# Centralities in Large Networks: Algorithms and Observations

U Kang
SCS, CMU

Spiros Papadimitriou*
Google

Jimeng Sun
IBM T.J. Watson

Hanghang Tong
IBM T.J. Watson

**Abstract**

Node centrality measures are important in a large number of graph applications, from search and ranking to social and biological network analysis. In this paper we study node centrality for very large graphs, up to billions of nodes and edges. Various definitions for centrality have been proposed, ranging from very simple (e.g., node degree) to more elaborate. However, measuring centrality in billion-scale graphs poses several challenges. Many of the "traditional" definitions such as closeness and betweenness were not designed with scalability in mind. Therefore, it is very difficult, if not impossible, to compute them both accurately and efficiently. In this paper, we propose centrality measures suitable for very large graphs, as well as scalable methods to effectively compute them. More specifically, we propose effective closeness and LINERANK which are designed for billion-scale graphs. We also develop algorithms to compute the proposed centrality measures in MAPREDUCE, a modern paradigm for large-scale, distributed data processing. We present extensive experimental results on both synthetic and real datasets, which demonstrate the scalability of our approach to very large graphs, as well as interesting findings and anomalies.

## 1 Introduction

*Centrality* is widely-used for measuring the relative importance of nodes within a graph [5, 12]. For example, who are the most well-connected people in a social network? Or who are critical for facilitating the transmission of information in a terrorist network [21]? Which proteins are the most important for the lethality of a cell in protein interactions biological network [16]? In general, the concept of centrality has played an important role in the understanding of various kinds of networks by researchers from computer science, network science, sociology, and recently emerging 'computational social science' [23].

Traditionally, centrality has typically been studied for graphs of relatively small size. However, in the past few years, the proliferation of digital collection of data has led to the collection of very large graphs, such as the web, online social networks, user preferences, online communications, and so on. Many of these networks reach billions of nodes and edges requiring terabytes of storage.

**1.1 Challenges for Large Scale Centralities** Measuring centrality in very large graphs poses two key challenges.

First, some *definitions* of centrality have inherently high computational complexity. For example, shortest-path or random walk betweenness [6, 25] have complexity at least $O(n^3)$ where $n$ is the number of nodes in a graph. Furthermore, some of the faster estimation algorithms require operations, which are not amenable to parallelization, such as all-sources breadth-first search. Finally, it may not be straightforward or even possible to develop accurate approximation schemes. In summary, centrality measures should ideally be designed with scalability in mind from the outset. Traditionally, this has not always been the case [12]. However, with the recent availability of very large networks, there is a clear need for scalable measures.

Second, even if a centrality measure is designed in a way that avoids expensive or non-parallelizable operations, developing *algorithms* that are both efficient, scalable, and accurate is not straightforward. Clever approximation or parallelization schemes may need to be employed, in order to achieve all of these goals.

**1.2 Problem Definitions** In this paper we tackle the problem of efficiently and effectively measuring centrality for billion-scale networks. More specifically, we address the following problems:

1. **Careful Design.** How can we carefully design centrality measures that avoid inherent limitations to scalability and parallelization, yet are sufficiently informative?
2. **Algorithms.** How can we compute the large scale centralities quickly for billion-scale graphs? How can we leverage modern, large-scale, distributed data processing infrastructures?
3. **Observations.** What are key patterns and observations on centralities in large, real world networks?

In particular, we study popular definitions for three types of centrality: degree (baseline), closeness (diameter-based), and betweenness (flow-based), which cover the spectrum from simpler to more elaborate. Except for degree, the

*Work while at IBM.

other two centrality measures, closeness and betweenness, are prohibitively expensive to compute, and thus impractical for large networks. On the other hand, although simple to compute, degree centrality gives limited information since it is based on a highly local view of the graph around each node. We need a set of centrality measures that enrich our understanding of very large networks, and are tractable to compute.

**1.3 Our Contributions** To address the difficulty of computing centralities on large-scale graphs, we propose two new measures, one diameter-based and one flow-based, respectively.

The first measure we propose is the *effective closeness* of a node, which approximates the average shortest path length starting from the node in question. We employ sketches and MAPREDUCE [9] in order to make its computation tractable.

The second measure we propose is LINERANK, which intuitively measures the "flow" through a node. Our notion of "flow" is derived by finding the stationary probabilities of a random walk on the *line graph* of the original matrix and then aggregating the flows on each node[1]. In addition, we extend the line graph to weighted graphs as well as directed graphs. The advantage of this approach is that the stationary probabilities can be efficiently computed in a distributed setting through MAPREDUCE. However, materialization of the line graph is prohibitively expensive in terms of space. We show that we can decompose the computation in a way that uses a much sparser matrix than the original line graph, thus making the entire computation feasible.

Finally, we analyze the centralities in large, real-world networks with these new centrality algorithms to find important patterns.

In summary, the main contributions in this paper are the following:

1. **Careful Design.** We propose two new large-scale centralities: *effective closeness*, a diameter-based centrality, and LINERANK, a flow-based centrality. Both are carefully designed with billion-scale networks in mind from the beginning.
2. **Algorithms.** We develop efficient parallel algorithms to compute *effective closeness* and LINERANK on MAPREDUCE by using approximation and efficient line graph decomposition of billion scale graphs. We perform experiments to show that both of our proposed centralities have linear scale-up on the number of edges and machines.
3. **Observations.** Using our large-scale centralities, we analyze real-world graphs including YahooWeb, Enron and DBLP. We report important patterns including high effective closeness for high degree nodes, the distinguishing ability of effective closeness for low degree nodes, and the ability of LINERANK for discriminating relatively high degree nodes.

[1] See Section 4 for the formal definition of line graph.

| Symbol | Definition |
|---|---|
| $n$ | number of nodes in a graph |
| $m$ | number of edges in a graph |
| $A$ | adjacency matrix of a graph |
| $indeg(v)$ | in-degree of node $v$ |
| $outdeg(v)$ | out-degree of node $v$ |
| $N(r, v)$ | number of neighbors of node $v$ within $r$ steps |
| $G$ | original graph |
| $L(G)$ | line graph of the original graph $G$ |
| $S(G)$ | source incidence matrix of the graph $G$ |
| $T(G)$ | target incidence matrix of the graph $G$ |

Table 1: Table of symbols

The rest of the paper is organized as follows: Section 2 presents related work on node centrality. In Section 3 we define our proposed large scale centralities. In Section 4 we develop scalable algorithms to efficiently and effectively evaluate those measures. Section 5 presents scalability and accuracy results on synthetic and real data sets. After showing patterns and observations of centralities in large, real world graphs in Section 6, we conclude the paper in Section 7. Table 1 lists the symbols used in this paper.

## 2 Related Work

Related work forms two groups: centrality measures on graphs and parallel graph mining using HADOOP.

**2.1 Centrality Measures on Graphs** Centrality has attracted a lot of attentions as a tool for studying various kinds of networks including social, information, and biological networks [12, 5, 16]. The centrality of a node in a network is interpreted as the importance of the node. Many centrality measures have been proposed based on how the importance is defined.

In this section, we discuss various centrality measures around the three main centrality groups [12, 5] which represent distinguished types of walks.

**Degree related measures.** The first group of the centrality measures is the degree related measures. The degree centrality, the simplest yet the most popular centrality measure, belongs to this group. The degree centrality $c_i^{DEG}$ of node $i$ is defined to be the degree of the node.

A way of interpreting the degree centrality is that it counts the number of paths of length 1 that emanate from a node. A generalization of the degree centrality is the $K$-$path$ centrality which is the number of paths less than or equal to $k$ that emanate from a node. Several variations of the $K$-$path$ centrality exist based on the type of the path: geodesic,

edge-disjoint, vertex-disjoint $K$-$path$ are among them [5].

Another line of centralities are based on the 'walk' on the graph. The Katz centrality [20] counts the number of walks starting from a node, giving penalties to longer walks. In a mathematical form, the Katz centrality $c_i^{KATZ}$ of node $i$ is defined by $c_i^{KATZ} = e_i^T(\sum_{j=1}^{\infty}(\beta A)^j)\mathbf{1}$ where $e_i$ is a column vector whose $i$th element is 1, and all other elements are 0. The $\beta$ is a positive penalty constant to control the weight on the walks of different length. A slight variation of the Katz measure is the Bonacich centrality [4] which allows the negative $\beta$. The Bonacich centrality $c_i^{BON}$ of node $i$ is defined to be $c_i^{BON} = e_i^T(\frac{1}{\beta}\sum_{j=1}^{\infty}(\beta A)^j)\mathbf{1}$ where the negative weight allows to subtract the even-numbered walks from the odd-numbered walks which have an interpretation in exchange networks [5]. The Katz and the Bonacich centralities are special cases of the Hubbell centrality [15]. The Hubbell centrality $c_i^{HUB}$ of node $i$ is defined to be $c_i^{HUB} = e_i^T(\sum_{j=0}^{\infty}X^j)\mathbf{y}$ where $X$ is a matrix and $y$ is a vector. It can be shown that $X = \beta A$ and $y = \beta A\mathbf{1}$ lead to the Katz centrality, and $X = \beta A$ and $y = A\mathbf{1}$ lead to the Bonacich centrality.

Except for degree, most of variations require some parameter, which may not be easy to determine in real networks. Also computationally, degree is the only one that can be efficiently measured for large networks, which will be served as the baseline measure in this paper.

**Diameter related measures.** The second group of the centrality measures is diameter related measures, which count the length of the walks. The most popular centrality measure in this group is the Freeman's closeness centrality [12]. It measures the centrality by computing the average of the shortest distances to all other nodes. Let $S$ be the matrix whose $(i, j)$th element contains the length of the shortest path from node $i$ to $j$. Then, the closeness centrality $c_i^{CLO}$ of node $i$ is defined to be $c_i^{CLO} = e_i^T S\mathbf{1}$.

As we will see in Section 6, diameter-based measures are effective in differentiating low degree nodes. However, the existing diameter based measure does not scale up and therefore efficient computational method needs to be developed, which is one of the focus in this paper.

**Flow related measures** The last group of the centrality measures is the flow related measures. It is called 'flow related' since the information flowing through edges are considered. The most well-known centrality in this group is the Freeman's betweenness centrality [12]. It measures how much a given node lies in the shortest paths of other nodes. Let $b_{jk}$ is the number of shortest paths from node $j$ to $k$, and $b_{jik}$ be the number of shortest paths from node $j$ to $k$ that passes through node $i$. The betweenness centrality $c_i^{BET}$ of node $i$ is defined to be $c_i^{BET} = \sum_{j,k}\frac{b_{jik}}{b_{jk}}$. The naive algorithm for computing the betweenness involves all-pair shortest paths which require $\Theta(n^3)$ time and $\Theta(n^2)$ storage. Brandes [6] made a faster algorithm by running $n$ single-source-shortest-path algorithms which require $O(n + m)$ space and run in $O(nm)$ and $O(nm + n^2 \log n)$ time on unweighted and weighted networks, respectively, where $n$ is the number of nodes and $m$ is the number of edges in a graph.

Newman [25] proposed an alternative betweenness centrality based on random walks on the graph. The main idea is that instead of considering shortest paths, it considers all possible walks and compute the betweenness from them. Specifically, let $R$ be the matrix whose $(j, k)$th element $R_{jk}$ contains the probability of a random walk, starting from $j$ with the absorbing node $k$, passing through the node $i$. Then, the Newman's betweenness centrality $c_i^{NBE}$ of node $i$ is defined to be $c_i^{NBE} = \sum_{j\neq i\neq k} R_{jk}$. Computing the Newman's betweenness centrality requires $O(mn^2)$ time which is prohibitively expensive.

None of the existing flow related measures are scalable to large networks. In this paper, we propose LINERANK, a new flow-based measure scalable to large networks.

### 2.2 Parallel Graph Mining using HADOOP

MAPREDUCE is a distributed programming framework [10] for processing web-scale data. MAPREDUCE has two benefits: (a) The data distribution, replication, fault-tolerance, and load balancing is handled automatically; and furthermore (b) it uses the familiar concept of functional programming. The programmer needs to define only two functions, a *map* and a *reduce*. The general framework is as follows [22]: (a) the *map* stage reads the input file and emits (key, value) pairs; (b) the *shuffling* stage sorts the output and distributes them to reducers; (c) the *reduce* stage processes the values with the same key and emits another (key, value) pairs which become the final result.

HADOOP [1] is the open source equivalent of MAPREDUCE. HADOOP uses its own distributed file system HDFS, and provides a high-level language called PIG [26]. Due to its excellent scalability, ease of use, and cost advantage, HADOOP has been used for important graph mining algorithms (see [27, 19, 18, 17]). Other variants which provide advanced MAPREDUCE-like systems include SCOPE [7], Sphere [14], and Sawzall [28].

## 3 Large-scale Centrality Measures

In this section, we propose centrality measures which are designed for large-scale, distributed computation. We first review well-known centrality measures and analyze the computations required. While some centralities are easier to compute, others suffer from inherent limitations in achieving scalability, as explained in Section 2. We propose alternative centrality measures that follow similar motivation and intuition as existing measures, but are much more suitable for distributed computation on very large graphs.

Following the classification of centralities in Section 2,

we focus on the three most common and representative types of centrality measures: degree (local), diameter-based (closeness), and flow-based (betweenness).

**3.1 Degree** Degree centrality has a very simple and intuitive definition: it is the number of neighbors of a node. Despite, or perhaps because of its simplicity, it is very popular and used extensively. Not surprisingly, it is also the easiest to compute. The degree centrality vector $C^{DEG}$ of a graph with an adjacency matrix $A$ can be represented in matrix-vector multiplication form by

$$C^{DEG} = A\mathbf{1}. \tag{3.1}$$

Thus, the degree centrality of a large network can be exactly computed by a large scale matrix-vector multiplication. The major limitation of degree based centrality is that it only captures the local information of a node. In many applications, we need more informative measures that can further distinguish among nodes that have almost equally low degrees, or almost equally high degrees (see Section 6).

**3.2 Diameter-based Measures** Closeness centrality is the most popular diameter-based centrality measure. While degree centrality considers only one-step neighbors, closeness centrality considers all nodes in the graph, and gives high score to nodes which have short average distances to all the other nodes. Closeness of a node is typically defined as the inverse of the average over the shortest distances to all other nodes; to simplify formulas we omit the inverse. Exact computation requires an all-pairs shortest paths algorithm. Unfortunately, this operation requires $O(n^3)$ time. For the billion-scale graphs we consider in this work, computing closeness centrality is prohibitively expensive. To address this computational issue, we propose to use an accurate approximation instead of exact computation, leading to the following notion of centrality.

DEFINITION 1. (**EFFECTIVE CLOSENESS**) *The effective closeness centrality $C^{ECL}(v)$ of a node $v$ is defined as the approximate average distance from $v$ to all other nodes.*

We will next define the notion of "approximate" more precisely, by describing the approximation scheme we employ. Let $N(r, v)$ be the number of neighbors of node $v$ within $r$ steps, and $N_v(r)$ be the number of nodes whose shortest distances to $v$ is $r$. Notice that $N_v(r) = N(r, v) - N(r-1, v)$. Based on these quantities, standard closeness can be defined by

$$\begin{aligned} closeness &= \frac{\sum_{r=1}^{d} r \cdot N_v(r)}{n} \\ &= \frac{\sum_{r=1}^{d} r \cdot (N(r,v) - N(r-1,v))}{n} \end{aligned} \tag{3.2}$$

where $d$ is the diameter of the graph and $n$ is the number of nodes. Let's assume that we can easily get $\hat{N}(r, v)$, an unbiased estimate of $N(r, v)$. Define $\hat{N}_v(r)$ to be $\hat{N}(r, v) - \hat{N}(r-1, v)$. By the linearity of expectation, $\hat{N}_v(r)$ gives an unbiased estimate of $N_v(r)$. Thus, by using this approximation, we can define the effective closeness $C^{ECL}(v)$ by

$$\begin{aligned} C^{ECL}(v) &= \frac{\sum_{r=1}^{d} r \cdot \hat{N}_v(r)}{n} \\ &= \frac{\sum_{r=1}^{d} r \cdot (\hat{N}(r,v) - \hat{N}(r-1,v))}{n} \end{aligned} \tag{3.3}$$

The remaining question is how to efficiently get an accurate approximation $\hat{N}(r, v)$. For this purpose, we use the Flajolet-Martin [11] algorithm for estimating the number of unique items in a multiset. While many algorithms exist for the estimation (e.g., [3, 8, 13]), we choose the Flajolet-Martin algorithm because it gives an unbiased estimate, as well as a tight $O(\log n)$ space bound [2]. The main result of the Flajolet-Martin algorithm is that we can represent a set with $n$ unique nodes using a bitstring of size $O(\log n)$, and the bitstring can be used to estimate the number $n$ of unique items in the set. From its construction, the bitstring of the union of two sets can be obtained by bitwise-OR'ing the bitstrings of these sets. In our case, each node starts with a bitstring encoding a set containing only the node itself. At every step, each node updates its bitstring by bitwise-OR'ing with the bitstrings of its neighbors. This process continues until the bitstrings for all nodes converge.

**3.3 Flow-based Measures** Betweenness centrality is the most common and representative flow-based measure. In general, the betweenness centrality of a node $v$ is the number of times a walker visits node $v$, averaged over all possible starting and ending nodes. Different types of walks lead to different definitions for betweenness centrality. In Freeman betweenness [12], the walks always follow the shortest path from starting to ending node. In Newman's betweenness [25], the walks are absorbing random walks. Both of these popular definitions require prohibitively expensive computations: the best algorithm for shortest-path betweenness has $O(n^2 \log n)$ complexity, while the best for Newman's betweenness has $O(mn^2)$ complexity.

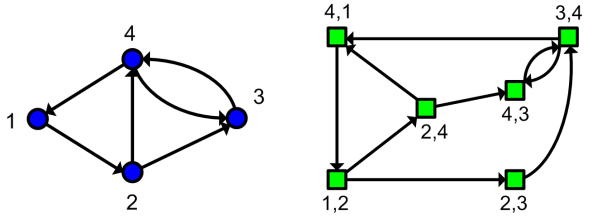
Since existing measures do not scale well, we propose a new flow-based measure, called LINERANK. The main idea is to measure the importance of a node by aggregating the importance score of its incident *edges*. This represents the amount of information that flows to the node. Several non-trivial questions need to be addressed for LINERANK to be useful. First, how can we define the edge importance? Second, how do we compute it efficiently?

For the first question, we define the edge importance by the probability that a random walker, visiting *edges* via *nodes* with random restarts, will stay at the edge. To define this random walk precisely, we induce a new graph, called directed line graph, from the original graph.

DEFINITION 2. (**DIRECTED LINE GRAPH**) *Given a directed graph G, its directed line graph $L(G)$ is a graph such*

*that each node of $L(G)$ represents an edge of G, and there is an edge from a node $e_1$ to $e_2$ in $L(G)$ if for the corresponding edges $(u_1, v_1)$ and $(u_2, v_2)$ in G, $v_1 = u_2$.*

For example, see a graph and its directed line graph in Figure 1. There is an edge from the node $(4, 1)$ to $(1, 2)$ in $L(G)$ since the edge $(4, 1)$ follows $(1, 2)$ in $G$.




(a) Original graph $G$ (b) Directed line graph $L(G)$

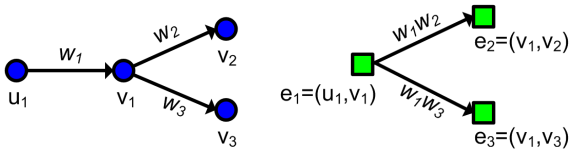
Figure 1: Original graph $G$ and its corresponding directed line graph $L(G)$. The rectangular nodes in (b) correspond to edges in (a). There is an edge from a node $e_1$ to $e_2$ in $L(G)$ if for the corresponding edges $(u_1, v_1)$ and $(u_2, v_2)$ in $G$, $v_1 = u_2$, or the first edge follows the second. For example, there is an edge from the node $(4, 1)$ to $(1, 2)$ in $L(G)$ since the edge $(4, 1)$ follows $(1, 2)$ in $G$.

Now think of a random walker visiting nodes on the line graph. The walker staying at a node at the current step will move to a neighboring node with high probability $c$, or to a random node with low probability $1 - c$, so that the walk mixes well. We seek the stationary probability of this random walk. Edges in the original graph are associated with the stationary probabilities by which we define LINERANK as follows.

DEFINITION 3. (**LINERANK**) *Given a directed graph G, the* LINERANK *score of a node $v \in G$ is computed by aggregating the stationary probabilities of its incident edges on the line graph $L(G)$.*

Another important question is how to determine edge weights in the line graph. The random walk in the line graph is performed with transition probabilities proportional to edge weights. For example, in Figure 2, the node $e_1$ in $L(G)$, which corresponds to the edge $(u_1, v_1)$ in $G$, transits to either $e_2 = (v_1, v_2)$ or $e_3 = (v_1, v_3)$ with the probability proportional to $w_2$ and $w_3$, respectively.

For an unweighted original graph, the line graph is also unweighted. However, for a weighted original graph, the line graph should have appropriate edge weights. We propose to multiply the weights of the adjacent edges in the original graph to compute the edge weights in the line graph. That is, assume two adjacent edges $e_1 \equiv (u_1, v_1)$ and $e_2 \equiv (v_1, v_2)$ in $G$ have weights $w_1$ and $w_2$, respectively. Then the edge $(e_1, e_2)$ in $L(G)$ have the weight $w_1 w_2$ where $e_1$ and $e_2$ are the corresponding nodes in $L(G)$ to $(u_1, v_1)$ and $(v_1, v_2)$ in




(a) Original graph $G$ (b) Directed line graph $L(G)$

Figure 2: Weights of the original graph $G$ and the corresponding directed line graph $L(G)$. The rectangular nodes in (b) correspond to edges in (a). If two consecutive edges $(u_1, v_1)$ and $(v_1, v_2)$ in $G$ have weights $w_1$ and $w_2$, respectively, then the corresponding induced edge in $L(G)$ have the weight $w_1 w_2$. For example, the edge $(e_1, e_2)$ in $L(G)$ has the weight $w_1 w_2$.

$G$, respectively. This weighting scheme enables a random walker to transit in proportion to the original edge weights in $G$, after normalization.

DEFINITION 4. (**WEIGHTS IN DIRECTED LINE GRAPH**) *If two consecutive edges $(u_1, v_1)$ and $(v_1, v_2)$ in G have weights $w_1$ and $w_2$, respectively, then the corresponding induced edge in $L(G)$ have the weight $w_1 w_2$.*

The remaining challenge is to compute LINERANK on the line graph. In the next section we show how to design efficient algorithms for LINERANK, as well as effective closeness, of billion-scale graphs using HADOOP [1], an open-source MAPREDUCE framework.

## 4 Large-scale Centrality Algorithms

In this section, we describe HADOOP algorithms to compute centralities for large-scale graphs. Specifically, we focus on effective closeness and LINERANK, and propose efficient algorithms.

**4.1 Effective Closeness** The effective closeness requires $\hat{N}(r, v)$, an approximation of $N(r, v)$ which is the number of neighbors of node $v$ within $r$ steps. As described in Section 3, we use Flajolet-Martin algorithm for the approximation. The HADOOP algorithm for the effective closeness iteratively updates the Flajolet-Martin (FM) bitstrings for every node. The crucial observation is that the bitstrings update operation can be represented in a form similar to matrix-vector multiplication [18]. Specifically, let $b(r - 1, v)$ be node $v$'s bitstring encoding the set of nodes within distance $r - 1$. Then the next-step bitstring $b(r, v)$ is computed by BITWISE-OR'ing the current bitstring $b(r - 1, v)$ of $v$ and the current bitstrings of the neighbors of $v$.

(4.4)
$$b(r, v) = b(r-1, v) \text{ BITWISE-OR } \{b(r-1, u) | (v, u) \in E\}$$

Since the above equation is a generalized form of matrix-vector multiplication, a repeated matrix-vector mul-

tiplication with BITWISE-OR customization computes the approximation $\hat{N}(r,v)$ and thus can compute the effective closeness using Equation (3.3), as shown in Algorithm 1. The *InitialBitstring*(line 2) and *DecodeBitstring*(line 11,13) create and decode the FM bitstrings. The $sum_{cur}$ and $sum_{next}$ variables are used to check whether $r$ reached the maximum diameter of the graph, and to finish the computation early if possible.

**Algorithm 1** Effective Closeness

**Input:** Edge $E = \{(i,j)\}$ of a graph $G$ with $|V| = n$
**Output:** Effective Closeness $C^{ECL} = \{(score_v)\}$

1: **for** $v = 1$ to $n$ **do**
2: $b(0,v) \leftarrow InitialBitstring$;
3: $C^{ECL}(v) = 0$;
4: **end for**
5: $sum_{next} \leftarrow 0$;
6: **for** $r = 1$ to $MaxIter$ **do**
7: $sum_{cur} \leftarrow sum_{next}$;
8: $sum_{next} \leftarrow 0$;
9: // Update effective closeness of nodes
10: **for** $v = 1$ to $n$ **do**
11: $\hat{N}(r-1,v) \leftarrow DecodeBitstring(b(r-1,v))$;
12: $b(r,v) =$ $b(r-1,v)$ BITWISE-OR $\{b(r-1,u)|(v,u) \in E\}$;
13: $\hat{N}(r,v) \leftarrow DecodeBitstring(b(r,v))$;
14: $C^{ECL}(v) =$ $C^{ECL}(v) + r \times (\hat{N}(r,v) - \hat{N}(r-1,v))$;
15: $sum_{next} = sum_{next} + C^{ECL}(v)$;
16: **end for**
17: // Check whether the effective closeness converged
18: **if** $sum_{next} = sum_{cur}$ **then**
19: break for loop;
20: **end if**
21: **end for**
22: $C^{ECL}(v) = C^{ECL}(v)/n$;

The effective closeness algorithm is much efficient than the standard closeness. The effective closeness requires $O(dm)$ time, where $d$ is the diameter of the graph and $m$ is the number of edges, since it requires at most $d$ matrix-vector multiplications. In contrast, the standard closeness requires $O(n^3)$ time, where $n$ is the number of nodes, which is much longer than $O(dm)$, given that real-world graphs are sparse($m << n^2$) with very small diameter, a phenomena known as "six degrees of separation".

**4.2 LINERANK** How can we compute the LINERANK efficiently? A naive algorithm would explicitly materialize the line graph of the original graph. However, the line graph can grow very large since a node $v$ with in-degree $\alpha$ and out-degree $\beta$ in the original graph will generate $\alpha\beta$ edges in the line graph. Thus, the number $|E_{L(G)}|$ of edges in the line graph is

$$|E_{L(G)}| = \sum_{v=1}^{n} indeg(v) \cdot outdeg(v). \tag{4.5}$$

Real-world graphs have nodes with very large in and out degrees, as the power-law degree distribution has long tails. Thus, even though the original graph is sparse, the line graph can be much denser than the original. For example, the line graph of the YahooWeb graph in Table 2 has 251 billion edges which is $\sim 250\times$ more edges than the original graph. Thus, explicit construction is not tractable for large graphs.

Our proposed main idea is to compute the LINERANK without explicitly constructing the line graph. It turns out that the weighted, directed line graph $L(G)$, in our Definition 4, has a decomposition into sparse matrices and thus LINERANK can be computed efficiently on those sparse matrices rather than on the dense matrix $L(G)$.

To describe the decomposition, we need to define two types of incidence matrices.

DEFINITION 5. (**SOURCE INCIDENCE MATRIX**) *The source incidence matrix $S(G)$ of a graph $G$ with $n$ nodes and $m$ edges is an $m \times n$ matrix with entries $S(G)_{ij} = w_i$ if the $i^{th}$ edge with the weight $w_i$ in $G$ has node $j$ as its source, and $S(G)_{ij} = 0$ otherwise.*

DEFINITION 6. (**TARGET INCIDENCE MATRIX**) *The target incidence matrix $T(G)$ of a graph $G$ with $n$ nodes and $m$ edges is an $m \times n$ matrix with entries $T(G)_{ij} = w_i$ if the $i^{th}$ edge with the weight $w_i$ in $G$ has node $j$ as its target, and $T(G)_{ij} = 0$ otherwise.*

Note that if the original graph is sparse, both the incidence matrices are sparse with exactly $m$ non-zero elements where each row contains only 1 non-zero element. Now we introduce our proposed decomposition of the weighted, directed line graph.

LEMMA 4.1. (**LINE GRAPH DECOMPOSITION**) *Given a directed, weighted graph $G$ with $n$ nodes and $m$ edges, its line graph $L(G)$ has a decomposition with sparse matrices.*

$$L(G) = T(G)S(G)^T \tag{4.6}$$

*where $T(G)$ and $S(G)$ are the target and the source incident matrices, respectively.*

*Proof.* The $(i,j)^{\text{th}}$ element $L(G)_{ij}$ of $L(G)$ is nonzero and have the value $w_i w_j$ if and only if there exists two consecutive edges $e_i = (u_i, v_i)$, and $e_j = (v_i, v_j)$ in $G$ with weights $w_i$ and $w_j$, respectively. On the right side of the equation, $(i,j)$th element is computed by $t_i^T s_j$ where $t_i$ is the $i$th row of $T$, and $s_j$ is the $j$th row of $S$. By the definition of the incidence matrix, it follows that $t_i^T s_j = w_i w_j$. □

The stationary probability of a random walk on the line graph $L(G)$ can be computed by the so called power

method, which repeatedly multiplies $L(G)$ with a random initial vector. Thanks to the decomposition (4.6), we can multiply $L(G)$ with a vector $\mathbf{v}$ by first multiplying $S(G)^T$ with $\mathbf{v}$, then multiplying $T(G)$ with the previous result. After computing the stationary probability of the random walk on the line graph, we aggregate the edge scores for each node. This can be done by right multiplying the edge score by the overall incidence matrix $B(G)$ of $G$, where $B(G) = S(G) + T(G)$. Algorithm 2 shows the complete LINERANK algorithm.

**Algorithm 2** LINERANK

**Input:** Edge $E = \{(i, j, weight)\}$ with $|E| = m$,
Damping Factor $c = 0.85$
**Output:** LINERANK vector $linerank$

1: Build incidence matrices $S(G)$ and $T(G)$ from $E$
2: // Compute normalization factors
3: $\mathbf{d_1} \leftarrow S(G)^T\mathbf{1}$;
4: $\mathbf{d_2} \leftarrow T(G)\mathbf{d_1}$;
5: $\mathbf{d} \leftarrow \mathbf{1}./\mathbf{d_2}$;
6: // Run iterative random walks on $T(G)S(G)^T$
7: $\mathbf{v} \leftarrow$ random initial vector of size $m$;
8: $\mathbf{r} \leftarrow \frac{1}{m}\mathbf{1}$; // restart prob.
9: **while** $v$ does not converge **do**
10: $\quad\mathbf{v_1} \leftarrow \mathbf{dv}$; // Hadamard product
11: $\quad\mathbf{v_2} \leftarrow S(G)^T\mathbf{v_1}$;
12: $\quad\mathbf{v_3} \leftarrow T(G)\mathbf{v_2}$;
13: $\quad\mathbf{v} \leftarrow c\mathbf{v_3} + (1-c)\mathbf{r}$; // add with the restart probability
14: **end while**
15: $linerank \leftarrow (S(G) + T(G))^T\mathbf{v}$;

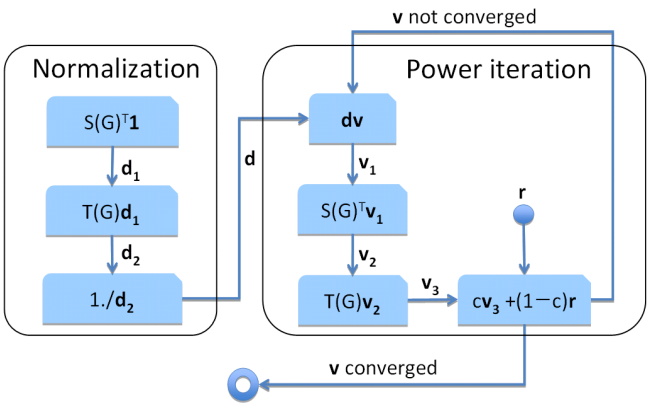



Figure 3: Dependency of different MAPREDUCE jobs for LINERANK computation. Each shaded box represents a MAPREDUCE job. Notice that the most expensive operations are matrix-vector multiplications which can be performed efficiently since the sparse matrices $S(G)$ and $T(G)$ are used instead of the dense $L(G)$ thanks to the line graph decomposition in Lemma 4.1.

We describe the above algorithm in detail and also illustrate it through a flowchart on Figure 3.

**Building Incidence Matrices.** We first construct the incident matrices $S(G)$ and $T(G)$ from the sparse adjacency matrix $E$. These matrices can be built in $O(m)$ time by reading edges and emitting the corresponding outputs.

**Computing Normalization Factors.** The $i^{\text{th}}$ element of the diagonal matrix $D$ contains the sum of $i^{\text{th}}$ column of $L(G)$. $D$ is used to column-normalize $L(G)$ so that the resulting matrix can be used for the power iteration. The './' in line 5 represents the element-wise inverse operation.

**Random Walk on the Line Graph.** From line 7 to 14, the random walk on the decomposed line graph is performed. Notice that all the operations are either matrix-vector multiplication, vector addition, or vector Hadamard products (line 10), all of which are not expensive. Also, notice that the matrices $S(G)$ and $T(G)$ contain only $m$ nonzero elements for each, which is typically much smaller than the $L(G)$ if explicitly constructed.

**Final LINERANK Score.** The edge scores are summed up in line 15 to get the final LINERANK score for each node.

Note the most expensive operation in Algorithm 2 is matrix-vector multiplication which can be performed efficiently in HADOOP [19].

**4.3 Analysis** We analyze the time and the space complexity of LINERANK. The main result is that thanks to our line graph decomposition(lemma (4.1)), LINERANK has the same complexity as random walks on the original graph, although the line graph is much bigger than the original graph.

LEMMA 4.2. (**TIME COMPLEXITY OF LINERANK**) LINERANK *takes $O(km)$ time where $k$ is the number of iterations and $m$ is the number of edges in the original graph.*

*Proof.* The time complexity is dominated by the while loop from line 9 to 14. Inside the while loop, the most expensive operations are the matrix-vector multiplications which take $O(m)$ time since the number of nonzero elements in $S(G)$ or $T(G)$ is $m$. □

The number $k$ of iterations depends on the ratio of the absolute values of the top two largest eigenvalues of the line graph. An advantage of LINERANK is that one can stop the computation after a few iterations to get reasonable accuracy, while the other betweenness centralities can not be stopped in an any-time fashion. A similar results holds for space complexity: LINERANK requires the same space as random walks on the original graph.

LEMMA 4.3. (**SPACE COMPLEXITY OF LINERANK**) LINERANK *requires $O(m)$ space.*

*Proof.* The space complexity is dominated by the incidence matrices $S(G)$ and $T(G)$ which have $m$ elements each. □

## 5 Experiments

We present our experimental evaluation, which has a two-fold goal. The first goal is to demonstrate the *efficiency and scalability* of our proposed solutions, by focusing on the following two questions:

**Q1** How fast are our proposed large-scale centralities, compared to the "standard" centralities?
**Q2** How do our algorithms scale with the graph size, as well as with the number of machines?

The second goal is to study the *effectiveness* of our approach on real graphs. More specifically, we focus on the following two questions:

**Q3** How well does the effective closeness approximate standard closeness?
**Q4** What are the patterns of centralities in real networks? Are there correlations between centralities? Are there outliers?

After summarizing the datasets used in the experiments, the rest of this section first addresses questions **(Q1–2)**, and then **(Q3)**. **(Q4)** is answered in Section 6.

**5.1 Datasets and setup** The graphs used in our experiments along with their main characteristics are summarized in Table 2. [2] We use both real-world and synthetic datasets.

The YahooWeb graph contains the links between web hosts. The weight of an edge is the number of web pages between the hosts. The Enron data contain the email exchanges of Enron employees, where the weight is the number of emails between the two people. AS-Oregon contains the router connection information. DBLP Authors contains co-author relationships among prolific authors; according to DBLP, authors are "prolific" if they have published at least than 50 papers. The weight of an edge is the number of papers co-authored by the incident authors. Note that the degree does not necessarily correspond to the total number of papers authored, since the dataset represents the induced subgraph among prolific authors only. We chose this version of the DBLP dataset to facilitate experiments and comparisons with "standard" measures of centrality.

Scalability experiments are performed on synthetic datasets, since this allows flexibility in choosing graphs of any size. We used a generator based on Kronecker multiplication [24], which produces realistic graphs.

[2] YahooWeb: released under NDA.
Kronecker: http://www.cs.cmu.edu/∼ukang/dataset
Enron: http://www.cs.cmu.edu/∼enron
AS-Oregon: http://topology.eecs.umich.edu/data.html
DBLP: http://www.informatik.uni-trier.de/∼ley/db/indices/a-tree/prolific/index.html, also in http://www.cs.cmu.edu/∼ukang/dataset

| Name | Nodes | Edges | Description |
|---|---|---|---|
| YahooWeb | 24 M | 1 B | WWW links between hosts |
| Kronecker | 177 K<br>120 K<br>59 K<br>19 K | 1,977 M<br>1,145 M<br>282 M<br>40 M | synthetic |
| Enron | 80 K | 575 K | Email |
| AS-Oregon | 13 K | 74 K | Router connections |
| DBLP Authors | 3 K | 22 K | DBLP prolific authors |

Table 2: Summary of datasets and main characteristics.

We implemented our scalable algorithms in Java using HADOOP version 0.20.1. Large-scale experiments were run on the Yahoo! M45 cluster, using 10 to 70 machines. For standard measures of centrality on small graphs, we used the iGraph package for R on a single machine.

**5.2 Scalability and efficiency** Figure 4 shows the results from experiments on efficiency and scalability.

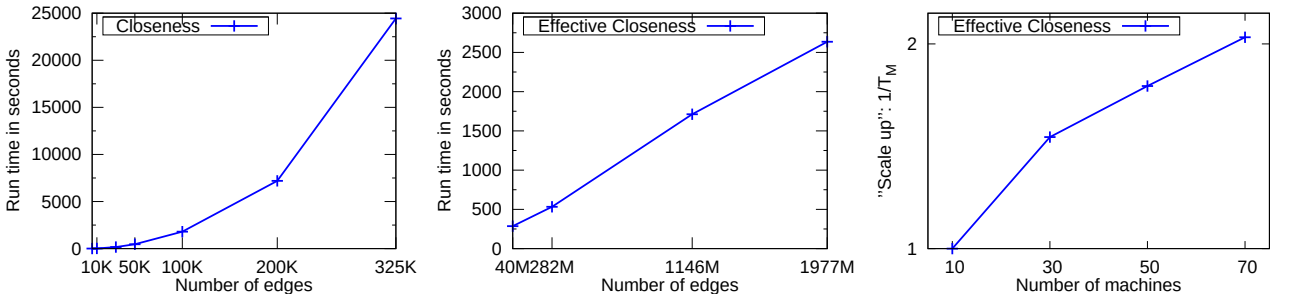
Figures 4(a,d) show the running time for "standard" centrality measures (closeness and shortest-path betweenness) on a single machine. The running time clearly grows super-linearly with respect to the number of edges.

Figures 4(b,c,e,f) show the running time of our distributed algorithms for our proposed centralities. For each of the two centrality measures (effective closeness and LINERANK) we vary both the number of machines, as well as the size of the graphs.

Both effective closeness and LINERANK show linear scale-up with respect to the number of edges, in figures 4(b,e). For this set of experiments, the number of machines was fixed at 50.

Scale-up is also near-linear with respect to the number of machines. Figures 4(c,f) shows the scale-up $1/T_M$ where $T_M$ is the running time with $M$ machines. The scale-up score is normalized so that it is 1 when $M$=10. Both of the algorithms scale near-linearly with respect to the number of machines. For this set of experiments, the size of the graph was fixed to 282M edges.

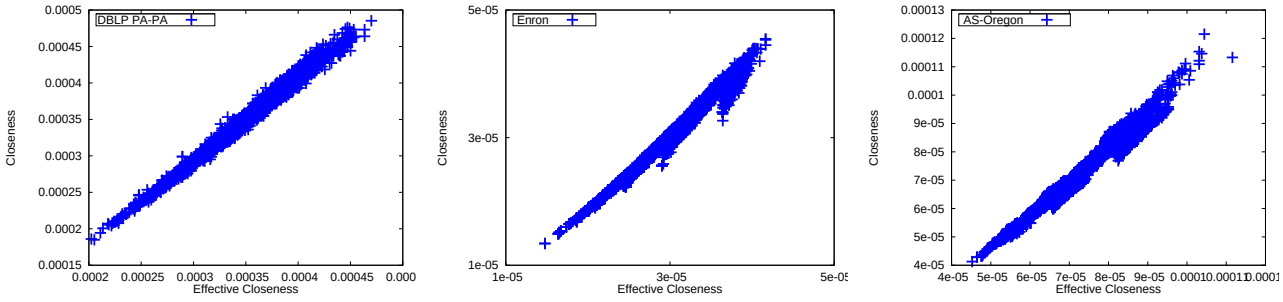
**5.3 Effective Closeness** Figure 5 shows the scatter plots of standard closeness versus our proposed effective closeness, on relatively small graphs where it is feasible to compute the former measure. Each point in the scatter plot corresponds to a node in the graph. Across all datasets there exist clear linear correlations between the two measures with correlation coefficient at least 0.978. Therefore, effective closeness is a good substitute for standard closeness. More importantly, effective closeness can also be used in billion-scale




(a) Closeness: time vs. edges (b) Effective Closeness: time vs. edges (c) Effective Closeness: time vs. machines

(d) Betweenness: time vs. edges (e) LINERANK: time vs. edges (f) LINERANK: time vs. machines

Figure 4: **(a, d)**: Scalability of the standard closeness and betweenness. Notice that running time grows super-linearly with respect to the number of edges, and is quite long for a relatively small graph with 325 K nodes. **(b, c)**: Scalability of effective closeness. The running time is for one iteration on the Kronecker datasets. The experiment (c) is performed on a graph with 282 million edges. Scale-up is linear with respect to the number of edges, and near-linear with respect to the number of machines. **(e, f)**: Scalability of LINERANK. The running time is for one power iteration on the line graphs of the Kronecker graphs. The experiment (f) is performed on a graph with 282 million edges. As with the effective closeness, the scale-up is linear both with respect to the number of edges and the number of machines.




(a) DBLP Authors
Corr. Coefficient: 0.994

(b) Enron
Corr. Coefficient: 0.992

(c) AS-Oregon
Corr. Coefficient: 0.978

Figure 5: The correlations between our proposed effective closeness and the standard closeness on small, real-world graphs. Each of the score is normalized to sum to 1. The two measures are near-linearly correlated with correlation coefficients at least 0.978, but our solution is much faster to compute.

graphs, while standard closeness is limited to small graphs.

## 6 Patterns of Centralities in Large Networks

In this section, we present patterns of centralities in real world, large scale networks. Figure 6 shows the relationships between degree (baseline measure) against effective closeness as well as LINERANK. Each point in the scatter plots corresponds to a node in the graph. We have the following observations for our proposed measures.

**6.1 Effective Closeness and Degree** In Figures 6 (a), (c), and (e), we observe high degree nodes have high effective closeness. This reflects that high degree nodes have higher chances to reach all nodes within short number of steps, due to its many connections.

OBSERVATION 1. (EFF. CLO. FOR HIGH DEGREE NODES) *High degree nodes have high effective closeness, reflecting their higher chances to reach other nodes quickly.*

In contrast to high degree nodes, low-degree nodes have various effective closeness values. This implies that nodes that are hard to be differentiated by degree measure now can be easily separated by our effective closeness measure. The reason is that, if a node $v$ has an effective closeness $f$, then a neighboring node of $v$ will also have an effective closeness similar to $f$. Thus, two nodes with the same degree can have different effective closeness based on which nodes they connect to. For example, in the DBLP prolific authors dataset, both Foto N. Afrati and Massimo Pocino have degree 5 (the degree here is the number of prolific co-authors). However, despite having the same degree, Foto N. Afrati has 1.6 times larger effective closeness than Massimo Pocino, since she has co-authored a paper with Jeffrey D. Ullman who has the highest effective closeness. Similarly, in the Enron dataset, Kenneth Lay, the CEO of Enron, has high effective closeness. fei.yan@enron.com has degree 1 but $1.81\times$ higher effective closeness than swann@enron.com with the same degree, since fei.yan@enron.com has exchanged email with the CEO. Finally, in the YahooWeb dataset, the site www.robertsonbonded.com has degree 1 but has high effective closeness $4.4 \times 10^{-8}$ which is more than 4 times larger than the effective closeness of some pages with the same degree. The reason is that www.robertsonbonded.com is pointed by dmoz.org which has very high effective closeness. Thus, we conclude that the effective closeness gives additional useful information not conveyed by the degree.

OBSERVATION 2. (EFF. CLO. FOR LOW DEGREE NODES) *Low degree nodes have varying effective closeness based on the closeness of their neighbors. For this reason, effective closeness can be used to distinguish low degree nodes.*

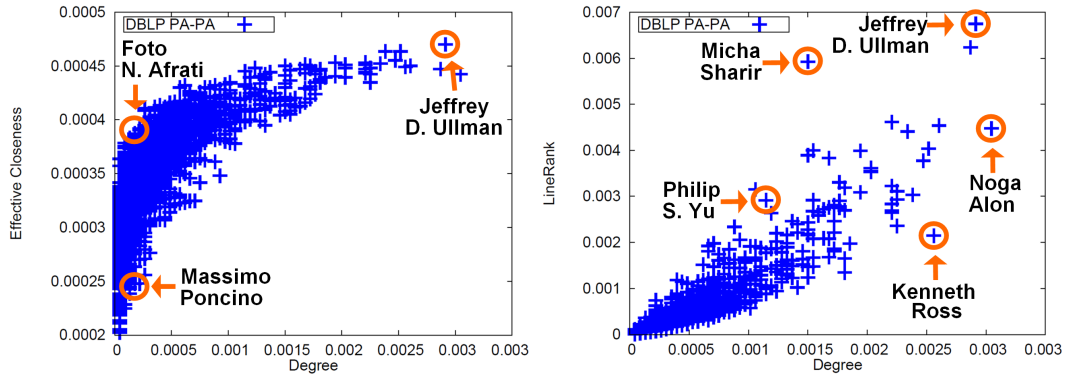
**6.2 LINERANK and Degree** The effective closeness gives another dimension of information which can be used to differentiate nodes further than possible by degree alone. However, nodes with high degree tend to have high effective closeness, and thus can not be distinguished by the effective closeness. LINERANK can be used to distinguish high degree nodes. In contrast to the degree which considers only one-step neighbors, LINERANK considers also the quality of the connections of a node's neighbors where the quality is acquired by stationary probabilities in random walks over the whole graph. Thus, some nodes have high degree but have relatively low LINERANK due to the quality of the edges. For example, Noga Alon has the highest degree, which is the number of co-authors, in the DBLP prolific authors dataset, but his LINERANK is smaller than Jeffrey D. Ullman since Noga Alon co-authored 125 papers which is smaller than 199 papers that Jeffrey D. Ullman co-authored with other prolific authors. On the other hand, some authors have high LINERANK compared to the degree. For example, Philip S. Yu has 26 prolific co-authors but published 147 papers with them, thus has higher LINERANK than Kenneth A. Ross who has 58 prolific co-authors but published 66 papers with them. The same applies to Micha Sharir who has 34 prolific co-authors but 223 papers co-authored, and thus has higher LINERANK. Similarly, in the Enron data, the CEO Kenneth Lay has the highest degree, but his LINERANK is smaller than Jeff Dasovich, the governmental affairs executive, since Jeff exchanged about $10\times$ more email than the CEO, probably due to his role. In the YahooWeb data, the top 3 highest degree hosts(www7.calle.com, dmoz.org, and www.dmoz.org), are different from the top 3 highest LINERANK hosts(geocities.yahoohost.com, www.angelfire.com, and members.aol.com). Again, the reason for this difference is the strength of the connections: the top 3 highest LINERANK hosts have more total neighboring *pages* than the top 3 highest degree hosts. We conclude that LINERANK gives yet additional useful information for distinguishing high degree nodes.

OBSERVATION 3. (LINERANK FOR HIGH DEGREE NODES) *High degree nodes have varying* LINERANK *based on the strength of the incident edges. Thus,* LINERANK *can be used to distinguish high degree nodes.*

## 7 Conclusion

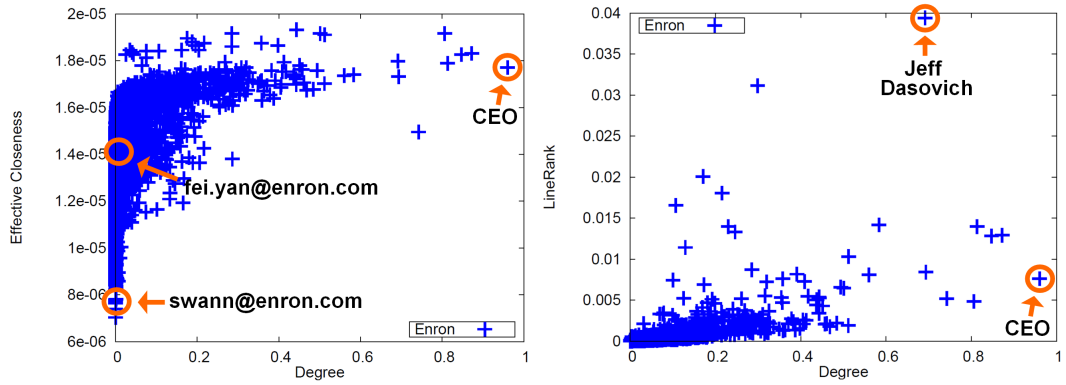
In this paper we address challenges in computing informative measures of centrality on billion scale graphs. The main contributions are the following:

1. **Careful Design.** We propose *effective closeness*, a diameter-based centrality, and LINERANK, a flow-based centrality, both of which are by design suitable for large-scale, distributed processing platforms.
2. **Algorithms.** We develop the scalable and effective algorithms for MAPREDUCE by using approximation and efficient line graph decomposition. We perform experi-



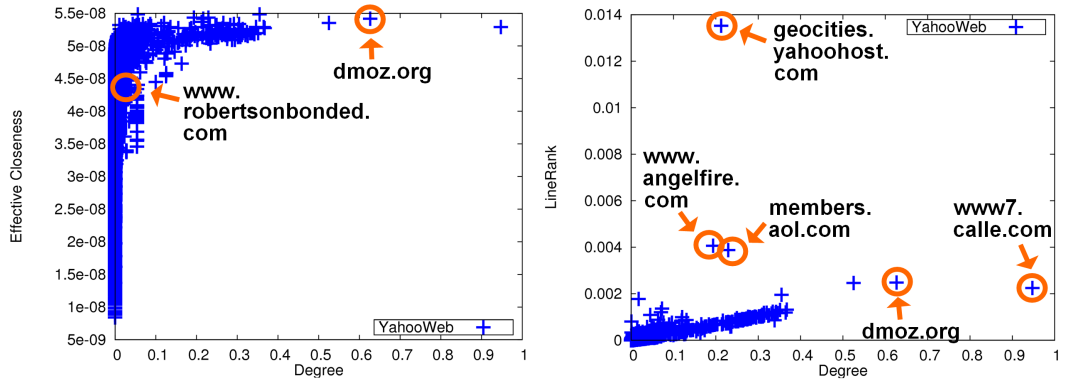

(a) DBLP Authors: Effective Closeness vs. Degree

(b) DBLP Authors: LINERANK vs. Degree




(c) Enron: Effective Closeness vs. Degree

(d) Enron: LINERANK vs. Degree




(e) YahooWeb: Effective Closeness vs. Degree

(f) YahooWeb: LINERANK vs. Degree

Figure 6: **[Best Viewed In Color]** The scatter plots from all pairs of large scale centrality measures. The effective closeness and the degree centralities are normalized. Notice high degree nodes have high effective closeness since they can reach other nodes within small number of steps due to many neighbors. However, low degree nodes have varying effective closeness. Thus, effective closeness can be used to distinguish nodes with low degrees. High degree nodes can be distinguished by LINERANK.

ments on large datasets with HADOOP, and demonstrate the effectiveness as well as the scalability of our method for billion-scale graphs.

3. **Observations.** We show how our proposed measures can reveal interesting correlations and anomalies of real-world graphs. We report that nodes with high effective closeness have high degree. Furthermore, we show that the effective closeness and LINERANK can be used for discriminating low degree nodes and high degree nodes, respectively.

Researches on social network analysis and computational social science [23] can benefit significantly from our proposed large scale centralities and efficient algorithms. Future research direction includes extending the current algorithms to time-evolving networks.


### Acknowledgments

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.


### References


[1] Hadoop information. http://hadoop.apache.org/.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments, 1996.

[3] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. SIGMOD, 2007.

[4] P. Bonacich. Power and centrality: a family of measures. *American Journal of Sociology*, 92:1170–1182, 1987.

[5] S. P. Borgatti and M. G. Everett. A graph-theoretic perspective on centrality. *Social Networks*, 2006.

[6] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 2001.

[7] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: easy and efficient parallel processing of massive data sets. *VLDB*, 2008.

[8] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. PODS, 2000.

[9] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

[11] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 1985.

[12] L. Freeman. Centrality in networks: I. conceptual clarification. *Social Networks*, 1979.

[13] M. N. Garofalakis and P. B. Gibbon. Approximate query processing: Taming the terabytes. *VLDB*, 2001.

[14] R. L. Grossman and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. *KDD*, 2008.

[15] C. Hubbell. An input output approach to clique identification. *Sociometry*, 28:377–399, 1965.

[16] H. Jeong, S. P. Mason, A.-L. Barabasi, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, (411):41–42, 2001.

[17] U. Kang, D. H. Chau, and C. Faloutsos. Mining large graphs: Algorithms, inference, and discoveries. *IEEE International Conference on Data Engineering*, 2011.

[18] U. Kang, C. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. *SIAM International Conference on Data Mining*, 2010.

[19] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *IEEE International Conference on Data Mining*, 2009.

[20] L. Katz. A new index derived from sociometric data analysis. *Psychometrika*, 18:39–43, 1953.

[21] V. Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[22] R. Lämmel. Google's mapreduce programming model – revisited. *Science of Computer Programming*, 70:1–30, 2008.

[23] D. Lazer, A. Pentland, L. Adamic, S. Aral, A.-L. Barabasi, D. Brewer, N. Christakis, N. Contractor, J. Fowler, M. Gutmann, T. Jebara, G. King, M. Macy, D. Roy, and M. V. Alstyne. Computational social science. *Science*, (323):721–723, 2009.

[24] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Generation and evolution, using kronecker multiplication. In *PKDD*, 2005.

[25] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 2005.

[26] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD '08*, pages 1099–1110, 2008.

[27] S. Papadimitriou and J. Sun. Disco: Distributed co-clustering with map-reduce. *ICDM*, 2008.

[28] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming Journal*, 2005.