A Time-Efficient Connected Densest Subgraph Discovery Algorithm for Big Data

Bo Wu and Haiying Shen
Department of Electrical and Computer Engineering
Clemson University, Clemson, South Carolina 29634
{bwu2, shenh}@clemson.edu

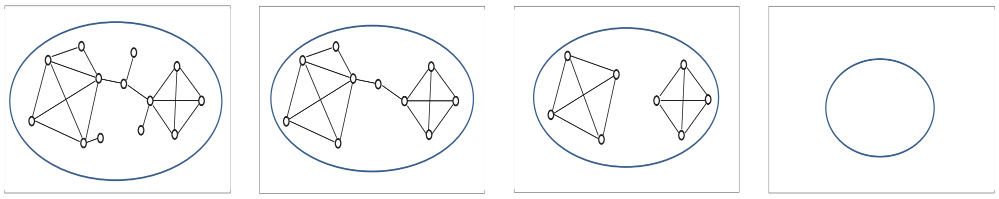
***Abstract*—In this paper, we propose a time-efficient and exact algorithm for the problem of discovering the densest subgraph in big data. Current algorithms for solving this problem have three problems: i) they cannot handle the dilemma between the efficiency of handing big data and the precision of the discovered densest subgraph; ii) they cannot take advantage of both the parallel computing on MapReduce and in-memory computing on one computer; iii) their applicability to different kinds of graphs has not been discussed. Our proposed algorithm combines the MapReduce parallel computing with in-memory computing on one computer together to improve the efficiency and precision of discovering the densest subgraphs. The algorithm consists of two computational phases: i) the graph reduction in the MapReduce framework; ii) the densest subgraph discovery in memory. Further, we theoretically analyze the correctness of this algorithm and its applicability in different natural graphs. We conduct extensive experimental evaluations in a MapReduce framework on both massive real-world graphs and simulated graphs to test our algorithm in comparison with other algorithms. Experimental results show that our algorithm is more time-efficient and precise than other algorithms.**

I. INTRODUCTION

In this paper, we focus on designing a new exact densest subgraph discovery algorithm for big data. The densest subgraph in a graph is the subgraph in which the vertices have the highest average degree. The concept of the densest subgraph is widely applied in defining and identifying communities, which is a challenging problem in the graph analysis. The most well-known definitions of the community include module [1], clique [2], dense subgraph [3], and so on. When applying these community definitions to real applications, the module performs well when the dataset is small [1], but it is intractable when it meets large datasets [4]; the clique is a strict concept (each pair of the vertices must be connected), and is rarely applied to real applications; while the densest subgraph performs well in various applications [5–7], and is much easier to implement when datasets become large. Therefore, discovering the densest subgraph plays a significant role in many applications [5–7], especially considering the popular graph structure in this era of big data (e.g., WWW and social networks). Exact [3, 8], approximate [9, 10] and heuristic algorithms [6, 7, 11] have been proposed for solving the densest subgraph problem.

However, current algorithms for solving the densest subgraph problem have three problems. First, they cannot handle the dilemma between the efficiency of handing big data and the precision of the discovered densest subgraph. Exact algorithms [3, 8] can find the exact (i.e., precise) densest subgraph but are very time-consuming. The approximate algorithms [9, 10] improve the time-efficiency but reach a 2-approximation of the density of the discovered densest subgraph, which is still far from the exact result. Their discovered densest subgraph may contain disconnected components, which leads to a low precision result since the densest subgraph should be connected in the applications. Figure 1 shows an example of the connectivity problem of current approximate algorithms. Figure 1(a) shows the initial graph. Then, in each step, several vertices are deleted greedily until all the vertices are deleted (as shown in Figure 1(d)). During this process, the algorithm pick the subgraph with the highest density (as shown in Figure 1(c)) as the outcome. However, the outcome consists of two isolated components.

Second, the current algorithms cannot take advantage of both the parallel computing on MapReduce and in-memory computing on one computer. MapReduce is good at handling big data due to its parallel computing capacity. Using the MapReduce to discover the densest subgraph, few rounds (i.e., one time of Map and Reduce process) of MapReduce process generates insufficiently precise result [10] while many rounds to produce a more precise result lead to low time-efficiency due to data transmission between computers. On the other hand, using the in-memory computing on one computer, the exact densest subgraph discovery algorithms [3, 8] are time consuming to handle large datasets and lack the capability to handle large graphs which cannot be fitted in memory, especially considering that datasets become increasingly larger in this era of big data. Therefore, it is a challenge to take advantage of both MapReduce and in-memory computing on one computer to calculate the exact connected densest subgraph in a time-efficient manner for big data. Third, the applicability of current algorithms to different kinds of graphs has not been discussed. Different kinds of natural graphs have different structure features. For example, some of the natural graphs [12] have community structures and some others do not have community structures [13]. Actually, the densest subgraph discovery is only meaningful when the natural graphs have community structure. For the graphs without community structure, the densities of different subgraphs are close to each other and the densest subgraph loses its representativeness. However, current works on exact and approximate algorithm-



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

Fig. 1. An example of the process of current approximate algorithms

s [3, 9, 10] fail to discuss the applicability of their algorithms on different kinds of natural graphs.

In this paper, we design a time-efficient algorithm for discovering the densest connected subgraph for undirected massive graphs in a MapReduce framework. Our algorithm has two phases. In the first phase, it carefully reduces the dataset in a MapReduce framework without loss of any nodes existing in the exact densest subgraph. In the second phase, the reduced dataset can be handled in-memory in one computer by an exact densest subgraph discovery algorithm [3]. Therefore, we can find the exact densest subgraph for big data by taking advantage of both MapReduce and in-memory computing on one computer. As a result, the first and second problems mentioned above can be resolved. Furthermore, we theoretically prove the correctness of our algorithm and analyze its applicability on different complex network models [13, 14]. Finally, we conduct extensive experimental evaluations in a MapReduce framework on both massive real-world graphs and simulated graphs to test our algorithm in comparison with other algorithms. Experimental results show that our algorithm is capable of discovering the connected densest subgraph. Also, it is more time-efficient and precise than other algorithms.

In summary, our paper has the following contributions:

1) This work is the first that proposes and handles the problem of *discovering connected densest subgraph*. Current densest subgraph detection algorithms cannot guarantee the connectivity of the detected densest subgraph.
2) We provide a time-efficient exact algorithm to discover the connected densest subgraph for large datasets by taking advantage of both MapReduce and in-memory computing on one computer. The graph reduction phase deletes vertices with degrees below a certain threshold recursively. Our algorithm is more efficient since it reduces two rounds in one recursion in the current algorithm [10] to one round.
3) We theoretically analyze the correctness of this algorithm and its applicability in different natural graphs with and without community structures.
4) We conduct extensive experiments on massive real world and simulated natural graphs. Experimental results show the effectiveness and efficiency of our algorithms on natural graphs with community structures in comparison with current algorithms.

The rest of this paper is organized as follows. Section II describes the design of our densest subgraph discovery algorithm. Section III proves the validity of the algorithm. Section IV presents extensive experimental evaluation on massive real-world graphs and simulated graphs. Section V presents a concise review of the related work. Section VI summarizes this paper with remarks on our future work.

II. Time-Efficient Connected Densest Subgraph Discovery Algorithm

Before we present the details of the time-efficient connected densest subgraph discovery algorithm, let us first introduce the concept of the density of an undirected graph. Let $G = (V, E)$ be an undirected graph. For a subset $S \subseteq V$, the *induced edge set* is defined as $E(S) = E \cap S^2$ and the *induced degree* of a node $v_i \in S$ is defined as $\deg_S(v_i) = |\{v_j|(v_i, v_j) \in E(S)\}|$, where $|\cdot|$ means the number of elements in the set.

Density of an undirected graph [10] Let $G = (V, E)$ be an undirected graph. Given $S \subseteq V$, its density $\rho(S)$ is defined as

$$\rho(S) = \frac{|E(S)|}{|S|}.$$

The maximum density $\rho^*(S)$ of the graph is

$$\rho^*(S) = \max_{S \subseteq V}\{\rho(S)\}.$$

Since our algorithm takes advantage of both the parallel computing on MapReduce and in-memory computing on One computer, we call our algorithm *M-O* algorithm for short in the rest of the paper.

A. The Design of the Algorithm

The natural graphs usually follow a power-law degree distribution [13]. Intuitively, for graphs with such a feature, most of the vertices with low degrees have very small probabilities to be in the densest subgraph. Therefore, the basic idea of the M-O algorithm is trying to reduce the initial size of the dataset by deleting the vertices with very low degrees in order to fit the reduced dataset in the memory of one computer. Then, the algorithm applies the min-cut max-flow technique [3] to find the densest subgraph in the remaining graph on one computer. With only one round of the MapReduce process and in-memory computing on one computer without the data transfer between computers, the M-O algorithm achieves high time-efficiency.

Based on this intuition, the M-O algorithm has two phases: 1) the graph reduction phase, and

Algorithm 1: The pseudocode of the M-O algorithm.

```
1: Given: $G = (V, E)$;
2: $S \leftarrow V$, $\rho_{max} \leftarrow \rho(S)$;
3: while $S > threshold$ do
       $S_c \leftarrow \{v_i \in S | deg_S(v_i) \leq \rho_{max}\}$;
       $S \leftarrow S \backslash S_c$;
       if $\rho(S) > \rho_{max}$ then
           $\rho_{max} \leftarrow \rho(S)$;
       end
   end
4: $G_0 = (S_0, E(S_0)) \leftarrow G_S = (S, E(S))$;
5: Given: $G = (V, E)$;
6: $l \leftarrow 0, u \leftarrow n$;
7: while $(l - u) \geq \frac{1}{n(n-1)}$ do
       $g \leftarrow \frac{l+u}{2}$;
       Construct $N = (V_N, E(V_N))$;
       Find min-cut $(S, T)$;
       if $S = \{s\}$ then
           $u \leftarrow g$
       end
       if $S \neq \{s\}$ then
           $l \leftarrow g$;
           $V_1 \leftarrow S - \{s\}$;
       end
   end
8: return subgraph of $G$ derived by $c(S, T)$;
```

2) densest subgraph discovery phase. Figure 2 illustrates the flowchart of the M-O algorithm. Algorithm 1 presents the pseudocode of the M-O algorithm. For a given $G = (V, E)$, we first delete all the vertices, which have degrees equal or smaller than the maximum density of remaining graph during the deleting process streamingly (blocks 1-3). After the size of the datasets is reduced to a suitable size (i.e., threshold in Figure 2) that can fit into the memory of a computer for the densest subgraph discovery phase, we apply min-cut max-flow technique to find the densest subgraph in the remaining graph (blocks 5-7). Block 8 returns the results. In Section III, we prove the correctness and applicability of the M-O algorithm.

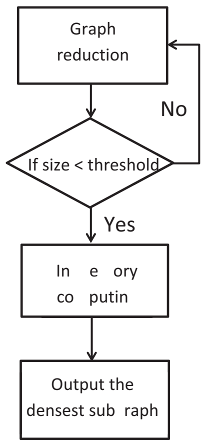


Fig. 2. Flowchart of the M-O algorithm

Next, we introduce the min-cut max-flow based exact densest subgraph discovering algorithm. The edges in the initial graph are assigned with capacity 1. A min-cut of a graph is a cut (a partition of the vertices of a graph into two disjoint subsets) whose cut edge set (consisting of edges that cross the two disjoint subsets) has the smallest sum of capacities. We use $c(S, T)$ to denote the min-cut of a graph that splits the graph to two partitions, S and T. The capacity of min-cut $c(S, T)$ is the sum of the capacities of the cut edge set. Then, the densest subgraph can be found by recursively constructing a new graph based on the current graph and finding the min-cut $c(S, T)$ on the new graph. In each iteration of the recursion, we first construct a new graph by adding two vertices, s and t. We build an edge between each vertex v_i and s and assign each edge with capacity e, where e is the number of edges in the initial graph. We also build an edge between each vertex v_i and t and assign each edge with capacity $e+2g-d_{v_i}$ where g is an estimated value of the density of the densest subgraph and d_{v_i} is the degree of v_i in the initial graph. Second, we update the upper bound p of g by the current value of g if $S = \{s\}$ and update the lower bound l of g by the current value of g if $T = \{t\}$. Third, we assign g with value $\frac{p+l}{2}$ and start the next iteration. The recursion is stopped when $(l-p) < \frac{1}{n(n-1)}$ where n is the number of vertices in the new graph. Then, the partition $S - \{s\}$ is the densest subgraph. For more details of the min-cut max-flow technique, please refer to [3].

B. Implementation on MapReduce

In the graph reduction phase, we delete a batch of vertices every time, which can be implemented in parallel computing frameworks such as MapReduce [15]. MapReduce is running based on the data structure in $< key; value >$ pairs. Current densest subgraph algorithm [10] takes two rounds of MapReduce processes for deleting the vertices with degrees smaller than a certain value. In the first round, it tags all the vertices that need to be deleted. Then, in the second round, it deletes all the tagged vertices. Our algorithm is more time-efficient in that it can delete the vertices with degrees smaller than a certain value in one round of MapReduce process as shown in Algorithm 2.

Algorithm 2: MapReduce implementation of the graph reduction

```
1: Mapper
       Input: $< v; u >$ and (or) $< v; \widetilde{u} >$
       emit Input;
   end
   Reducer
       Input: $< v; neighborlist >$
       clean the neighbor list;
       calculate the degree of $v$;
       if $degree(v) > threshold$ then
           foreach $v$ in $neighborlist$ do
               emit $< v; u_i >$;
           end
       else
           foreach $v$ in $neighborlist$ do
               emit $< u_i; \widetilde{v} >$;
           end
       end
   end
```

The input of the first round of our MapReduce program is the edge list, which records each edge connecting vertices u and v in the graph twice; that is, $< u; v >$ and $< v; u >$. The Map function outputs the neighbor list of each vertex in the form of $< v; u_1, u_2, ..., u_n >$, where v is an arbitrary vertex and $u_1, u_2, ..., u_n$ are the other ends in the neighbor list of vertex v (block 1). The Reduce function outputs edges in the form of $< v; u_1 >, < v; u_2 >, ..., < v; u_n >$ if the degree of v is larger than the threshold, which is ρ_{max} in Algorithm 1. Otherwise, it outputs edges in the form of $< u_1; \widetilde{v} >, < u_2; \widetilde{v} >, ..., < u_n; \widetilde{v} >$. $\widetilde{v}$ indicates that vertex v should be deleted later on in the neighbor list of each node u_i $(i = 1, 2, ..., n)$ (block 2).

This output of the Reduce function is the input of the next round of our MapReduce program, which consists of the edge list and the edges that should be deleted recorded as $< u; \widetilde{v} >$. In the next round, the Map function outputs the neighbor list of each vertex in the form of $< v; u_1, u_2, ..., u_n, \widetilde{u}_i >$, where $\widetilde{u}_i$ indicates that vertex u_i should be deleted (block 1). In the Reduce function, for an input of neighbor list of a vertex v, we first delete the vertices which have been tagged as $\widetilde{u}$ in last round in the neighbor list of vertex v and reproduce its neighbor list in the remaining graph. Then, we output edges in the form of $< v, u_1 >, < v, u_2 >, ..., < v, u_n >$ if the degree of v is larger than the threshold. Otherwise, we output edges in the form of $< u_1; \widetilde{v} >, < u_2; \widetilde{v} >, ..., < u_n; \widetilde{v} >$ (block 2). This process of the round repeats until the size of the data is suitable for in-memory computing.

Our MapReduce strategy only uses one MapReduce round for deleting the vertices tagged in the previous round and tagging the vertices that should be deleted in the next round at the same time. Therefore, our algorithm only uses one MapReduce round for one recursion of data reduction.

The M-O algorithm applies advanced MapReduce strategy for the graph reduction phase, so that it can further improve the time efficiency.

III. Theoretical Analysis

In this section, we analyze the correctness and applicability of the M-O algorithm. There are a number of requirements that the M-O algorithm needs to meet to verify its correctness and applicability.

1) **Correctness**
   a) **Requirement 1:** The process of the graph reduction cannot delete any vertices in the densest subgraph.
   b) **Requirement 2:** The final densest subgraph discovered by the M-O algorithm must be connected.
2) **Applicability**
   a) **Requirement 1:** The graph size must be reduced to a suitable size that can fit in the memory of one computer for in-memory computing.
   b) **Requirement 2:** The number of rounds of the graph reduction should be as small as possible to achieve high time-efficient.

Furthermore, the performance of the M-O algorithm is dependent on the graph topology. Therefore, we further discuss the applicability of the M-O algorithm to different kinds of natural graphs.

A. Correctness of the Reduction

First, we prove that the strategy of the graph reduction of the M-O algorithm does not delete any vertices in the densest subgraph.

Lemma 3.1: Suppose $G_S = (V_S, E(V_S))$ is the densest subgraph of graph G, then we have $deg_{V_S}(v_i) \geq \rho(V_S)$ for any vertex $v_i \in V_S$.

Proof: We prove it by contradiction. Suppose there is at least one vertex $v_i \in V_S$ and $deg_{V_S}(v_i) < \rho(V_S)$, then we delete vertex v_i from G_S, and get graph $G_{S^-} = (V_S \backslash \{v_i\}, E(V_S \backslash v_i))$. The density of graph G_{S^-} is:

$$\begin{aligned} \rho(V_S \backslash \{v_i\}) &= \frac{|E(V_S \backslash v_i)|}{|V_S \backslash \{v_i\}|} \\ &= \frac{\rho(V_S)|V_S| - deg_{V_S}(v_i)}{|V_S| - 1} \end{aligned}$$

Since $deg_{V_S}(v_i) < \rho(V_S)$, we have:

$$\rho(V_S \backslash \{v_i\}) > \frac{\rho(V_S)(|V_S| - 1)}{|V_S| - 1} > \rho(V_S)$$

This contradicts with the precondition. Therefore, we have $deg_{V_S}(v_i) \geq \rho(V_S)$ for any vertex $v_i \in V_S$. ■

Theorem 3.1: After we delete all the vertices with degrees no more than the maximum density of the remaining graph of G in block 3 of Algorithm 1 to obtain G_S, the densest subgraph of G is in G_S.

Proof: We prove it by contradiction. Suppose there is one densest subgraph $G_x = (V_x, E(V_x))$, where $G_x \not\subset G_S$, then there is a vertex subset I where $I \subset V_x$ and $I \not\subset V_s$. There is a time in the deleting process that the first vertex v_i in I is deleted from the current V_s (denoted by ${V_s}^+$). Therefore, we have:

$$\rho(V_x) \leq deg_{V_x}(v_i) \leq deg_{{V_s}^+}(v_i) < \rho({V_s}^+)$$

This implies that $\rho(V_x) < \rho({V_s}^+)$, and at this moment, we have $V_x \subset {V_s}^+$. Hence, from the definition of the densest subgraph, we know that G_x is not a densest subgraph of G. This contradicts with the assumption. Therefore, for any densest subgraph G_x of G, we have $V_x \subset V_S$. ■

Theorem 3.1 guarantees that the graph reduction in the M-O algorithm cannot delete any vertices in the densest subgraph.

B. Connectivity of the Solution

Theorem 3.2: Suppose $G_1 = (V_1, E(V_1))$ is the densest subgraph of graph G discovered by the M-O algorithm where G could be a connected or disconnected graph, then we can get that $G_1 = (V_1, E(V_1))$ is a connected graph.

Proof: We prove it by contradiction. Suppose $G_1 = (V_1, E(V_1))$ is a disconnected graph which consists of two isolated subgraphs, $G_3 = (V_3, E(V_3))$ and $G_4 = (V_4, E(V_4))$, then we have the capacity of the min-cut from which we derived G_1 equals:,

$$\begin{aligned} c(S,T) &= \sum_{i\in S, j\in T} c_{ij} \\ &= m|V_2| + (m|V_1| + 2g|V_1| - \sum_{i\in V_1} d_i) + \sum_{i\in V_1, j\in V_2} c_{ij} \\ &= m|V| + 2|V_3|(g - \frac{\sum_{i\in V_3} d_i - \sum_{i\in V_3, j\in V_2} 1}{2|V_3|}) \\ &+2|V_3|(g - \frac{\sum_{i\in V_4} d_i - \sum_{i\in V_4, j\in V_2} 1}{2|V_4|}) \end{aligned}$$

where V_2 is the other part of the min-cut. Since g is the density of $\rho(V_1)$, suppose $\rho(V_3) \le \rho(V_4)$, then we have:

$$\begin{aligned} c(S, T\cup V_3) &= m|V| + 2|V_3|(g - \frac{\sum_{i\in V_3} d_i - \sum_{i\in V_3, j\in V_2} 1}{2|V_3|}) \\ &< c(S,T) \end{aligned}$$

Since the min-cut from which we derived G_1 should be a min-cut in the densest subgraph discovery phase of the M-O algorithm, this contradicts with the precondition. Therefore, $G_1 = (V_1, E(V_1))$ is a connected graph. ■

Theorem 3.2 guarantees that the densest subgraph discovered by the M-O algorithm is connected.

C. Applicability of the M-O Algorithm

There are two main factors that influence the time efficiency of the M-O algorithm. One is the number of rounds required for the graph reduction. The other is the remaining graph size after the graph reduction. These two factors are closely related to the topology of the initial graph. For example, the graph in Figure 3(a) cannot be reduced, while the graph in Figure 3(b) only takes one round for a great size reduction. In this section, we analyze the relationship between the performance of the M-O algorithm and the topologies of natural graphs. In order to study the relationship, we use the BA network [13] and BTER network [14] (which are two typical complex networks) for our study. The BA network has the feature of power law degree distribution but has no community structures. The BTER network not only has the feature of power law degree distribution, but also has the community structures. We theoretically analyze the performance of the M-O algorithm on the BA network and BTER network to show its performance on the complex networks with or without community structures.

Theorem 3.3: Suppose $G = (V, E(V))$ is a BA network and $G_S = (V_S, E(V_S))$ is the densest subgraph of BA network G, then we have

$$\rho(V_S) = \rho(V) = m.$$

where m is the number of edges each vertex created when the vertex first joined the network.

Proof: Suppose there is a process that we can get the densest subgraph $G_S = (V_S, E(V_S))$ by deleting vertex set

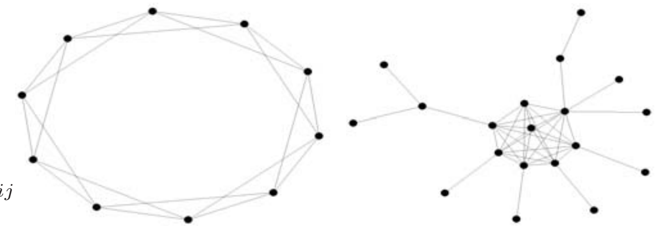

(a) Unsuitable for reduction (b) Suitable for reduction

Fig. 3. Examples of different graph topologies

$V_d = \{v_1, v_2, v_3, v_4, ..., v_n\}$, then we have:

$$\begin{aligned} \rho(V_S) &= \frac{|E(V_S)|}{|V_S|} \\ &= \frac{|E(V)| - \sum_{v_i\in V_d} \deg_d(v_i)}{|V| - |V_d|} \end{aligned}$$

where $\deg_d(v_i)$ is the degree of v_i when v_i is deleted from the remaining graph.

Since each vertex created m edges with other vertices when the vertex first joins the network, we have $\sum_{v_i\in V_d} \deg_d(v_i) \ge m|V_d|$. Therefore, we have:

$$\rho(V_S) = \frac{|E(V_S)|}{|V_S|} \le m$$

Also, when the network tends to be very large, we have:

$$\rho(V) = \frac{|E(V)|}{|V|} = \frac{m|V|}{|V|} = m$$

Therefore, we have:

$$\rho(V) = \rho(V_S) = m$$

■

Since the density of the densest subgraph is equal to the density of the whole network, there are no denser subgraphs in the network. Therefore, Theorem 3.3 indicates that BA network has no communities based on the density criteria. Then, it is meaningless to discover the densest subgraph in a BA network. At the same time, from the definition of BA network, we know that each vertex has a degree at least m since each vertex created m edges once it participated the network. Therefore, the graph reduction in the M-O algorithm is useless for reducing the size of a BA network since all the vertices have a degree equal or larger than the density of the densest subgraph. However, since it is meaningless to discover the densest subgraph on a BA network, the useless graph reduction in the M-O algorithm on BA networks does not influence the applicability of this algorithm.

Lemma 3.2: Suppose $G = (V, E(V))$ is a BTER network which approximately satisfies $\log y = \beta - \gamma \log x$, where x is the degree of the vertices with the same degree and y is the number of vertices with degree x, then we can delete $\frac{\sum_{x=1}^{\frac{\zeta(\gamma-1)}{\zeta(\gamma)}} \frac{1}{x^\gamma}}{\zeta(\gamma)}$ of the total vertices in the first round of the graph reduction. Here, γ is the exponent parameter of degree distribution of the BTER network and $\zeta(\gamma)$ is the Riemann zeta function [16] of γ.

Proof: Since $0 \leq \log y = \beta - \gamma \log x$, we have $x \leq e^{\frac{\beta}{\gamma}}$. Then, we have:

$$\begin{aligned}\rho(V) &= \frac{|E(V)|}{|V|} \\ &= \frac{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^{\beta}}{x^{\gamma}} \times x}{2\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^{\beta}}{x^{\gamma}}} = \frac{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{1}{x^{\gamma-1}}}{2\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{1}{x^{\gamma}}}\end{aligned}$$

When $e^{\frac{\beta}{\gamma}} \to +\infty$, according to the Riemann zeta function [16], we have:

$$\rho(V) = \frac{\zeta(\gamma-1)}{2\zeta(\gamma)}$$

The number of the vertices with degrees smaller than $\rho(V)$ equals $\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^{\beta}}{x^{\gamma}} \times x$. Therefore, the percent of deleted vertices in V equals:

$$\frac{\sum_{x=1}^{\frac{\zeta(\gamma-1)}{2\zeta(\gamma)}} \frac{e^{\beta}}{x^{\gamma}}}{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^{\beta}}{x^{\gamma}}} = \frac{\sum_{x=1}^{\frac{\zeta(\gamma-1)}{2\zeta(\gamma)}} \frac{1}{x^{\gamma}}}{\zeta(\gamma)}$$

■

Theorem 3.4: Suppose $G = (V, E(V))$ is a BTER network which approximately satisfies $\log y = \beta - \gamma \log x$, where x is the degree of the vertices and y is the number of vertices, then we can delete

$$\frac{\sum_{x=1}^{(\zeta(\gamma-1)-\sum_{x=1}^{\rho_{-1}(V)} x^{1-\gamma})/(\zeta(\gamma)-\sum_{x=1}^{\rho_{-1}(V)} x^{-\gamma})} x^{-\gamma}}{\zeta(\gamma) - \sum_{x=1}^{\rho_{-1}(V)} x^{-\gamma}}$$

of the total vertices in the ith round of the graph reduction where $\rho_{-1}(V)$ is the $\rho(V)$ in the $(i-1)$th round.

Proof: Since vertices with a same degree form into an isolated subgraph in a BTER network, then we can consider that the deleted vertices do not influence the degrees of the remaining vertices. Therefore, we have:

$$\begin{aligned}\rho(V) &= \frac{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^{\beta}}{x^{\gamma}} \times x - \sum_{x=1}^{\varepsilon} \frac{e^{\beta}}{x^{\gamma}} \times x}{2(\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^{\beta}}{x^{\gamma}} - \sum_{x=1}^{\varepsilon} \frac{e^{\beta}}{x^{\gamma}})} \\ &= \frac{\zeta(\gamma-1) - \sum_{x=1}^{\varepsilon} x^{1-\gamma}}{2(\zeta(\gamma) - \sum_{x=1}^{\varepsilon} x^{-\gamma})}\end{aligned}$$

where $\rho(V)$ is the current density.

Therefore, the percent of deleted vertices in V equals:

$$\frac{\sum_{x=1}^{(\zeta(\gamma-1)-\sum_{x=1}^{\varepsilon} x^{1-\gamma})/(\zeta(\gamma)-\sum_{x=1}^{\varepsilon} x^{-\gamma})} x^{-\gamma}}{\zeta(\gamma) - \sum_{x=1}^{\varepsilon} x^{-\gamma}}$$

■

In the real-world graphs, γ is usually around 2 [1]. Therefore, $\frac{\zeta(\gamma-1)}{2\zeta(\gamma)}$ is very large since $\zeta(1) = +\infty$ and $\zeta(2) \approx 1.645$. Theorem 3.4 indicates that a great percentage of vertices can be deleted in the first few rounds of the graph reduction and also the graph reduction is decreasingly efficient as the percentage of deleted vertices increases. For example, when $\gamma = 1.2$, about $\frac{1+\frac{1}{2^{2.2}}}{1.49} \approx 0.82$ of the vertices are deleted in the first round of the graph reduction. This result is also consistent with the experiment result in Section IV.

From our analysis, we have the following conclusions:

1) The complex network without community structures (e.g., BA networks) is uniformly distributed, which makes the densest subgraph discovery meaningless since the density of the densest subgraph equals the density of the whole network in such kind of complex networks.
2) For the complex network with community structures (e.g., BTER networks), the M-O algorithm not only can reduce such kind of complex networks to a suitable size that can be fitted in memory, but also can reduce it to a suitable size in only a few rounds.

IV. Performance Evaluation

In this section, we conducted comprehensive performance evaluation of the M-O algorithm and compared it with the MapReduce based approximate algorithm [10] (called ApproxMR for short). ApproxMR is a greedy algorithm for discovering the densest subgraph for undirected graphs. It greedily deletes a batch of vertices with degrees less than certain thresholds recursively and records the density of the remaining graph until the remaining graph is empty. The densest remaining graph in this deleting process is the discovered densest subgraph. ApproxMR can guarantee a 2-approximation for the density of the accurate densest subgraph, $\rho^*(S)$. ApproxMR is implemented in MapReduce for parallelism.

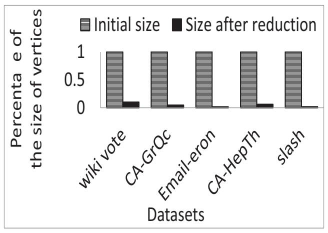
In the experiments, we first used the real-world datasets listed in Table I [17] to test the performance of the M-O algorithm in comparison with ApproxMR. In Section IV-B, we evaluated the graph reduction phase, which focuses on two aspects: i) how many percent of vertices can be reduced from the initial graph size, and ii) how many MapReduce rounds are needed to reach a suitable size for the densest subgraph discovery phase. The first aspect evaluates the effectiveness of the graph reduction phase, which determines the feasibility of the densest subgraph discovery phase. If the dataset size after reduction is still big, the M-O algorithm cannot handle it in the second phase. The second aspect evaluates the efficiency of the M-O algorithm since one round of MapReduce process is time consuming. In Section IV-C, we evaluated the density and the running time of the discovered densest subgraph. In Section IV-D, we measured the connectivity of the discovered densest subgraph. Finally, in Section IV-E, we used simulated datasets to evaluate the M-O algorithm in comparison with ApproxMR.

A. Experiment Environment and Datasets

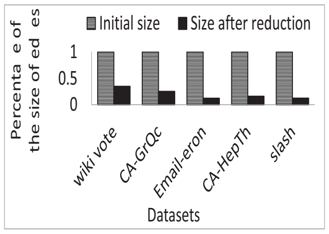
We run the M-O algorithm on Hadoop [18] with 4 PCs; each PC is equipped with 2.1GHz Intel core i3 processor with 2 cores, and a 2GB memory. The M-O algorithm was implemented in Python. We used the datasets in Table I in the experiments, which are from the SNAP library [17].

TABLE I
DESCRIPTION OF REAL-WORLD DATASETS

ID	Name	Description	$\|V\|$	$\|E\|$	Type
Dataset 1	Wiki-Vote [21]	Wikipedia who votes on whom network	7,115	207,378	small
Dataset 2	CA-GrQc [22]	Collaboration network of Arxiv General Relativity	12,008	237,010	small
Dataset 3	Email-Enron [23]	Enron company email list	36,692	367,662	small
Dataset 4	CA-HepPh [24]	Arxiv High Energy Physics paper citation network	34,546	421,578	small
Dataset 5	slash [25]	Slashdot social network from November 2008	77,360	905,468	small
Dataset 6	com-youtube [26]	Youtube online social network	1,134,890	2,987,624	large
Dataset 7	com-lj [26]	LiveJournal online social network	3,997,962	34,681,189	large
Dataset 8	com-orkut [26]	Orkut online social network	3,072,441	117,185,083	large

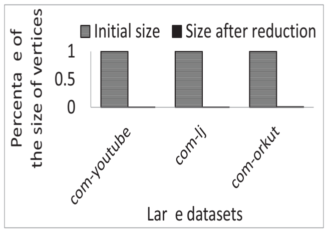


(a) The # of vertices

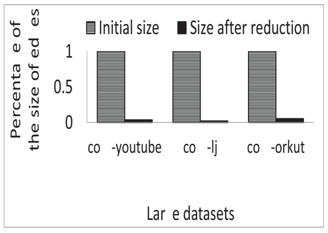


(b) The # of edges

Fig. 4. The size of the datasets before and after the reduction



(a) The # of vertices



(b) The # of edges

Fig. 5. The size of the datasets before and after the reduction for large datasets

We classified the datasets to two different types (small and large); small datasets can be fitted in the memory of one computer and large datasets are too large to be fitted in memory. We would like to see the different performances of the M-O algorithm on small datasets and large datasets. The table lists the name, the description, the number of vertices ($|V|$), the number of edges ($|E|$) and the type of each dataset. In the datasets, *Wiki-Vote* is the only directed graph. However, directed graphs have similar natural graph features as undirected graphs [19]. Therefore, we just treat *Wiki-Vote* as an undirected graph in order to enrich our datasets as the current work [20].

B. Efficiency of Graph Reduction

In this section, we evaluate the performance of the graph reduction phase from the two aforementioned aspects.

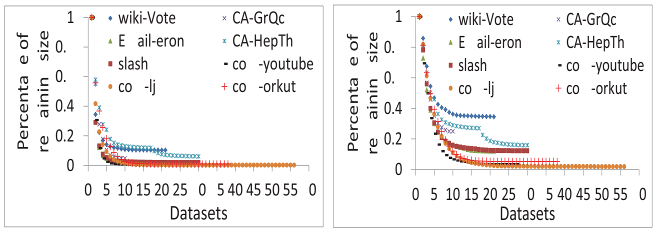
1) Comparison of sizes before and after reduction: Table II shows the comparisons of the number of vertices and the number of edges of the datasets before and after the graph reduction phase by the M-O algorithm. In order to show the reduction performances clearly, we also show Figure 4 and Figure 5, which plot the ratio of the number of vertices and the number of edges before and after the reduction for the small datasets and large datasets, respectively. From Table II, Figure 4 and Figure 5, we see that the number of vertices after reduction is less than 1% of the initial number of vertices on average. Especially, for the large datasets (e.g., *com-youtube* and *com-lj*), the number of vertices after reduction is only about 0.1% of the initial number of vertices on average. The number of edges is also reduced to an average of 18% of the initial size for the small datasets. For the large datasets, the number of edges is only about 1% of the initial size. The reduction performance is even better for the large datasets. This large size reduction makes it possible to run the densest subgraph discovery phase in one computer, since the time complexity of this phase is determined by the number of vertices and edges.

Take the large dataset *com-youtube* as an example, after the reduction, the number of vertices is 0.1% of the initial dataset and the number of edges is 3% of the initial dataset. If we use the push-relabel algorithm [27] for the densest subgraph discovery phase, which is the fastest algorithm for the min-cut max-flow problem with time complexity $O(|V||E|^2)$, then discovering the densest subgraph only takes 0.00009% ($|V||E|^2 = 0.1\% \times 3\% \times 3\%$) of the time for discovering the densest subgraph in the initial graph which can be easily calculated based on the time complexity. For the memory consumption, we only use about 3% of the memory for the initial graph since 97% ($1 - 3\% = 97\%$) of the edges have been deleted. Therefore, the large dataset reduction in the first graph reduction phase makes the dataset possible to be handled in the second densest subgraph discovery phase in the M-O algorithm.

2) The number of MapReduce rounds vs. data size: Figure 6 shows the percentage of the vertex size and the edge size of the remaining graph versus the number of MapReduce rounds in the graph reduction phase. It is interesting to see that the huge reductions only happened in the first a few rounds (5 rounds for most of the datasets). Also most of the datasets tend to be stable after 10 rounds of MapReduce processes. Figure 7 further shows the results for large datasets. After about 10 rounds of reductions, we can reduce the number of vertices to around 0.1% of the initial number of vertices and

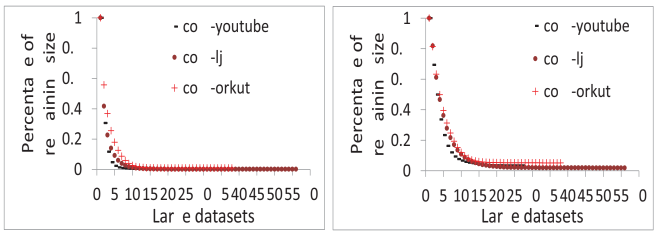
TABLE II
COMPARISON OF DATASETS BEFORE AND AFTER REDUCTION

Datasets	# of vertices			# of edges		
	Before	After	After/Before	Before	After	After/Before
wiki-vote	7,115	727	10%	207,378	71,518	34%
CA-GrQc	12,008	123	1%	237,010	4,812	2%
Email-Eron	36,692	592	1%	367,662	44,182	12%
CA-HepTh	34,546	77	0.2%	421,578	1,964	0.4%
slash	77,360	1,417	1%	905,468	98,556	10%
com-youtube	1,134,890	1,685	0.1%	2,987,624	130,062	3%
com-lj	3,997,962	4,136	0.1%	34,681,189	650,724	1%
com-orkut	3,072,441	25,776	0.8%	117,185,083	9,800,872	8%



(a) The size of reduced vertices (b) The size of reduced edges

Fig. 6. The size of reduced vertices and edges vs. the # of rounds



(a) The size of reduced vertices (b) The size of reduced edges

Fig. 7. The size of reduced vertices and edges vs. the # of rounds for large datasets

reduce the number of edges to around 1% of the initial number of edges. Although we can continue the graph reduction for more rounds for large datasets, the extra graph reduction does not help much in reducing the size of the dataset. Figure 7 indicates that the reduction performance is even better for the large datasets in which the data size can be reduced to less than 1% of the initial size in less than 10 rounds. This result indicates that the M-O algorithm is efficient in the graph reduction phase since the data can be reduced to the suitable size in only a few rounds.

C. Density and Running Time of the Discovered Densest Subgraph

In this section, we evaluate the performance of discovering the densest subgraph of the M-O algorithm in comparison with ApproxMR in terms of the density and the running time of the discovered densest subgraph. In ApproxMR, parameter ε is a slack variable for controlling the tradeoff between precision and efficiency. We set $\varepsilon = 0$ as in paper [10] to reach its best performance. Table III shows the density of the discovered densest subgraph in the M-O algorithm and ApproxMR. We

TABLE III
THE DENSITY OF THE DISCOVERED DENSEST SUBGRAPH

Datasets	M-O	ApproxMR
Wiki-Vote (small)	**49.2**	43.9
CA-GrQc (small)	**22.4**	**22.4**
Email-Eron (small)	**37.3**	35.3
CA-HepTh (small)	**15.5**	**15.5**
slash (small)	**41.9**	38.7
com-youtube (large)	**38.6**	34.4
com-lj (large)	**47.4**	35.3
com-orkut (large)	**189.1**	176.2

bold the result for the better performance in comparison in all tables in the evaluation. As shown in the table, the M-O algorithm performs better almost for all the datasets while ApproxMR only performs as well as M-O algorithm on very small datasets (e.g., *CA-GrQc* and *CA-HepTh*). However, for small datasets, we do not need to apply the approximate algorithm since the exact algorithm can achieve both high efficiency and accuracy.

To compare the running time of the M-O algorithm and ApproxMR, we not only measured the numbers of MapReduce rounds of the MapReduce process but also measured the actual execution time in seconds. Table IV shows the comparison of the numbers of rounds and the execution time. Based on the analysis in Section IV-B, we set the numbers of rounds to 7 which is enough to get suitable reduced size of dataset for in-memory computing for all the datasets. As shown in the table, the M-O algorithm is terminated by 7 rounds while ApproxMR is terminated by at least 10 rounds. Also, the M-O algorithm is much faster than ApproxMR. From Table IV, we can conclude the M-O algorithm is more time-efficient than ApproxMR.

D. Connectivity of the Discovered Densest Subgraph

In this section, we measure the connectivity of the densest subgraph discovered by the M-O algorithm and ApproxM-R. Figure 8 shows the percentage of connected subgraph-s among the 8 discovered densest subgraphs from the 8 real-world datasets in the M-O algorithm compared to Ap-proxMR. Only 87.5% of the densest subgraph discovered by ApproxMR in all the datasets are connected. Although most of the densest subgraphs discovered by ApproxMR are connected, the lack of connectivity guarantee still in-

TABLE IV
THE NUMBER OF ROUNDS AND THE EXECUTION TIME OF THE ALGORITHMS

Datasets	Rounds		Time (second)	
	M-O	ApproxMR	M-O	ApproxMR
Wiki-Vote (small)	**7**	12	**187**	367
CA-GrQc (small)	**7**	16	**172**	482
Email-Eron (small)	**7**	10	**192**	312
CA-HepTh (small)	**7**	14	**183**	423
slash (small)	**7**	16	**207**	514
com-youtube (large)	**7**	18	**310**	740
com-lj (large)	**7**	24	**2,756**	4,014
com-orkut (large)	**7**	18	**11,126**	26,175

fluences its application. All the densest subgraphs discovered by the M-O algorithm are connected. This experimental result is consistent with our proved conclusion in Section III. This result confirms that the M-O algorithm can guarantee the connectivity of the discovered densest subgraph.

E. Evaluation on Simulated Natural Graphs

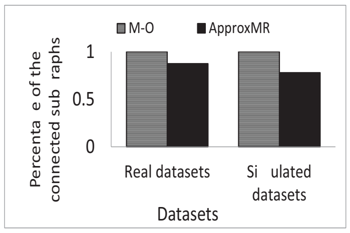


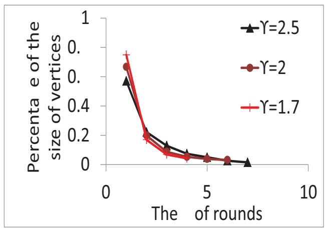
Fig. 8. Connectivity comparison

The previous experimental results show that the M-O algorithm performs well on real-world datasets. We then measure its performance on simulated natural graphs generated by the BTER model [14] for discovering the densest subgraph.

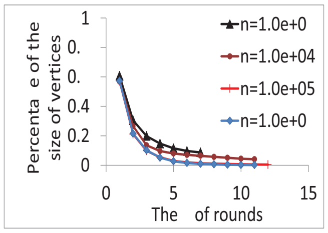
1) Efficiency of graph reduction: Natural graphs usually follows a power law degree distribution with exponent parameter $\gamma \in (1, 3)$ [1]. Therefore, we show the percentage of the size of vertices in the remaining graph versus the number of rounds with different value of γ in $(1, 3)$ and different numbers of vertices separately in Figure 9.

Figure 9(a) shows the results of the simulated datasets with different degree power law parameter γ when we set the number of vertices in the datasets (denoted by n) to 1000. As γ increases, the required number of rounds for reducing the dataset to a suitable size for in-memory computing slightly increases. Figure 9(b) shows the results of the simulated datasets with different number of vertices n when we set $\gamma = 2.5$ which is a most common value for the normal natural graphs [1]. For all the simulated datasets with different sizes, the size of the dataset reduces quickly at the beginning and reaches less than 1% of the initial size only in 10 rounds. The sizes are reduced even faster for the datasets with bigger n. These phenomena are consistent with the phenomena in real-world datasets as in Figure 6 and Figure 7. Therefore, we conclude that the M-O algorithm is suitable for the big natural graphs with power law degree distribution and community features (which is different from the BA network introduced in section III).

2) Density of the Discovered Densest Subgraph: Figure 10 shows the density of the discovered densest sub-



(a) With different γ



(b) With different n

Fig. 9. Performance of the graph reduction on simulated datasets

graph of the M-O algorithm and ApproxMR on each of the 50 randomly simulated graphs. As shown in the figure, we can see clearly that our algorithm can find denser subgraph comparing with ApproxMR. These results match the experimental results on real-world datasets in Table III.

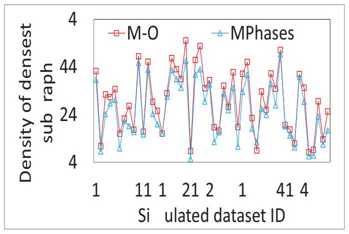


Fig. 10. The density of the discovered densest subgraph in 50 simulated datasets

3) Connectivity of the discovered densest subgraph: We compare the connectivity of 50 densest subgraphs discovered by the M-O algorithm and ApproxMR in Figure 8. All of the densest subgraphs discovered by the M-O algorithm are connected, but there are 22% of the densest subgraphs discovered by ApproxMR which are disconnected.

V. RELATED WORK

The problem of discovering the densest subgraph in undirected graphs was first proposed by Goldberg [3] who gave a solution with time complexity $O(logn)$. The algorithm transforms the initial problem to a series of the min-cut max-flow problems. Later, several subproblems were derived from the traditional densest subgraph discovery problem. A similar polynomial time complex algorithm for discovering the densest subgraph in a directed graph was proposed by Khuller *et al.* [8]. The exact algorithms work well when the dataset size is small, but lack the capability to handle big data. Therefore, various approximate and heuristic algorithms were proposed to meet the computing time and space challenges of big data. Charikar [9] proposed a simple greedy algorithm for solving the original densest subgraph problem, which leads to a 2-approximation to the optimum. Later, this algorithm was improved in a MapReduce framework [10]. Samir *et al.* [8] developed fast polynomial time algorithms for several variations of the dense subgraph problem for both directed and undirected graphs. When it comes to heuristic algorithms. A shingling technique based dense subgraph discovering algorithm is designed by Gibson *et al.* [7] in the application of link spam in World Wide Web. In our previous work [11], we also designed a heuristic densest subgraph discovering algorithm based on the unique feature of natural graphs. However, our new algorithm focuses on the exact solution based on solid theoretical analysis, while our previous heuristic algorithm focuses on the time efficiency only with a tradeoff of precision of results.

Furthermore, all the above algorithms fail to distinguish their applicability to different kinds of graphs. However, it is well-known that the degree distribution of many real-world networks follow a power-law [13], which means that although there may be millions of nodes, most of the nodes are impossible to be in the densest subgraph. In this paper, we discuss the applicability of our exact algorithm to different kinds of graphs. Also, we take advantage of the power-law feature to design our algorithm for discovering the densest subgraph discovering.

VI. Conclusion

In this paper, we proposed the M-O algorithm which combines the MapReduce computing with the in-memory computing on one computer together to improve the efficiency and precision of discovering the densest subgraph. The M-O algorithm consists of two computation phases: 1) the graph reduction in the MapReduce framework, and 2) densest subgraph discovery in memory. We then theoretically proved the correctness and the connectivity of the discovered densest subgraph of the two computational phases. We proved that 1) the density of the BA network (the complex network without community structures) is uniform; 2) the density of BTER network (the complex network with community structures) is nonuniform; and 3) the M-O algorithm is well adapted to the graphs with nonuniform density. Finally, we conducted extensive experimental evaluations on both real-world datasets and simulated graphs to compare the performance of the M-O algorithm with ApproxMR. Experimental results shows that 1) only 87.5% of the densest subgraphs discovered by ApproxMR are connected, while all the densest subgraphs discovered by the M-O algorithm are connected; 2) the graph reduction phase reduces the number of vertices to about 1% of the initial size and about 0.1% for large datasets; this large-size reduction makes it possible to run the densest subgraph discovery phase on one computer; and 3) the M-O algorithm uses fewer MapReduce rounds than ApproxMR on the computing process. The experiment conclusions are consistent with our theoretical results.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, Microsoft Research Faculty Fellowship 8300751.

References

[1] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review*, vol. E 69, 2004.

[2] L. Wan, B. Wu, N. Du, Q. Ye, and P. Chen, "A new algorithm for enumerating all maximal cliques in complex network.," *ADMA*, vol. 4093, 2006.

[3] A. V. Goldberg, "Finding a maximum subgraph," *Technical report*, 1984.

[4] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *PNAS*, vol. 101, pp. 2658–2663, 2004.

[5] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," in Proc. of WWW, pp. 1481–1493, 1999.

[6] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *IEEE Trans. Knowl. Data Eng.*, vol. 24, pp. 1216–1230, 2012.

[7] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *in Proc. of VLDB*, pp. 721–732, 2005.

[8] S. Khuller and B. Saha, "On finding dense subgraphs," *ICALP*, vol. 14, 2009.

[9] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph.," in *APPROX*, vol. 1913 of *Lecture Notes in Computer Science*, pp. 84–95, Springer, 2000.

[10] B. Bahmani, R. Kumar, and S. Vassilvitskii, "Densest subgraph in streaming and mapreduce," *CoRR*, vol. abs/1201.6567, 2012.

[11] B. Wu and H. Shen, "Discovering the densest subgraph in mapreduce for assortative big natural graphs," in *Proc. of ICCCN Workshop on BDeHS*, pp. 1–6, 2015.

[12] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, no. 393, pp. 440–442, 1998.

[13] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.

[14] C. Seshadhri, T. G. Kolda, and A. Pinar, "Community structure and scale-free collections of er graphs," *CoRR*, vol. abs/1112.3644, 2011.

[15] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun ACM*, vol. 51, pp. 107–113, 2008.

[16] V. Adamchik and H. M. Srivastava, "Some series of the zeta and related functions," *Analysis*, vol. 18, pp. 131–144, 1998.

[17] "Stanford network analysis project," 2013. https://snap.stanford.edu/.

[18] "Apache hadoop," 2015. http://hadoop.apache.org/.

[19] M. Newman, "The structure and function of complex networks," *Review, SIAM*, 2003.

[20] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proc. of KDD*, pp. 837–846, ACM, 2009.

[21] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg, "Signed networks in social media," *CoRR*, vol. abs/1003.2424, 2010.

[22] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Laws of graph evolution: densification and shrinking diameters," *Knowledge Discovery*, vol. 1, pp. 1–40, 2006.

[23] B. Klimt and Y. Yang, "Introducing the enron corpus," in *CEAS*, 2004.

[24] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Laws of graph evolution: densification and shrinking diameters.," *Knowledge Discovery*, vol. 1, pp. 1–40, 2006.

[25] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters.," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[26] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *CoRR*, vol. abs/1205.6233, 2012.

[27] B. V. Cherkassky and A. V. Goldberg, "On implementing the push-relabel method for the maximum flow problem.," *Algorithmica*, vol. 19, no. 4, pp. 390–410, 1997.