Implementation and Evaluation of Distributed Graph Sampling Methods with Spark

Fangyan Zhang

Song Zhang (Advisor)

Christopher Lightsey

Mississippi State University, Mississippi State, MS, USA

EI 2018, Burlingame, California USA February 27, 2019

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

MISSISSIPPI STATE UNIVERSITY

Outline

- Introduction
- Graph sampling methods
- Methodology
- Implementation and usage
- Experiment and results
- Analysis
- Conclusion

Introduction

- This is an extension of the previous work (Graph Sampling for Visual Analytics)
- Why do we develop distributed graph sampling methods?
  - Sampling Efficiency

Sampling Methods

Distributed Node Sampling	Distributed Edge Sampling	Distributed Topology-based Sampling
Random Nodes (RN)	Random Edge (RE)	Breadth-First (BF)
Random Node-Edge (RNE)	Random Hybrid (RH)	Forest Fire (FF)
Random Node-neighbor (RNN)	Induced Edge (IE)	Snowball (SB)

Methodology

- Distributed Node Sampling
  - Given a graph $G = (V, E)$, where V and E are vertices and edges of G.
  - Let G's degree sequence be $\{1, 2 \ldots i \ldots k\}$, 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:

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

  - If a graph is distributed into n partitions, then the expected nodes in one partition can be represented as:

$E_p(NS(r)) = N_p(1)*r + N_p(2)*r + \ldots + N_p(t)*r$, where t is maximum degree in partition p, and $t <= k$.

  - If the graph has n partitions, then for overall partitions: $E_{distributed}(NS(r)) = E_1(NS(r)) + E_2(NS(r)) + \ldots + E_n(NS(r)) = \sum_{t=1}^{n} E_t(NS(r))$, Since $N_1(i) + N_2(i) + \ldots + N_n(i) = N(i)$, then:

$$E_{distributed}(NS(r)) = \sum_{t=1}^{n} E_t(NS(r)) = N(1)*r + N(2)*r + \ldots + N(k)*r = \sum_{i=1}^{k} \sum_{t=1}^{n} N_t(i) * r = r* \sum_{i=1}^{k} N(i) \quad (2)$$

Methodology

- Distributed Edge Sampling
  - Suppose the degree original graph is {*1, 2,…i… 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: $E\,(ES(r)) = N\,(1) * [1-(1-\frac{1}{|E|})]^{r|E|} + N\,(2) * [1-(1-\frac{2}{|E|})]^{r|E|} + \ldots + N\,(k) * [1-(1-\frac{k}{|E|})]^{r|E|} = \sum_{t=1}^{k} N(t) * [1-(1-\frac{t}{|E|})]^{r|E|}$ (3)
  - $E_p\,(ES(r)) = N_p\,(1) * [1-(1-\frac{1}{|E|})]^{r|E|} + N_p\,(2) * [1-(1-\frac{2}{|E|})]^{r|E|} + \ldots + N_p\,(t) * [1-(1-\frac{t}{|E|})]^{r|E|}$, where *t* is maximum degree in partition *p*, and *t* <= *k*. If graph has *n* partitions, then for overall partitions: $E_{distributed}\,(ES(r)) = E_1\,(ES(r)) + E_2\,(ES(r)) + \ldots + E_n\,(ES(r)) = \sum_{t=1}^{n} E_t\,(ES(r))$,
  - $E_{distributed}(ES(r)) = \sum_{t=1}^{n} E_t\,(ES(r)) = N\,(1) * [1-(1-\frac{1}{|E|})]^{r|E|} + N\,(2) * [1-(1-\frac{2}{|E|})]^{r|E|} + \ldots + N\,(k) * [1-(1-\frac{k}{|E|})]^{r|E|} = \sum_{i=1}^{k}\sum_{t=1}^{n} N_t(i) * [1-(1-\frac{i}{|E|})]^{r|E|} = \sum_{i=1}^{k} N(i) * [1-(1-\frac{i}{|E|})]^{r|E|}$ (4)

Methodology

- Distributed Topology-based Sampling
- Two challenges:
  - Not easy to create visited index
  - Multiple unconnected components in a graph.
- Solution
  - Two stages: vertex labeling and sampling
  - Check components in the graph
  - Indicate each vertex with an index number

Distributed Breadth First Sampling

Input: Graph G = (V, E), sampling rate θ
Output: Sampled graph $G_s = (V_s, E_s)$
Begin
  // cache sampling rate on each machine
  Broadcast (θ)
  $G_c \leftarrow$ *add one vertex attribute (index* $\leftarrow$ *MaxValue)*
  step $\leftarrow$ *1*
  componentsList $\leftarrow$ components in G_c
  Repeat: *each component c in* G_c
    startNode $\leftarrow$ *choose a random node in c*
    iteratively modify vertex attribute in c (index $\leftarrow$ *step)*
    step $\leftarrow$ *step + 1*
  Until *termination;*
  Repeat: *s in 1 to step*
      check the number of vertices when index < s
  Until *the number of vertices is satisfied*
  $G_{subgraph} \leftarrow G_c((V, index < s), E)$
End algorithm

Implementation

- Platforms or packages used
  - Spark (a fast and general engine for large-scale data processing)
  - GraphX (Apache Spark's API for graphs and graph-parallel computation)
  - Pregel (A system for large-scale graph processing, developed by Google)

- The distributed sampling algorithms are written in Scala language and compiled into a JAR file for distribution.

Usage

- Package Usage (Example)

```
import msu.dasi.distributedSamplingMethods._
……
val conf = new SparkConf().setAppName("Sampling").setMaster("local[*]")
val sc = new SparkContext(conf)
val graph = GraphLoader.edgeListFile(sc, "…/friendster.txt",
true).partitionBy(PartitionStrategy.RandomVertexCut)
val percent = 0.15
randomNodeGraph = randomNode(sc, percent, graph)
……
```

Methodology

- Skew divergence is used to evaluate sampling results.
- KL Divergence = $\sum(P * log \frac{P}{Q})$
- $SD(P, Q, \alpha) = KL[\alpha P + (1 - \alpha)Q || \alpha Q + (1 - \alpha)P]$, where α is 0.99.

- Graph properties used in comparison
  - Degree Distribution (DD)
  - Average Neighbor Degree Distribution (ANDD)
  - PageRank Distribution (PRD)
  - Triangle Distribution (TD)
  - Local Clustering Coefficient Distribution (LCCD)

Graph Datasets

Datasets	Type	Model	Nodes	Edges
Facebook	Undirected	SNAP	4,039	88,234
Amazon	Undirected	SNAP	334,863	925,872
Friendster	Undirected	SNAP	65,608,366	1,806,067,135

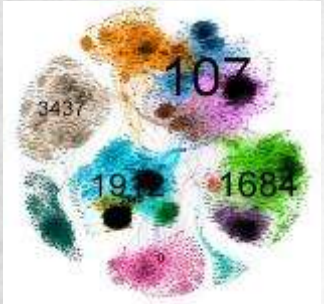
SNAP: https://snap.stanford.edu/data/

Sampling Rate:

- 15% based on vertices
- 25% based on vertices

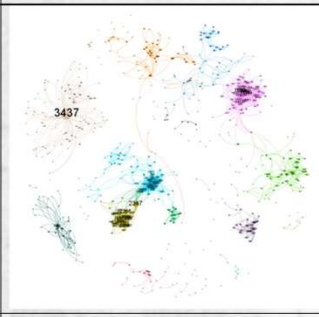
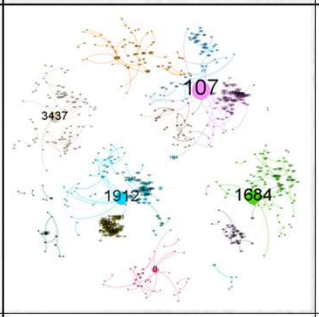
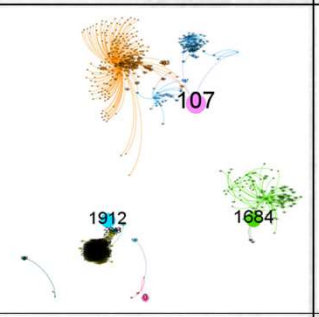
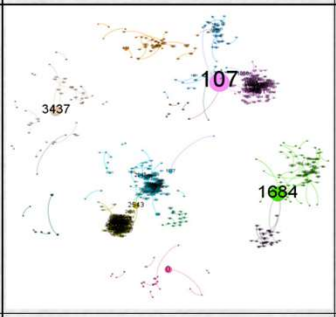
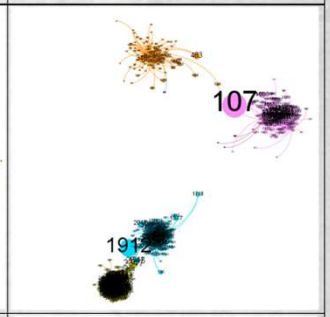
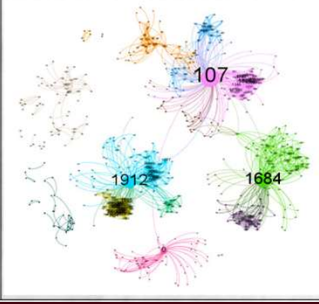
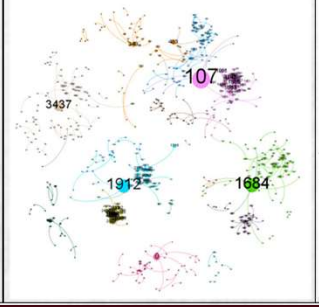
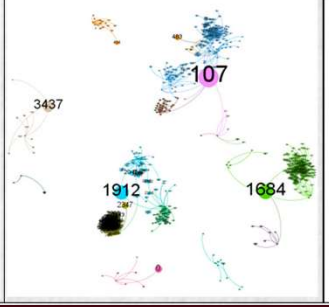
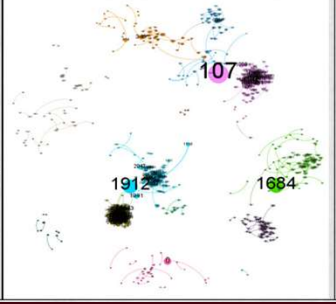
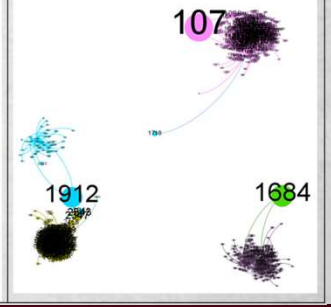
5 different runs

Visual Comparison Results



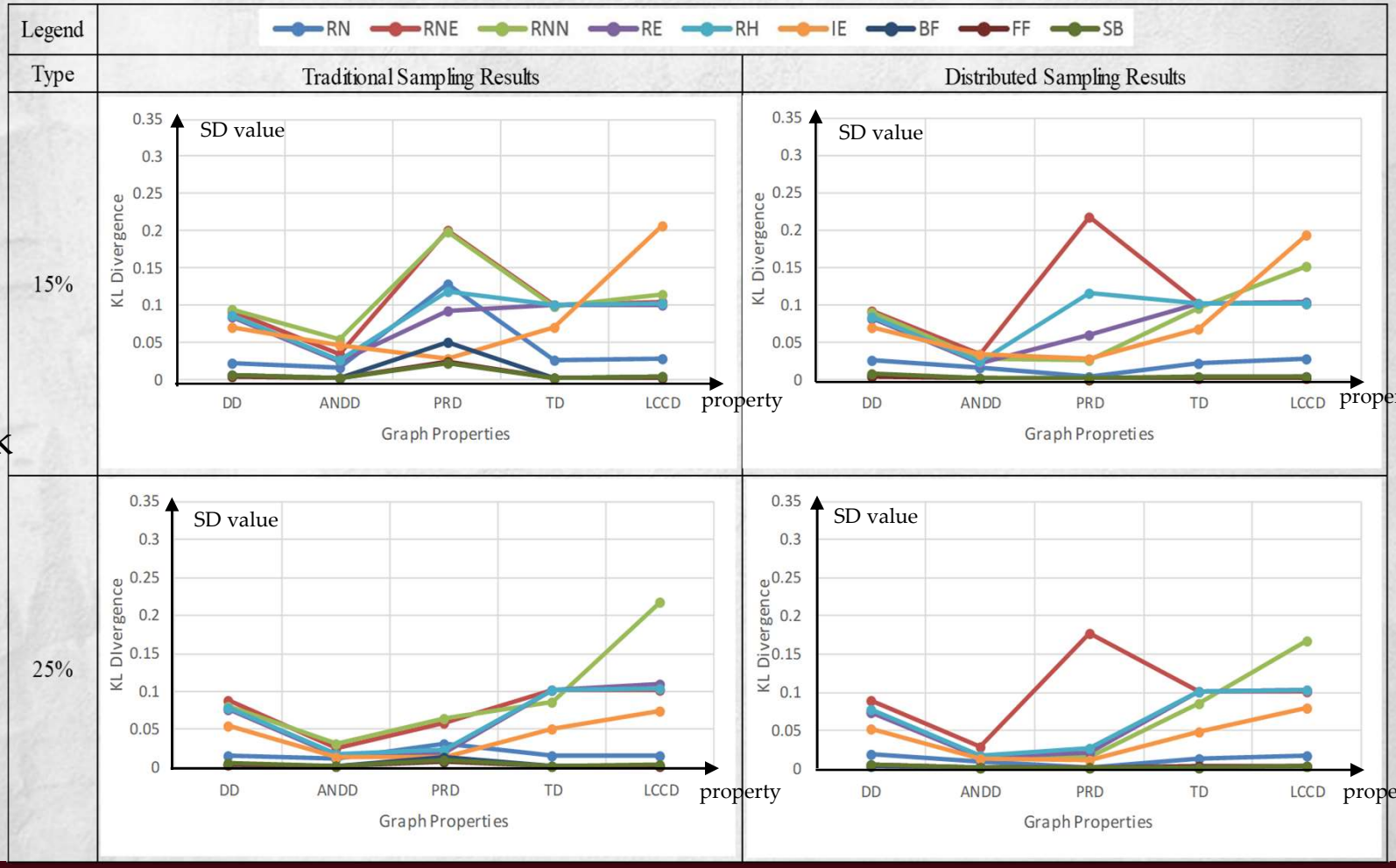
Original Facebook Graph

Sampling rate: 15%

Sampling Methods	Ramdom Node	Random Node-Edge	Random Node-Neighbor	Random Edge	Induced Edge
Traditional Sampleing Results					
Distributed Sampling Results					

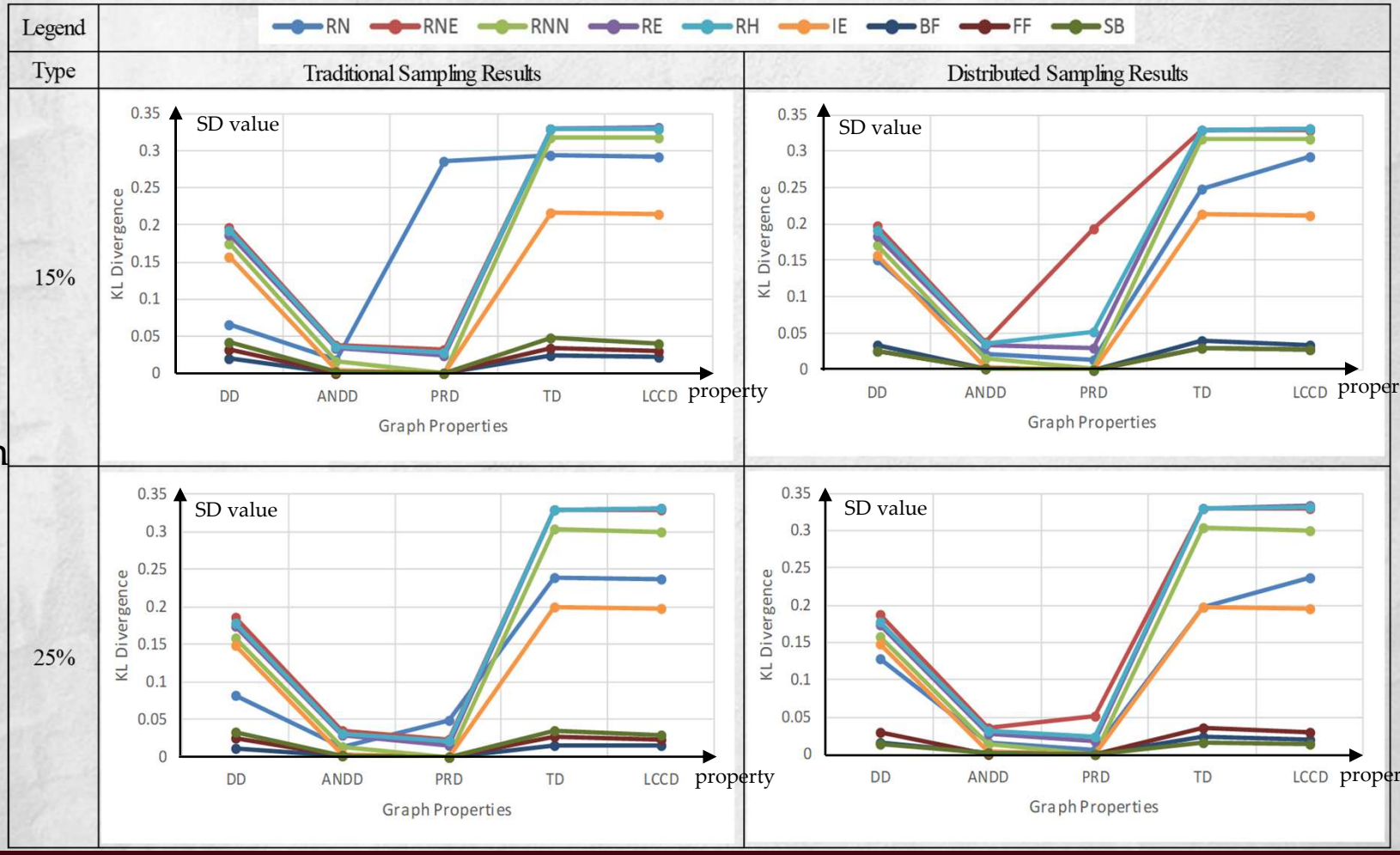
Statistical Comparison Results

Facebook

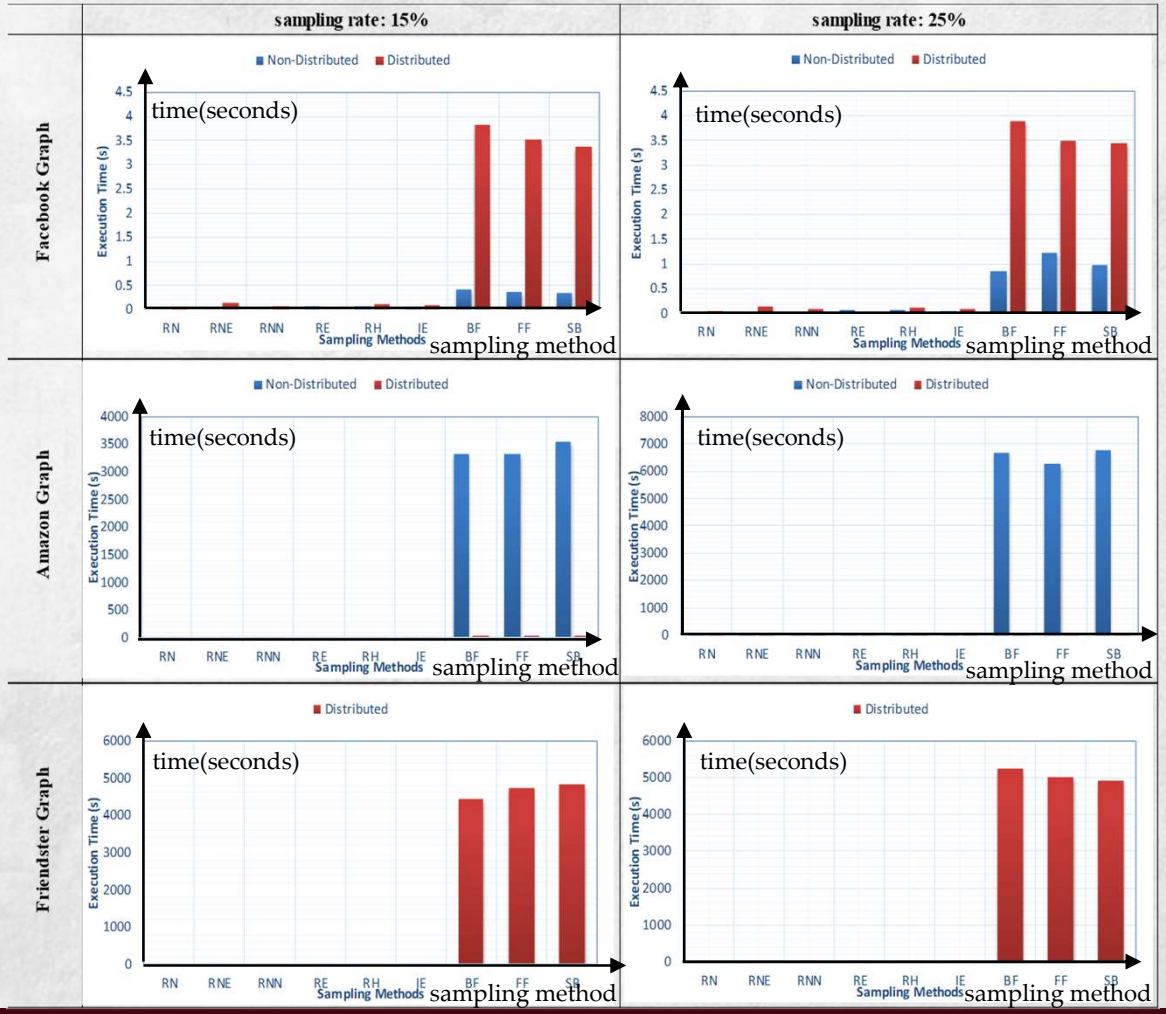


Statistical Comparison Results

Amazon



Efficiency Comparison Results



Analysis: Visual Comparison

- First, distributed random sampling methods have very similar performance with non-distributed random sampling methods in preserving spatial coverage.
- Sometimes distributed sampling methods have better ability to preserve the number of clusters.
- For topology-based sampling, their ability to preserve spatial coverage and clusters is more random.
- 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.

Analysis: Statistical Comparison

- Generally, distributed sampling methods have similar performance to the non-distributed in preserving graph properties except for a few sampling methods.
- Both non-distributed and distributed random sampling methods (e.g. random node sampling, random edge sampling) are unable to behave as well as topology-based sampling in preserving degree, triangle, and local clustering coefficient, particularly for large graphs.

Analysis: Efficiency Comparison

- For small graphs, non-distributed sampling methods have higher efficiency than distributed sampling methods.
- 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 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.

Analysis

- Statistical comparison
- Visual comparison
- Efficiency comparison
- Scalability

Questions?

Thank you!

Fangyan Zhang
Email: fz56@msstate.edu