TeGViz: Distributed Tera-Scale Graph Generation and Visualization

ByungSoo Jeon
Dept. of Computer Science
Korea Advanced Institute of
Science and Technology (KAIST),
Daejeon, Korea 34141
Email: bsjeon@kaist.ac.kr
Phone : +82-10-5001-4553

Inah Jeon
Future IT R&D Lab.
LG Electronics
Seoul, Korea 06772
Email: june5324@gmail.com
Phone : +82-10-3089-5324

U Kang
Dept. of Computer Science and Engineering
Seoul National University
Seoul, Korea 08826
Email: ukang@snu.ac.kr
Phone : +82-2-880-7254

***Abstract*—How can we generate and visualize tera-scale graphs efficiently? Although there are several graph generators available, they lack scalability, and require additional time to upload graphs for further analysis in distributed systems. Visualizing a massive graph from a graph generator is another challenge since the amount of information in the graph exceeds the resolution of a typical screen.**

In this paper, we proposes TeGViz, a distributed Tera-scale graph generation and visualization tool. TeGViz consists of two modules: graph generation and visualization. The graph generation module TeG generates a wide range of graphs including Erdős-Rényi random graph and realistic graphs including R-MAT and Kronecker directly on distributed systems. Thanks to our carefully designed parallel algorithms, TeG outperforms competitors in terms of scalability. The visualization module NetRay summarizes graphs using spy plot, distribution plot, and correlation plot to find regularities and anomalies effectively and makes it easy to understand generated graphs.

***Keywords*—Graph Generator; Visualization; Large-scale Graph; Random Graph; R-MAT; Kronecker; MapReduce**

I. Introduction

Graphs are used extensively to model many real world processes, like social network, the Web, citation network, etc. Although there are many public graphs available for study, an important tool for graph analysis is a graph generator which generates graphs with arbitrary sizes and various characteristics. Graph generator has a wide range of applications:

- **Simulation**: in designing a new graph analysis algorithm, we want to test the algorithm's behavior on different graph sizes and characteristics.
- **Sampling/Extrapolation**: if we cannot run an algorithm on a given graph due to its complexity, we want to generate a smaller graph with similar characteristics. On the other hand, we might want to generate a much larger graph to a given graph with similar characteristics.
- **Graph understanding**: by understanding how the model parameters affect the network properties, we can better understand the graph.

Although there are several graph generators [1] [2] [3] available, they have limitations. First, while the needs for generating very large graphs are increasing, most of the existing generators run on a single machine, and thus cannot generate very large graphs exceeding the capacity of a single machine. Second, recently significant portion of the data are stored in distributed systems (e.g. MapReduce and Hadoop) for which many data analysis tools are designed. Since the existing generators store the data in a system separated from the distributed systems, it takes another huge amount of time to upload the generated graph to the distributed systems. Thus, the need for an efficient and effective graph generation system that conquers the above limitations are ever increasing.

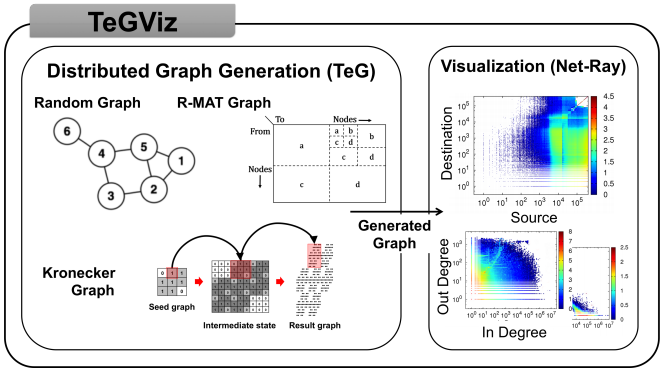


Fig. 1: Overview of TeGViz. TeGViz consists of 2 modules which work in a fully distributed way on Hadoop Framework. The graph generation module efficiently generates various random and realistic graphs. The visualization module visualizes the generated graph to show the connectivity, community, and outliers in the graph.

Another challenge is to visualize a massive graph from a graph generator; visualization of the graph is important since 1) it directly gives important information on the graph including connectivity patterns, community structures, outliers, etc., and 2) it gives a crucial tool for data miners to communicate with non-experts: e.g., government officials, domain experts, etc.

In this paper, we proposes TeGViz, a distributed Tera-scale graph generation and visualization tool. It runs on MapReduce [4], or its open-source counterpart Hadoop [5], framework for scalability. TeGViz supports a wide variety of graphs including Erdős-Rényi random graph, and realistic graph models like R-MAT [6] and Kronecker [7] which generate graphs reflecting the characteristics of real world graphs such as power law degree distribution and small diameter.

*Corresponding author: U Kang (email: ukang@snu.ac.kr)

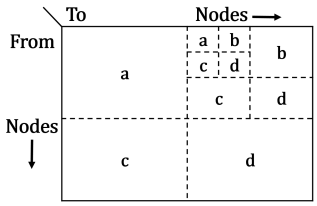


Fig. 2: R-MAT model. Each edge is recursively placed in one of the 4 partitioned regions with probabilities of a, b, c, and d, respectively.

After the user specifies the input parameters, TeGViz carefully distributes the required graph generation tasks into distributed machines which work independently to generate the graph. The resulting graph is fed into our distributed graph visualization module which shows connectivity and community structure patterns.

Our main contributions are the followings.

- **Efficient algorithm and implementation**: TeGViz is carefully designed to work on distributed systems. TeGViz works in an embarassingly parallel way; each machine has a balanced amount of tasks.
- **Scalability**: TeGViz generates at least 512× larger graph than existing generators. Also, TeGViz achieves near linear scalability on the number of machines. Generating and visualizing a billion-scale graph takes less than a minute.
- **Generation inside distributed systems**: Since the generated graphs are on distributed systems, there is no need to upload the graphs to distributed systems to apply many data analysis tools (e.g., [8]) designed for distributed systems.

The TeGViz code is available at http://datalab.snu.ac.kr/tegviz. The rest of the paper is organized as follows: system overview in Section 2, demonstration plan in Section 3, related works in Section 4, and conclusion in Section 5.

II. System Overview

A. Overview

TeGViz includes TeG, a distributed graph generator we propose in this paper. TeG generates three types of graphs from different models: Erdős-Rényi, R-MAT, and Kronecker. Erdős-Rényi is a random graph model. R-MAT and Kronecker are widely used models to generate realistic graphs that capture the properties of real world graphs. TeG also uses Net-Ray [9] to visualize tera-scale graphs using spy plot, distribution plot, and correlation plot. We implement our software on MapReduce [4] framework for scalability.

B. Graph Generation

1) R-MAT Graph: R-MAT [6] is one of the most widely used graph generative models. The main idea is to place each edge in one of the 4 partitioned regions with different probabilities recursively, as shown in Figure 2.

A challenge in designing a distributed algorithm for R-MAT is to evenly distribute the work into each machine. Our main idea is to assign each block to each machine in a greedy way until balance is not destroyed. Our proposed distributed

Algorithm 1: Distributed R-MAT in TeG: work assignment

Input: n: number of nodes,
m: number of edges,
N: number of machines,
ϵ : unbalance parameter, and
probabilities: a, b, c, and d
Output: $rectListForMachine(M_i)$ for $i = 1..N$ (group of rectangles of adjacent matrix for each machine)

```
1: initialize rectListForMachine(M_i) to be empty
2: set rectList to the 4 rectangles from the original adjacent matrix
3: repeat
4:    /* fill rectListForMachine(M_i) for each machine */
5:    pop first item from rectList
6:    if there exists rectListForMachine(M_i) whose number of edges after adding item to it is smaller than m/N + ε then
7:       add item to rectListForMachine(M_i)
8:    else
9:       split item into 4 pieces (for a,b,c,d)
10:      add 4 pieces to rectList
11:   end if
12: until rectList becomes empty
```

Algorithm 2: Distributed R-MAT in TeG: graph generation in machine i

Input: $rectListForMachine(M_i)$: rectangles for machine i to generate, and
probabilities: a, b, c, and d
Output: edges generated from machine i

```
1: repeat
2:    pop first item from rectListForMahcine(M_i)
3:    if the number of edges in item is 1 then
4:       generate a random edge in item
5:    else
6:       split item into 4 pieces (for a,b,c,d)
7:       add 4 pieces to rectListForMachine(M_i)
8:    end if
9: until rectListForMachine(M_i) becomes empty
```

R-MAT in TeG consists of two steps: 1) work assignment step, and 2) distributed generation step. In the work assignment step (Algorithm 1) which is done in a single machine, the goal is to assign k_i edges, where $\frac{m}{N} - (N-1)\epsilon \leq k_i \leq \frac{m}{N} + \epsilon$, to each machine i where m is the total number of edges, N is the number of machines, and ϵ is a parameter to bound the maximum error in balance. We start with the initial 4 blocks from the original adjacency matrix, and assign each block to an available machine whose total number of edges after adding the edges in the block is less than $\frac{m}{N}+\epsilon$. If none of the machines are available since the block is too large, we recursively partition the block, until each of the divided block is assigned to at least one machine. Note that in the end the difference of the assigned edges in any two machines is at most $N\epsilon$ since in the extreme case the first $N-1$ machines will have $\frac{m}{N} + \epsilon$ edges while the last machine would have $\frac{m}{N} - (N-1)\epsilon$ edge. In the distributed generation step (Algorithm 2), each machine generates edges based on the assignment determined in the previous step, in a fully distributed way without any communication with other machines.

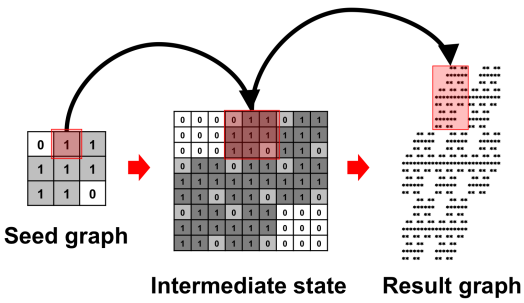


Fig. 3: The construction of Kronecker graph in TeG. The entire graph is divided into several subgraphs which are assigned to each machine; each machine recursively generates the assigned subgraph.

2) Random Graph: In random graph, edges are generated randomly. Users give the number of nodes and edges as input, and the random graph generator outputs edges between randomly selected nodes. TeG's distributed algorithm to generate random graph is very similar to the algorithm for distributed R-MAT in Section II-B1 since random graph is a special case of R-MAT graph where $a = b = c = d = 0.25$. The only difference is that the number of nodes in random graphs does not need to be a power of 2.

3) Kronecker Graph: Kronecker graph model [7] is another real-world graph generative model. Kronecker graph model recursively performs the *Kronecker product* to generate graph. The *Kronecker product* of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is denoted by $\mathbf{A} \otimes \mathbf{B}$. The result is a matrix of size $(IK) \times (JL)$ and defined by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{a}_{11}\mathbf{B} & \mathbf{a}_{12}\mathbf{B} & \cdots & \mathbf{a}_{1J}\mathbf{B} \\ \mathbf{a}_{21}\mathbf{B} & \mathbf{a}_{22}\mathbf{B} & \cdots & \mathbf{a}_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{I1}\mathbf{B} & \mathbf{a}_{I2}\mathbf{B} & \cdots & \mathbf{a}_{IJ}\mathbf{B} \end{bmatrix}$$

To generate Kronecker graph, TeG receives the number of iterations, the number of machines, and a small sized seed graph as input parameters. In each iteration the current graph is expanded to a larger graph by Kronecker multiplication with the seed graph. As in the R-MAT and random graphs, an important point in generating Kronecker graph is to divide the work evenly into each machine. TeG exploits the characteristic of Kronecker product that each nonzero at an iteration is expanded to the same number of nonzeros in later iterations. Thus, TeG first expands the seed graph until the number of nonzeros exceeds the number of machines; then each nonzero is assigned to a machine which expands the assigned nonzeros into many edges. Note that this method guarantees that each machine is assigned at most two nonzeros at the time of assignment.

4) Performance of Graph Generator: We compare the running time of TeG and existing graph generators, GTgraph [1], GraphStream [2] and NetworkX [3]. To run TeG, we use a Hadoop cluster with 40 machines where each machine is equipped with an Intel Xeon E3-1230v3 CPU (quad-core at 3.30GHz), and 32GB RAM. To run single machine graph generators, we use a machine in the Hadooop cluster. In R-MAT and random graph generator of Figure 4, existing generators run out of memory when the numbers of edges are beyond 10^6 and 10^7, respectively, while TeG continues to run. The figure shows TeG generates at least 512× larger graphs than existing generators. Note that all three generators make a graph with 1 billion edges in less than a minute. We also evaluate TeG for machine scalability; TeG shows a linear machine scalability.

C. Visualization

The visualization module of TeGViz provides a quick and useful view of tera-scale graphs. Tera-scale graphs are difficult to visualize due to the amount of information overflowing the resolution of a typical screen. TeGViz uses distributed systems to efficiently generate visualization of the graph; such visualization helps user easily find interesting characteristics such as regularities, communities, and anomalies including near spokes, near cliques and spikes. TeGViz uses the following three plots to visualize large graphs.

1) Spy Plot: Spy plot shows connectivity patterns between nodes and community structures in a graph. For example, the spy plot of a Web graph shown in Figure 5 shows a high activity region in the lower left area.

2) Distribution Plot: Distribution plot (e.g., degree distribution and triangle distribution) reveals abundant information on the regularities as well as deviating patterns of a graph. For example, in many cases degree distribution of a real world graph follows a power-law; anomalous nodes with strange number of neighbors are represented by "spikes" in the degree distribution plot [9].

3) Correlation Plot: Correlation plot of two features of a graph helps find correlations and anomalies in nodes. For example, correlation plot of degree and triangle of nodes can be used to detect anomalous nodes which form a near-clique with its neighbors [10].

Since the visualization is performed in a fully distributed way to process data residing in a Hadoop cluster, the visualization finishes quickly. For example, generating a spy plot from a billion scale graph takes less than a minute.

III. Demonstration Plan

A. Present State of Demo

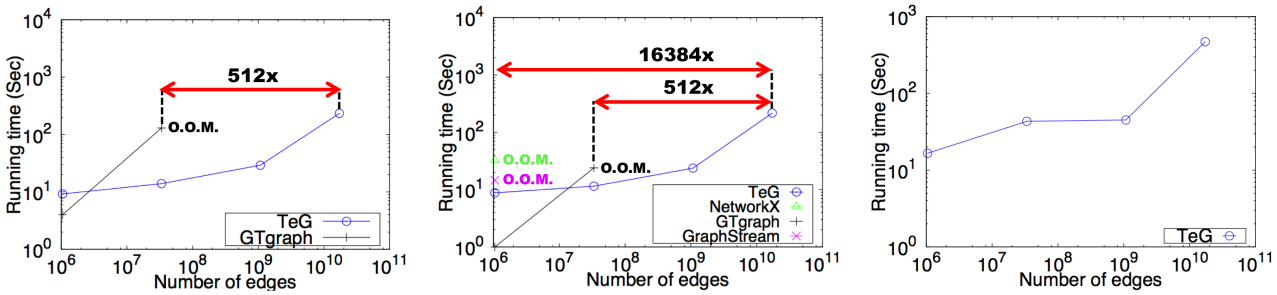
Our current prototype generates and visualizes a billion-scale graph within 1 minute.

B. Demonstration Details

The audience will be invited to 1) generate billion-scale graphs which existing generators cannot easily create, and 2) visualize it. We will connect to our remote cluster using our laptop. We will also pre-install Hadoop on our laptop to run our software using pseudo-distributed mode in case the network connection is not stable.

Graph Generation. We will let audiences generate billion-scale graphs. Audiences can select one of thee three types of graphs: random, R-MAT, and Kronecker. Users can specify parameters for each type of graph to customize the characteristics of the graphs that they want to generate.

Graph Visualization. TeGViz visualizes massive graphs using three types of plots: spy plot, distribution plot, and correlation plot. We will mainly focus on the spy plot since



(a) R-MAT Graph Generator (b) Random Graph Generator (c) Kronecker Graph Generator

Fig. 4: Comparison of the running time between TeG and existing generators. The label 'O.O.M.' means 'Out of Memory'. In R-MAT and random graph, existing generators run out of memory when the numbers of edges are beyond 10^6 and 10^7, respectively, while TeG continues to run. Note that TeG generates at least 512× larger graphs than existing generators. Note also that all three generators make a graph with 1 billion edges in less than a minute.

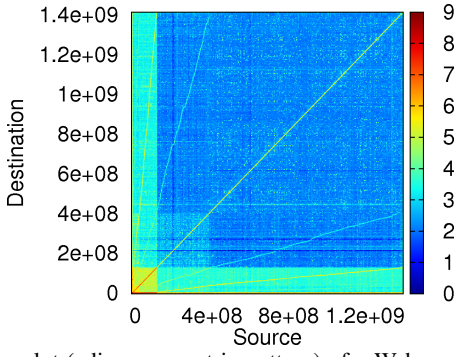


Fig. 5: Spy plot (adjacency matrix pattern) of a Web graph with 1.4 billion nodes and 6.6 billion edges. Note that the dense region in the lower left area is clearly identified.

it directly shows the connectivity and community patterns of the generated graph quickly. Generating distribution and correlation plots on the fly requires more time since it needs statistics of the graph like degree distribution and PageRank; thus we will show these plots using pre-computed statistics.

IV. Related Work

Several graph generators have been proposed. In [1], Bader et al. propose GTgraph, a synthetic random graph generator which supports random, R-MAT, and SSCA♯2 graphs. GraphStream [2] is a graph library that includes graph generators for random, Barabasi-Albert, and grid graphs. NetworkX [3], a python graph library, provides random, line, bipartite graph generators, etc. Although they support several graph generators, they are designed to run on a single machine, and thus cannot generate very large graphs exceeding the capacity of a single machine. On the other hand, TeGViz is carefully designed to generate very large graphs and supports a wide range of graph generators for both random and realistic graphs. The Parallel BGL [11] is a parallel C++ graph generator which provides random, R-MAT, SSCA graphs, etc. Unlike TeGViz which uses Hadoop, Parallel BGL uses MPI, and thus the burden of data management and fault tolerance is up to the user which hinders its usability.

V. Conclusion

In this paper we proposes TeGViz, a distributed tera-scale graph generation and visualization tool that supports a wide range of large-scale graphs including random and realistic graphs. TeGViz is carefully designed to work in an embarrassingly parallel way, and the task is evenly divided into each machine. TeGViz is scalable and nimble: e.g., TeGViz creates at least 512× large graphs than its competitors, and processes billion-scale graphs in less than a minute. TeGViz also visualizes the generated graph to give users insights on the connectivity and community patterns in it.

Future works include extending the work to generate and visualize time-evolving graphs, or tensors [12].

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) Grant funded by the Korean Government(MSIP)(No. 2013R1A1A1064409).

References

[1] D. Bader and K. Madduri, "Gtgraph: A synthetic graph generator suite," http://www.cse.psu.edu/madduri/software/GTgraph/, 2006.

[2] "Graphstream library," http://graphstream-project.org/.

[3] "Networkx information," http://networkx.github.io/.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters." in *OSDI*. USENIX Association, 2004, pp. 137–150.

[5] "Hadoop information," http://hadoop.apache.org/.

[6] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *SDM*, 2004.

[7] J. Leskovec, D. Chakrabarti, J. M. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication." in *PKDD*, 2005.

[8] U. Kang, D. H. Chau, and C. Faloutsos, "Pegasus: Mining billion-scale graphs in the cloud," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*, 2012, pp. 5341–5344.

[9] U. Kang, J. Y. Lee, D. Koutra, and C. Faloutsos, "Net-ray: Visualizing and mining billion-scale graphs," in *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, 2014, pp. 348–361.

[10] U. Kang, B. Meeder, E. Papalexakis, and C. Faloutsos, "Heigen: Spectral analysis for billion-scale graphs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 2, pp. 350–362, February 2014.

[11] "Parallel bgl information," http://www.boost.org/.

[12] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos, "Haten2: Billion-scale tensor decompositions," in *ICDE*, 2015.