased sampling in directed social graph," *ACM SIGCOMM Computer Communication Review*. ACM, 2010, vol. 40, pp. 401–402.

[100] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li, "Understanding graph sampling algorithms for social network analysis," *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*. IEEE, 2011, pp. 123–128.

[101] D. J. Watts and S. H. Strogatz, "Collective dynamics of'small-world'networks," *nature*, vol. 393, no. 6684, 1998, p. 440.

[102] H. Wickham and R. Francois, "dplyr: A grammar of data manipulation," *R package version 0.4*, vol. 1, 2015, p. 20.

[103] A. Y. Wu, M. Garland, and J. Han, "Mining scale-free networks using geodesic clustering," *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 719–724.

[104] Y. Wu, N. Cao, D. Archambault, Q. Shen, H. Qu, and W. Cui, "Evaluation of graph sampling: A visualization perspective," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, 2017, pp. 401–410.

[105] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 1, 2007, pp. 40–51.

[106] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, 2015, pp. 181–213.

[107] S. Yoon, S. Lee, S.-H. Yook, and Y. Kim, "Statistical properties of sampled networks by random walks," *Physical Review E*, vol. 75, no. 4, 2007, p. 046114.

[108] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, 2010, p. 95.

[109] F. Zhang, S. Zhang, P. Chung Wong, H. Medal, L. Bian, I. Swan, J. Edward, and T. Jankun-Kelly, "A Visual Evaluation Study of Graph Sampling Techniques," *Electronic Imaging*, vol. 2017, no. 1, 2017, pp. 110–117.

[110] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, 2009, pp. 718–729.