BGS: A Large-Scale Graph Visualization System

Fangyan Zhang, Song Zhang, Christopher Lightsey, Sarah Harun, and Pak Chung Wong

Abstract—we present BGS (Big Graph Surfer), a scalable graph visualization system that creates hierarchical structure from original graphs and provide 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 bigger than 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. To evaluate the effectiveness of BGS, we apply BGS to several large-scale graph datasets, and discuss its scalability, usability, and flexibility.

Index Terms— graph visualization, graph clustering, graph hierarchy, graph hierarchy

1 Introduction

Graphs, as a prevalent method to represent real world datasets, are widely used in diverse fields, like social network, Internet network, citation network, etc. 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 [1]. 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 [2].

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 [3] [4] 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 [5] [6]. 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 undermines 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 alleviate the above issues, a great number of visualization techniques have been proposed over the last few decades. We present a graph visualization system called BGS which is designed to visualize large-scale graphs by combining several ideas from prevailing graph visualization systems, and overcomes their drawbacks in dealing with the above issues. For BGS, the fundamental task is to visualize very large-scale graphs that are too large to fit into main memory, and interact with such graphs efficiently.

According to Shneiderman's visualization principle of "Overview fast, zoom and filter, then details-on-demand" [7], 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 [2][8][9]. 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 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 [10].

In term of architecture, BGS is developed on several platforms: Spark [11], R [12], RStudio [13], and Shiny [14]. Spark is a distributed computing framework deployed on supercomputers, which act as a back-end platform working on graph hierarchy construction, graph filtering, and aggregation etc. Shiny works as front-end to visualize graphs in 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 system 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 paper is organized as follows. In Section 2, we 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 (Section 3), followed by case study and evaluation on several graph datasets (Section 4). Afterwards, we discuss scalability and usability of the tool (Section 5). Finally, we conclude the work and introduce future work in Section 6.

2 Related Work

A variety of graph visualization systems have been proposed, such as ASK-GraphView [9], CGV [15], TeGViz [16], GraphVizdb [17], Network Explorer [18], Vizster [19], ZAME [20], Matrix Zoom [21], 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.

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.

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 1 shows the architecture of BGS. The Spark works on HPC clusters as server (back-end) undertaking heavy computation tasks like clustering, filtering, aggregation etc. Shiny and visNetwork [22] act as the client (front-end) interpreting graph data and displaying graph in WebUI [14]. R and RStudio act as intermediate module that works for the communication between client and server. R is connected to Spark via Sparklyr [33]. Sparklyr is a R package which provides a complete dplyr [23] backend and enable R to manipulate Spark. Shiny and visNetwork both are R packages. The former is a web application 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.

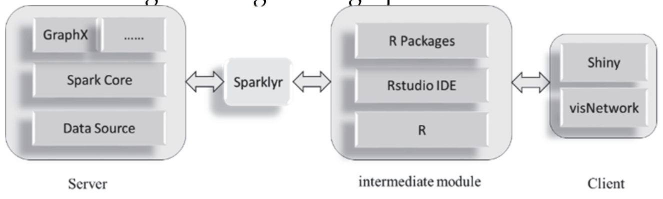


Figure 1: Architecture of BGS

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 [24]. 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 [25] and forceAtlas2Based algorithm [26] to optimize graph layout. For example, we can adjust arrangement of vertices in visualization by altering gravity value and spring constant value to increase its readability.

3.3 Hierarchy

For dense graph or large-scale graphs, some techniques are useful to maintain readability of graph visualization, such as dimensionality reduction [27], 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 [10]: divisive algorithms [28] [29] (detect inter-community links and remove them from the network), agglomerative algorithms [30] [31] (merge similar nodes recursively), and optimization methods (maximize an objective function) [10]. 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 Ref. [32]. Its complexity is *O (nlog₂n)* for a network with n vertices. ASK-GraphView uses one type of agglomerative algorithm- tuned Markov Cluster Algorithm (MCL) [31] 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 [10] 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:

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

- $A_{i,j}$: 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:

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

The classic Louvain clustering algorithm was proposed in Ref. [10], we improved the above Louvain clustering algorithm via controlling the number of nodes in clustering.

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 $G_i (V_i, E_i)$, then $G_0 (V_0, E_0)$ is *G (V, E)*, and $G_i (V_i, E_i)$ is abstracted from $G_{i-1} (V_{i-1}, E_{i-1})$.

For hierarchy tree, we define the following concepts:

- *T*: the whole hierarchy tree
- T_i : the subtrees at ith level.
- *Leaves (T)*: set of leaves of *T*. *Leaves (T) = V0 = V*.
- *Children* (T_i): the children of subtree T_i. *Children (Ti)* = 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.

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 following sections will discuss each functionality in detail.

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. Clusters' vertical information is absorbed by their children with expansion in graph view. Hierarchy view offers us high level abstractions of the original graph. More importantly, 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.

3.5.2 Expansion Mode

In order to satisfy users' different demands in graph visualization, we design two expansion modes (Minimum 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 observe 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.

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.

3.5.4 Hierarchy Exploration

Hierarchical structure represents graph's 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 information for users. BGS deals with such problem by allowing users to set serval top levels for observation at the beginning. If one hierarchy has depth *h*, and the initial hierarchy has *s* layers, then the initial hierarchy is $\{T_i, h-s+1 < i <= h\}$ which provides informative context for users to explore the graph hierarchy. Also, the several top levels in the hierarchy will consistently exist with expanding clusters. For example, Figure 2 shows selecting top two layers in hierarchy view.

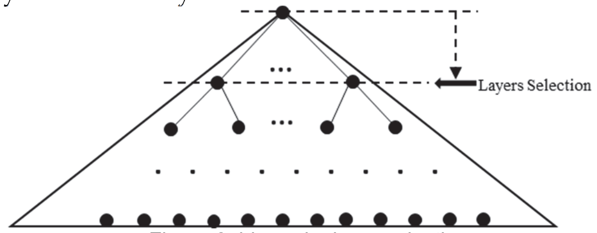


Figure 2: hierarchy layer selection

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 3 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).

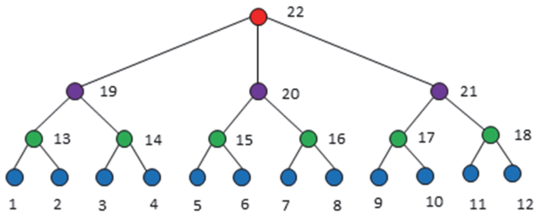


Figure 3: original graph hierarchy

As we mentioned before, there are two hierarchy expansion modes: Minimum hierarchy expansion and Add-Up hierarchy expansion. In Minimum mode, which is demonstrated in figure 4, 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.

In Add-Up mode, for example in figure 5, when expanding one cluster (node 13), just as Minimum mode, it will create out-going links from the cluster to connect its children (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 expanded cluster. Add-Up mode allows multiple clusters/nodes, their siblings, and their predecessors in hierarchy to be visualized.

c) Hierarchy Search

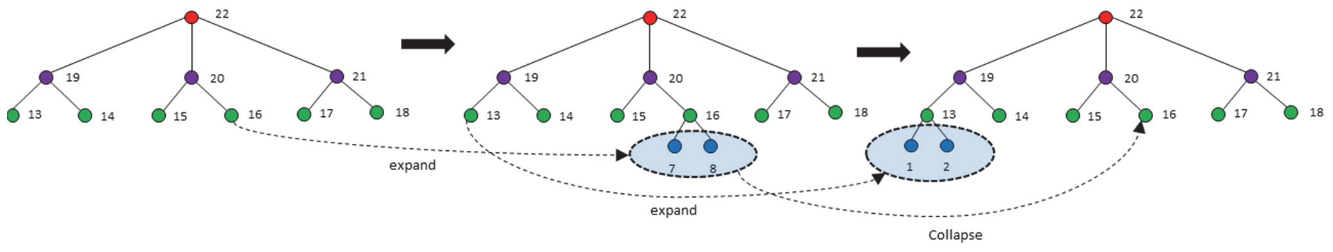


Figure 4: hierarchy expansion in Minimum mode

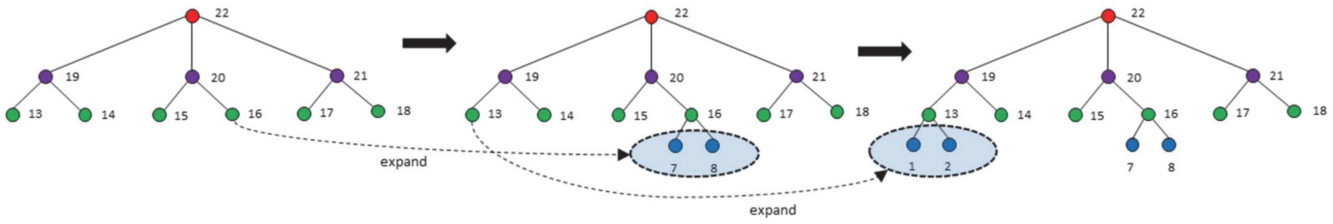
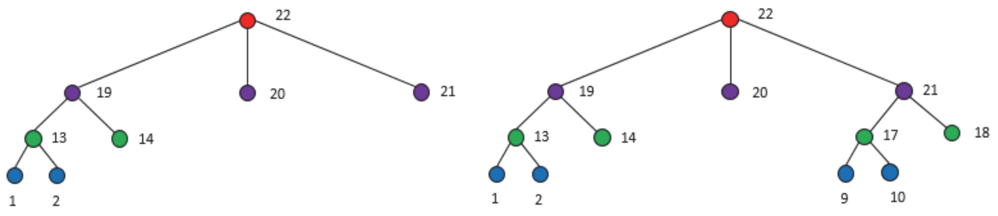


Figure 5: hierarchy expansion in Add-Up mode



(a) (b)

Figure 6: hierarchy search in Minimum mode (a) and Add-up mode (b)

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 6, (a) search node 1 or 2 at Minimum mode, (b) search node 1 or 2, and 9 or 10 at Add-Up mode.

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

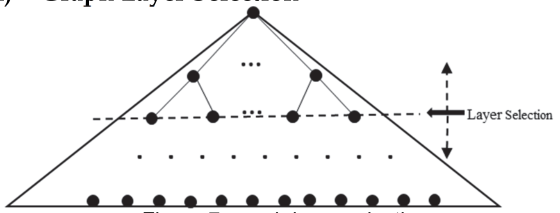


Figure 7: graph layer selection

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 7, 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.

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 cluster's children and the target cluster's neighbors. To check the crossover edges, all pairs of neighbors and children are reduced to the same layer. Like hierarchy expansion, graph expansion has Minimum mode and Add-Up mode. In Minimum mode, demonstrated in Figure 8, 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 cluster's (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 cluster's predecessors. Minimum mode only allows one cluster, its siblings, and its predecessors and their siblings to be visualized.

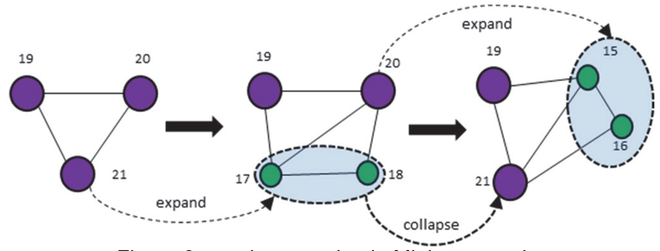


Figure 8: graph expansion in Minimum mode

In Add-Up mode, demonstrated in Figure 9, 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.

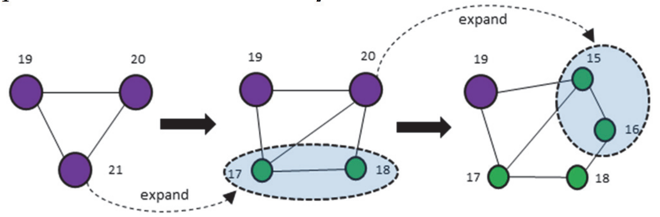


Figure 9: graph expansion in Add-Up mode

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 10, initially, only vertices are visualized in graph view, all edges within the vertices are hidden (Figure 10 (a)). If users want to observe the links adjacent to one vertex, just clicking on the vertex reveals the edges adjacent to that vertex (Figure 10). This is demonstrated in Figure 10 with no edges clicked (Figure 10 (a)), vertex 21 selected (Figure 10(b)) and vertex 20 selected (Figure 10(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.

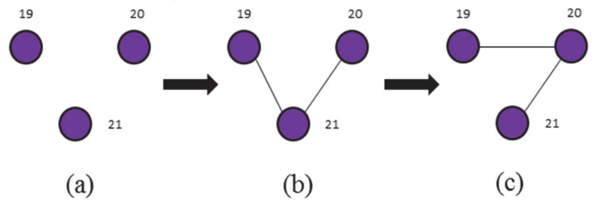


Figure 10: Graph View in Edge-Free mode

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 11, (a) search node 16, (b) search node 16 and 18.

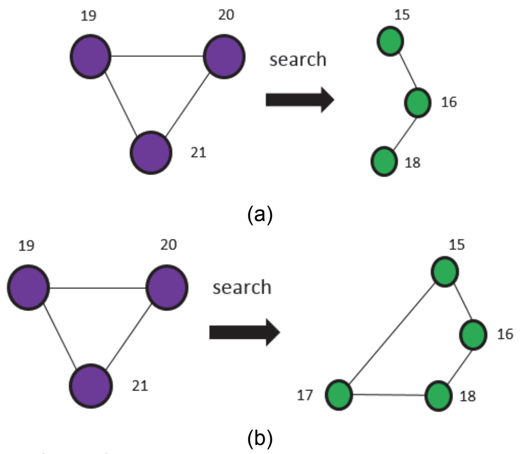


Figure 11: Graph Search on one node (a): node 16; and two nodes (b): node 16 and 18.

The above sections talked about two expansion modes (Minimum mode and Add-Up mode) on hierarchy exploration and graph exploration, and two graph view modes

(Regular mode and Edge-Free mode). These 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. 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.

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 cluster's 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 cluster's 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 BGS's efficiency but also maintain its visualization scalability.

3.5.7 Decorations and Interactions

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 Shneiderman's visualization principle, we can realize the importance of Interaction in graph exploration. According to BGS's 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 Case Study

To illustrate BGS's 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 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 BGS's performance on functionalities, scalability, and interactions. BGS can be thoroughly evaluated from these three aspects.

Table 1: Graph datasets for visualization

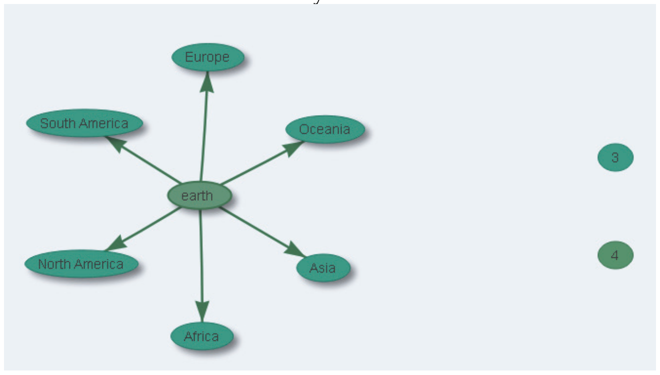
Graph	Vertices	Edges	Attributed	File Size
Facebook	4,039	88,234	No	1MB
Flight	3,125	58,568	Yes	1MB
Friendster	65,608,366	1,806,067,135	No	30GB

4.1 BGS Functionalities

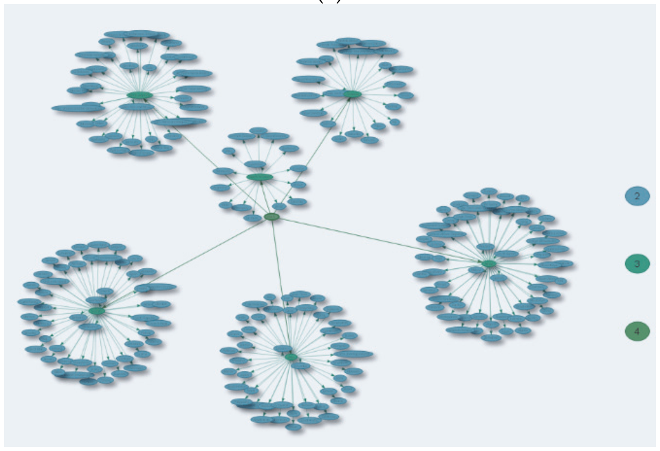
a) Hierarchy Layers Selection

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 12 is two views of hierarchy layers selection on Flight data. Figure 12(a) 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 cluster of the United Sates. Figure 12(b) shows the top three layers of the hierarchy, which reaches country layer. In Figure 12 (b) there are six clusters representing six continents. From this hierarchy view, users can expand one country to look for cities or airports.

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.



(a)



(b)

Figure 12: Hierarchy layer selection (a) top 2 layers; (b) top 3 layers of Flight Graph.

b) Hierarchy View

Figure 13 is the hierarchy view on Friendster graph in Minimum mode. Figure 14 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 13 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 14 is showing hierarchy of Nadi international airport and Auckland international airport. From the hierarchy we know both airports belong to different countries but both are located in Oceania.

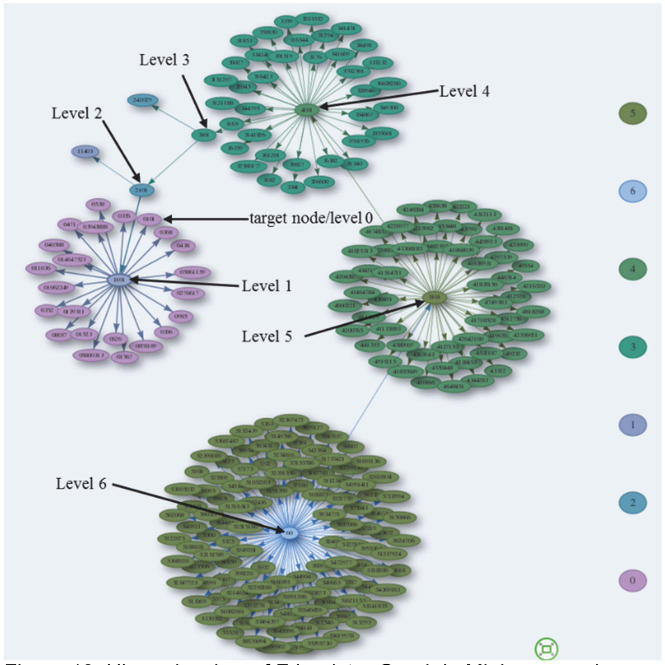


Figure 13: Hierarchy view of Friendster Graph in Minimum mode

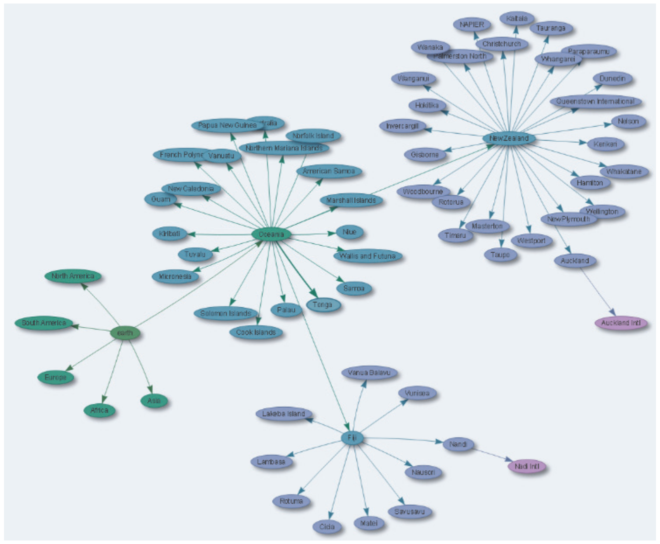


Figure 14: Hierarchy view of Flight Graph in Add-Up mode. Leaf Nodes: Nadi International Airport (Nandi, Fiji, Oceania) and Auckland International Airport (Auckland, New Zealand, Oceania)

c) Graph View

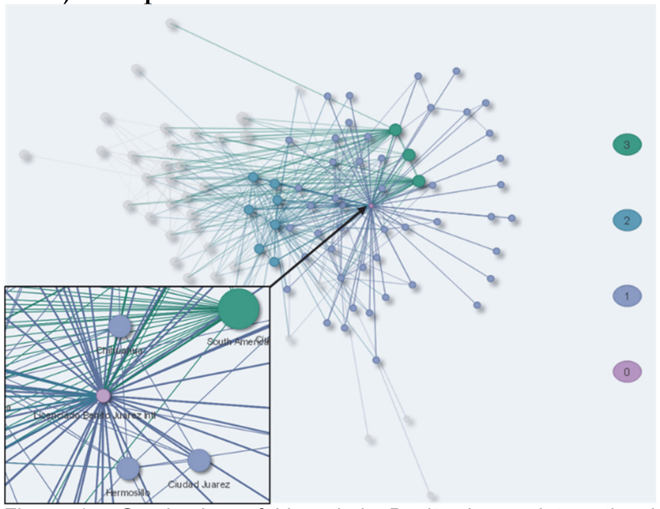


Figure 15: Graph view of Licenciado Benito Juarez International Airport (Mexico City) from the Flight graph

Figure 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 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.

d) Graph Search

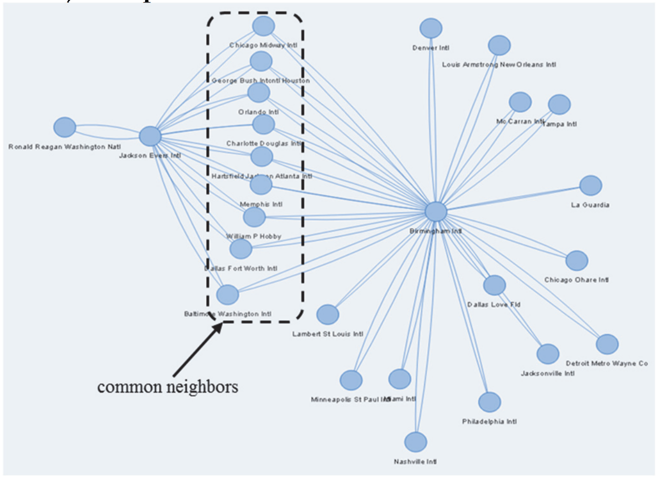


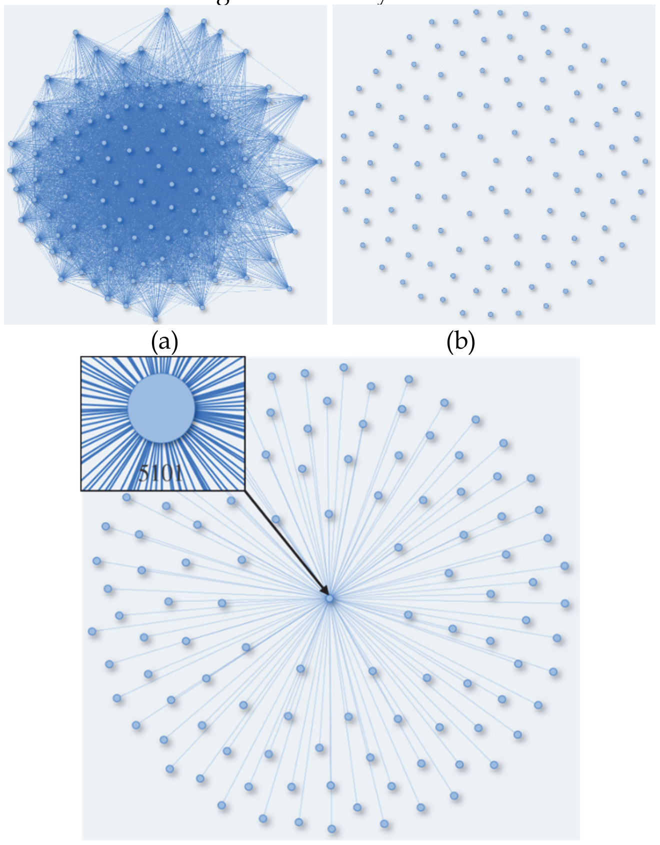
Figure 16: Graph search view of Jackson Evers international airport and Birmingham international airport from the Flight graph

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 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.

e) Graph View Mode

Figure 17 shows the graph view of Friendster data at top layer with Regular mode (a) and Edge-Free mode (b and c). In Regular mode, Figure 17 (a) has 113 vertices and 5603 edges. From such dense graph, we can hardly see any details due to the "hairballs". Figure 17 (b) 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 17 (c). BGS allows us to observe other edges adjacent to one node by clicking the node. Compared Figure 17 (a) and (c), Edge-Free mode is obviously very effective in reducing overlaps in graph view and increasing its readability.



(c)

Figure 17: Comparison of graph view mode on Friendster; (a) Regular mode; (b) Initial graph view in Edge-Free mode; (c) Edge-Free mode showing edges adjacent to 5101

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 18 shows graph layer selection at layer 4 to layer 1.

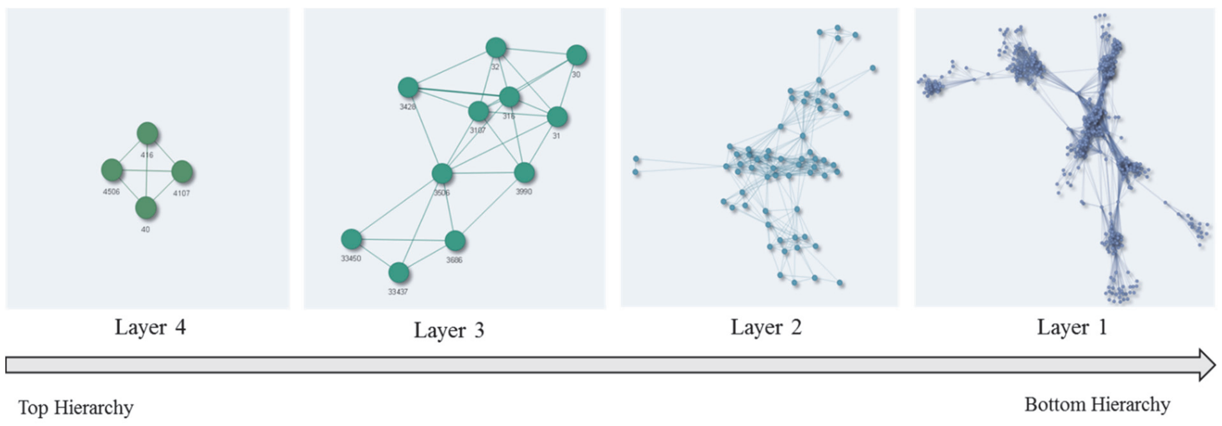


Figure 18: Graph layer selection on Facebook graph

Table 2: clustering time, loading time, and visualization time for graph datasets

Graph	Clustering Time	Graph Load Time	Visualization Delay Time	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

Table 3: combination of visualization mode and expansion mode

View mode / Visualization mode	Minimum mode	Add-Up mode
Local-Memory mode	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-Memory mode	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;

4.2 BGS Scalability

From the case studies, our visualization system can easily visualize graphs with billion scale edges. Table 2 shows the clustering time, loading time, and visualization time for the three graphs using BGS. The good efficiency of BGS benefits from 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 graph's structure and density. This feature increases BGS's flexibility in application.

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.

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.

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 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 "hairballs" 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.

6 Conclusion and Future Work

In this paper, 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 clustering 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 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.

Acknowledgment

This work has been supported by the United States Army Corps of Engineers under Contracts W912HZ-17-C-0016 and W912HZ-17-C-0015, by the U.S. Department of Defense, and by the Pacific Northwest National Laboratory which is managed for the U.S. Department of Energy by Battelle under Contract DE-AC05-76RL01830.

References

[1] J. Leskovec and E. Horvitz, "Planetary-scale views on a large instant-messaging network," *WWW conf.*, pp. 915–924, 2008.

[2] J. Abello and J. Korn, "MGV: A system for visualizing massive multidigraphs," *IEEE Trans. Vis. Comput. Graph.*, vol. 8, no. 1, pp. 21–38, 2002.

[3] C. Muelder and M. Kwan-Liu, "Rapid graph layout using space filling curves," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1301–1308, 2008.

[4] M. A. D. Storey and H. A. Muller, "Graph layout adjustment strategies," *Lect. Notes Comput. Sci.*, vol. 1027, pp. 487–499, 1996.

[5] H. Antoine and A. David, "Distributed graph layout with spark," in *Proceedings of the International Conference on Information Visualisation*, 2015, vol. 2015–Septe, pp. 271–276.

[6] C. Mueller, D. Gregor, and A. Lumsdaine, "Distributed Force-directed Graph Layout and Visualization," *Proc. 6th Eurographics Conf. Parallel Graph. Vis.*, pp. 83–90, 2006.

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

[8] D. Archambault, T. Munzner, and D. Auber, "TopoLayout: Multilevel graph layout by topological features," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 2, pp. 305–316, 2007.

[9] J. Abello, F. Van Ham, and N. Krishnan, "ASK-Graph View: A large scale graph visualization system," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, p. 669, 2006.

[10] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of community hierarchies in large networks," *Networks*, pp. 1–6, 2008.

[11] Apache Spark, "Apache Spark™ - Lightning-Fast Cluster Computing," *Spark.Apache.Org*, 2015. .

[12] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *J. Comput. Graph. Stat.*, vol. 5, no. 3, pp. 299–314, 1996.

[13] J. S. Racine, "RStudio: A Platform-Independent IDE for R and Sweave," *J. Appl. Econom.*, vol. 27, no. 1, pp. 167–172, 2012.

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

[15] C. Tominski, J. Abello, and H. Schumann, "CGV - An interactive graph visualization system," *Comput. Graph.*, vol. 33, no. 6, pp. 660–678, 2009.

[16] B. Jeon, I. Jeon, and U. Kang, "TeGViz: Distributed Tera-Scale Graph Generation and Visualization," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, pp. 1620–1623.

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

[18] J. A. Guerra-gomez, A. Wilson, J. Liu, and D. Davies, "Network Explorer□: Design , Implementation , and Real World Deployment of a Large Network Visualization Tool," *Proc. Int. Work. Conf. Adv. Vis. Interfaces - AVI '16*, pp. 108–111, 2016.

[19] J. Heer, "Vizster□: Visualizing Online Social Networks," 2003.

[20] N. Elmqvis, T. N. Do, H. Goodell, N. Henry, and J. D. Fekete, "ZAME: Interactive large-scale graph visualization," *IEEE Pacific Vis. Symp. 2008, PacificVis - Proc.*, pp. 215–222, 2008.

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

[22] B. V. Almende and B. Thieurmel, "visNetwork: Network Visualization using 'vis.js' Library," *CRAN*. 2016.

[23] H. Wickham and R. Francois, "The dplyr package," *R Core Team*, 2016.

[24] G. Csárdi and T. Nepusz, "The igraph software package for complex network research," *InterJournal Complex Syst.*, vol. 1695, pp. 1–9, 2006.

[25] J. Barnes and P. Hut, "A hierarchical O(N log N) force-

calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[26] 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.

[27] 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 Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 1, pp. 40–51, 2007.

[28] M. E. J. Newman, "Detecting community structure in networks," *Eur. Phys. J. B - Condens. Matter Complex Syst.*, vol. 38, pp. 321–330, 2004.

[29] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks.," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 99, pp. 7821–7826, 2002.

[30] P. Pons and M. Latapy, "Computing communities in large networks using random walks," *J. Graph Algorithms Appl.*, vol. 10, no. 2, pp. 191–218, 2006.

[31] S. van Dongen, "Graph clustering by flow simulation," *Graph Stimul. by flow Clust.*, vol. PhD thesis, p. University of Utrecht, 2000.

[32] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Cond-Mat/0408187*, vol. 70, p. 66111, 2004.

[33] https://spark.rstudio.com/