Discovering the Densest Subgraph in MapReduce for Assortative Big Natural Graphs

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

Abstract*—Discovering the densest subgraph is important in graph analysis, which has wide-ranging applications from social network community mining to the discovery of biological network modules. However, the previous algorithms neglect the connectivity of the dense subgraph since it is a challenge to give consideration to both subgraph structure and time efficiency. As a result, it may lead to isolated subgraphs in the output though they aim to find one connected dense subgraph. Also, there are lacking of efficient algorithms for big natural graphs, especially considering datasets become increasingly larger in this era of Big Data. Furthermore, previous algorithms fail to take advantage of various features of natural graphs (e.g., power-law degree distribution, homophyly of vertices, and power-law community size) which can be applied to improve the efficiency and precision of the densest subgraph discovery. To handle these problems, we design a heuristic algorithm for discovering the connected densest subgraph for massive undirected graphs in a MapReduce framework by taking advantage of the features of natural graphs. Experimental results show that our algorithm is capable of discovering the connected densest subgraph. Also, it can reduce the running times by 62% for average and discover denser subgraphs in 50% of the real datasets and 88% of the simulated datasets comparing with previous algorithm in a MapReduce framework.

***Index Terms*—Densest subgraph, Community, Natural graph, Heuristic algorithm**

I. INTRODUCTION

A challenge in the analysis of complex networks is to discover densest subgraph. The densest subgraph is often interpreted as “communities” [1–3]. Various algorithms [4–6] are proposed for solving the densest subgraph problem. The most important problem in the previous algorithms [4–6] is that they ignore the connectivity of the returned densest subgraph. It means that the returned subgraph may consist of several isolated connected components that maximize its density. For example, previous algorithms [4–6] may correctly find the two isolated components as one community that are not connected to each other in Figure 1. Also, in spite of many proposed densest subgraph discovery algorithm, there are lacking of efficient algorithms for massive graphs, especially considering datasets become increasingly larger in this era of Big Data. Furthermore, natural graphs are completely different from random graphs. They are uniquely featured by small world phenomenon [7], power-law degree distribution [8], high clustering coefficient [9], power-law community size distribution [10], self-similarity [11], hierarchical modularity [12], densification laws, shrinking diameters [13], assortativity of vertices [14] and so on. Actually, these features can help us improve the precision and efficiency of the densest subgraph discovery. However, previous algorithms [4–6] rarely take advantages of these unique features of natural graphs. Therefore, it is a challenge to take advantages of the unique features in densest subgraph discovery.

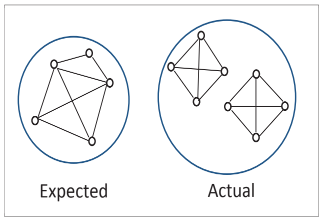

Fig. 1. An example of failing to find connected densest subgraph.

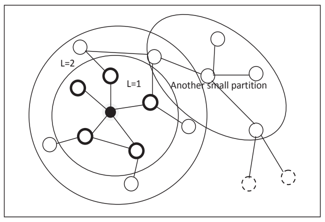

Fig. 2. An example of components when L = 1 and L = 2, respectively.

In this paper, first, based on the unique features of natural graphs, we derive some useful theorems for guiding the algorithm design. Then, we design a heuristic densest subgraph discovery algorithm that is more advantageous than the previous algorithms in that it is more time and space efficient and its outcomes are more precise. In the experiments, we apply our algorithm for massive natural graphs and compare the performance with previous algorithms [5, 6] to show its superior performance. We further verify the features of natural graphs based on the results of our algorithm, since it is designed by the features of natural graphs themselves.

To sum up, our paper can benefit the study of dense subgraph discovery from the following aspects:

1) We discuss the connectivity of the densest subgraph discovered by previous algorithms [4–6] and provide a heuristic densest subgraph algorithm based on our derived properties by taking advantage of natural graph features, which overcome the connectivity problem.
2) We apply our algorithm for massive natural graphs and compare its performance with previous algorithms [3, 5, 6].

The rest of this paper is organized as follows. Section II presents the related work. Section III describes the design of our densest subgraph discovery algorithm and the analysis of the algorithm. Section IV presents extensive experimental evaluation on massive real-world graphs and simulated graphs to show the performance and scalability of our algorithm in practice in a MapReduce computing framework(Mapreduce is running based on the data structure in $< key, value >$ pairs.) in comparison with other algorithms [5, 6]. Section V summarizes the paper with remarks on our future work.

II. Related Work

Dense subgraph: Goldberg [3] formally introduced the original problem of discovering the densest subgraph in an undirected graph and gave an algorithm that requires $O(\log n)$ running time (n is the number of vertexes in the graph) to find the optimal solution based on the min-cut technique. Khuller *et al.* [15] proposed a similar polynomial time complex algorithm for discovering densest subgraph in a directed graph. Several subproblems were derived from the traditional densest subgraph problem. Feige *et al.* [16] defined and studied the dense k-subgraph maximization problem, which is applied to find a set of k vertices with maximum average degree in the subgraph. Asahiro *et al.* [17] defined and studied the problem of discovering a k-vertex subgraph of a given graph G that has at least $f(k)$ edges. Saha *et al.* [4] defined the densest subgraph problems with distance (i.e., the diameter of output subgraph) restrictions and specific subset restrictions (a specific subset must be in the output), and provided algorithms for them. Also, they studied the problem of discovering the multiple almost densest subgraphs [4]. When it comes to big data era, due to the complexity of dense subgraph problem, various approximate and heuristic algorithms were designed to meet the computing time and space challenges. Charikar [5] described a simple greedy algorithm and showed that it leads to a 2-approximation to the optimum. This algorithm was improved by Bahmani *et al.* [6], which leads to within a factor $2(1+\varepsilon)$ of the optimum in a MapReduce framework. Samir *et al.* [15] developed fast polynomial time algorithms for several variations of dense subgraph problems for both directed and undirected graphs. Some heuristic algorithms [18, 19] were also designed based on different techniques. Gibson *et al.* [18] solved the discovering dense subgraphs problem based on a shingling technique, while Chen *et al.* [19] solved the same problem by the matrix blocking technique.

However, the previous dense subgraph discovery algorithms neglect the connectivity, which may lead to isolated sever components in the output subgraph, which are deviated from the original goal of connected subgraph discovery. Also, the previous algorithms are not efficient enough and scalable to handle massive datasets, which is very common in this Big Data era. Further, they fail to leverage the unique features of natural graphs, which otherwise can be used to improve the efficiency and precision of the algorithms. Unlike these previous algorithms, our work can handle these problems. By taking advantages of the features of natural graphs, we design a heuristic algorithm for discovering the densest connected subgraph for undirected massive graphs in a MapReduce framework.

III. Dense Subgraph Discovery Algorithm

In this section, guided by the natural graph features observed in previous studies, we design a heuristic densest subgraph discovery algorithm for undirected natural graphs. Our aim is not only to design the algorithm itself, but also to verify the unique features of natural graph structures. To be more specific, if our heuristic algorithm performs better, it can be used as an evidence to verify our theorems, and further the unique features of natural graph structure.

A. A High-Level Description

Intuitively, we assume that we can get all the vertices from one vertex in the dense subgraph by traversing on the edges in a small number of steps since dense subgraphs have very small diameters. The start points of the traversing should be the centers of the dense subgraphs. Although the subgraphs retrieved by traversing may contain some vertices that do not belong to the dense subgraphs, we can still easily eliminate it by the traditional method in [3] if the size of the subgraph is small enough. In other words, this method partitions the intractable big dataset to many small components, which still contain the dense subgraphs, but can be processed easily by traditional method [3], which assume that the memory is large enough to hold all the data.

Based on this rationale, we present a densest subgraph discovery algorithm. The algorithm partitions the graphs into overlapping small components, which contain the dense subgraphs. An example is shown in Figure 2. We then use traditional densest subgraph discovery algorithm [3] to discover the dense subgraphs in the small components very quickly since each entire small component can be stored in memory. Next, we measure the densities of the dense subgraphs which have been found in each small component to discover the densest subgraph and top dense subgraphs. We prove that this algorithm achieves high efficiency for the densest subgraph and dense subgraphs problems in the experiment section. Besides, our algorithm can guarantee the connectivity of the output subgraph since it considers structure of the graphs.

B. Parameter Determination

We need to determine three parameters for our algorithm: i) the number of traversing steps, ii) the criteria of seed selection (The seeds are the vertices chosen for further partition), and iii) the number of seeds. Below, we explain the details of the parameter determination.

The number of traversing steps (denoted by L): The quality of a subgraph is measured by the density of the subgraph; higher density means higher quality. If L is too large, then the size of each component will be too large; if L is too small, then we may not discover high-quality dense subgraphs. However, previous study [10] shows that the densest subgraph of a natural graph should have a very small diameter. Therefore, we infer that setting L to 1 is enough to discover qualified densest subgraphs since we can traverse all the vertices from any vertex within at most 2 steps in these subgraphs. Figure 2 shows an example of the components when $L = 1$ and $L = 2$, respectively.

Criteria of seed selection: Suppose we can properly set the value L, then the selected seeds are the main factor for the effectiveness of the algorithm in discovering dense subgraphs. An ideal method is to select the most highly connected vertex from each dense subgraph so that we can discover each dense subgraph non-repeatedly. However, this task is non-trivial since we do not know the places of the dense subgraphs in the graph, or whether two vertices are in the same dense subgraph. Therefore, we use a heuristic method which chooses vertices with higher degrees as seeds since high-degree vertices are more likely to be in dense subgraphs than low-degree vertices. However, for a large graph, choosing a vertex with the highest degree may lead to a memory overload, considering the small world feature of natural graphs. Also, based on previous study [10], we know that the densest subgraph should be small and we only need components with a size of order of 100 even for a graph with a million vertices as the previous example. Therefore, we select the seeds which have the suitable degrees so that it will be with high probability to find the densest subgraph.

The number of seeds (denoted by m): If the theoretical analysis is absolutely true, then we only need to choose one vertex to get the densest subgraph since we can exactly find a vertex in the densest subgraph. However, we should recognize that there must be a randomness in the real world. Therefore, we may need to choose more than one vertex, and compute the densest subgraph from multiple component candidates. Fortunately, the power-law degree distribution of natural graphs guarantees that even if the graph is large, there are a limited number of qualified seeds since the densest subgraph is small and each vertex in the densest subgraph has a narrow range of degree. For example, for a graph with a million vertices and has a degree power-law exponent $\gamma = 2$, there are only about 100 vertices with degree about 100. Also, for the MapReduce framework, a limited number of parallel processes do not influence the time efficiency. Therefore, we can select multiple seeds without at the cost of degraded algorithm efficiency by taking advantage of the MapReduce framework.

Since our algorithm is heuristic, in Section IV, we experimentally study the influence of the parameters on the effectiveness of the densest subgraph discovery algorithm, and find that a floor level of parameters, i.e., setting L to 1, and m to 0.01% of all the vertices, can guarantee a good performance.

C. Process of the Algorithm

Our dense subgraph discovery algorithm has two phases: 1) graph partition, and 2) densest subgraph discovery. Step 1 has the following two sub-phases.

(i) Sub-phase 1: we measure the degree of each vertex, and select the top m suitable degree vertices as seeds.

(ii) Sub-phase 2: we partition the large graph to the small components, which are generated by traversing the graph from the seeds for L steps. Then we can discover the densest subgraph from the small components

In Step 2, we can apply any traditional algorithms to discover the densest subgraph contained in the small components.

Our algorithm can be easily implemented in parallel computing frameworks such as MapReduce. Below, we introduce the details of phase 1: graph partition.

Algorithm 1: Seed selection in a MapReduce program

```
1: Mapper
     Input: $< u, v >$
     emit $< u, v >$;
   end
   Reducer
     Input: $< u, neighbor\_list >$
     if $|neighbor\_list| > threshold$ then
       emit $< u, \$ >$;
     end
   end
```

Figure 3 illustrates the flowchart of the graph partition. Algorithm 1 shows the process of seed selection in the first step in Figure 3. The input of this MapReduce program is the edge list, which records each edge (u, v) in the graph twice by (u, v) and (v, u) (u and v are the two endpoints of the edge). If vertex u is selected as a seed, then we add an output $< u, \$ >$, which will be used in the later part of the algorithm. The parameter m can be determined by different realistic demands.

Algorithm 2 shows the process of tagging seeds in each edge in the initial edge list, and it outputs the dataset with seeds tagged with *. Then, in Algorithm 3, we use two MapReduce rounds to traverse one more hop from the seeds, and tag all the edges whose two endpoints both belong to the vertex set in the traversal. In the first MapReduce round, we tag the edges in the traversal path. In the second MapReduce round, we tag the remaining edges whose both two endpoints belong to the vertex set in the traversal.

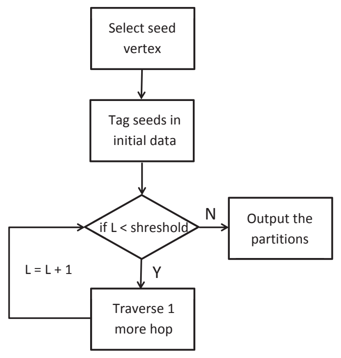

Fig. 3. Flowchart of the graph partition

D. Densest Subgraph Discovery

After we partition the large graph to small components, discovering the densest subgraph has become easy since each small component can be stored in memory and handled easily by traditional algorithms. In this paper, we can simply use *ExactAlg* [3] which is a classic polynomial algorithm to find the exact densest subgraph. *ExactAlg* is an exact polynomial time algorithm for discovering the densest subgraph for an undirected graph based on a min-cut max-flow technique. The algorithm transfers the densest subgraph problem to a series of min-cut problems and finds the densest subgraph by recursively constructs a new graph based on the initial graph and find the min-cut (S, T) on the new graph. Finally, the densest subgraph can be derived from the min-cut (S, T). The detailed design is presented in [3]. Since the small components are data independent from each other, we can process them in parallel in MapReduce.

Algorithm 2: Tag seeds in each edge

```
1: Mapper
     Input: $< u, v >$ and $< u, \$ >$
     emit Input;
   end
   Reducer
     Input: $< u, neighbor\_list >$
     if $\$ \in |neighbor\_list|$ then
       $neighbor\_list = neighbor\_list \backslash \$$;
       foreach $v$ in $neighbor\_list$ do
         emit $< u^*, v >$;
       end
     else
       foreach $v$ in $neighbor\_list$ do
         emit $< u, v >$;
       end
     end
   end
```

Algorithm 3: Traverse 1 more hop based on current component

```
1: Mapper
     Tag the edges in the traversal path.
   end
   Reducer
     Generate the edge list.
   end
2: Mapper
     Tag the remaining edges whose two endpoints both
     belong the vertex set in the traversal.
   end
   Reducer
     Generate the edge list.
   end
```

IV. Performance Evaluation

In this section, we present the performances of our dense subgraph discovery algorithm in comparison with not only *ExactAlg* but also *GreedyAlg* (greedy algorithm) and *PGreedyAlg* (parallel greedy algorithm). *GreedyAlg* [5] is a sequential greedy algorithm for discovering the densest subgraph for an undirected graph G. It greedily deletes the vertex with the smallest degree one by one 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. It has been proved that *GreedyAlg* can guarantee a 2-approximation for $\rho^*(S)$. *PGreedyAlg* [6] is an extension of *GreedyAlg* which is implemented for parallelism

TABLE I
Description of the ten datasets

ID	Description	$\|V\|$
1	Collaboration network of Arxiv General Relativity	5,242
2	Social circles from Facebook (anonymized)	4,039
3	DBLP collaboration network	317,080
4	Amazon product network	334,863
5	LiveJournal online social network	3,997,962
6	Enron company email list	36,692
7	Wikipedia who votes on whom network	7,115
8	Slashdot social network from November 2008	77,360
9	Arxiv High Energy Physics paper citation network	34,546
10	Web graph of Notre Dame	325,729

in MapReduce. Instead of deleting the vertex one by one, *PGreedyAlg* deletes a batch of vertices with degrees smaller than $(2+\varepsilon)$ times of the current density of the remaining graph in every two rounds of MapReduce process and records the current density of the remaining graph. After all the vertices are deleted from the initial graph, we obtain the densest subgraph from the records. It has been proved that *PGreedyAlg* can guarantee a $(2+\varepsilon)$-approximation for $\rho^*(S)$.

We implemented our algorithms on Hadoop (hadoop.apache.org) and ran it on 4 PCs; each PC is equipped with 2.1GHz Intel core i3 processor with 2 cores, and a 2GB memory. The algorithms are implemented in Python. The numbers of Mappers and Reducers in MapReduce are set manually. We used the datasets in Table I in the experiments. Since we tag the vertices with small component ID, there is an extra external memory for storing them in hard disk. Therefore, we need to evaluate the extra external memory for our algorithm. We evaluate our algorithm with focuses on two aspects. We first evaluate the extra external memory, running time, the densities of the discovered subgraphs of our algorithm with different parameters. We then compare these performance metrics of our algorithm with *GreedyAlg*, *PGreedyAlg* and *ExactAlg*.

A. Dataset Description

We first introduce the datasets from the SNAP library [20] used in our experments as shown in Table I. The table lists the name, the number of vertices, the number of edges, diameter, and a short description of each dataset. The detailed information are posted on http://snap.stanford.edu. We give each dataset an ID that will be used in the rest of the paper for expression convenience. Dataset 1 to Dataset 6 are undirected graphs. Dataset 7 to Dataset 10 are directed graphs. These directed graphs have similar natural graph features as the undirected graphs [8]. Therefore, we just treat them as undirected in order to enrich our datasets as the previous work did.

B. Effect of Parameters on Performance

We all know that external memory is not a bottleneck nowadays for computing environment. However, the linearly producing extra data produced increases the I/O costs, which is a core consideration for a MapReduce algorithm. The efficiency of an algorithm is determined by the running time, extra

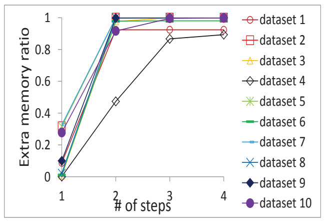

Fig. 4. Extra memory ratio vs. # of transverse steps

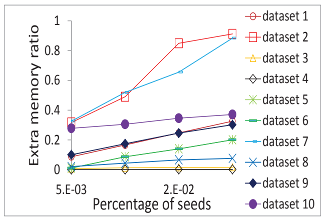

Fig. 5. Extra memory ratio vs. # of seeds selected

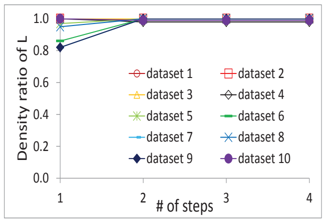

Fig. 6. The density ratio vs. # of transverse steps

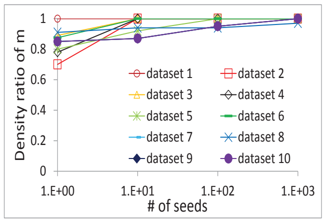

Fig. 7. The density ratio vs. # of seeds selected

external memory usage, the number of discovered qualified dense subgraphs. In this section, we study the following problems:

1) How does the number of traversing steps (L) influences the efficiency of the algorithm?
2) How does the number of the seeds (m) influences the efficiency of the algorithm?

Recall we select top suitable degree vertices as seeds. This process can guarantee that the components found by seeds have suitable sizes. We first used the top 0.01% of suitable degree vertices as the seed and measured the extra external memory with the increasing of parameter L. We define *extra memory ratio* as the ratio of the extra external memory divided by the memory used to store the initial datasets. Figure 4 shows the extra memory ratio versus L. As the figure shows, the extra external memory increases sharply with the increasing of L in the beginning due to the small world phenomenon [7], in which any two vertices can reach each other in a limited steps along the edges. The increasing speed of individual extra external memory depends on the diameter of the dataset. For example, for Dataset 5, the extra external memory needed is almost equal to the initial data size when $L = 2$, while for Dataset 4, the extra external memory needed increases much slower.

In addition to the parameter L, the size of extra external memory is also influenced by the number of seeds selected. Therefore, we set $L = 1$, and study the influence of the number of seeds on the size of extra external memory used. Figure 5 shows the extra memory ratio versus the percentage of number of seeds selected. We see that the external memory usage approximately increases linearly as the percentage of number of seeds increases.

Then, we study the influence of parameter L on the density of the densest subgraph. We set the number of seeds as 0.01% of the number of all vertices, and measure the density of the densest subgraph discovered with different values of parameter L. We define *density ratio of* L of a certain value of L as the ratio of the density of the densest subgraph found by the current value of L divided by the density of the densest subgraph discovered by setting L from 1 to the diameter of the initial graph. Figure 6 shows the density/densest density value versus the value of L. As the figure shows, for some of the datasets, it is intriguing to see that the quality of the subgraphs does not always increase as L increases. As L increases, the qualities slightly decrease in Dataset 2, Dataset 4 and Dataset 10, remain nearly constant in Dataset 1 and Dataset 3, but slightly increase in Dataset 5 to 9. We can conclude that the qualities of the discovered subgraphs are relatively constant with increasing of L. Considering that L increase will not significantly improve performance, we just set $L = 1$ for measuring the extra external memory. As Figure 4 shows, the extra external memory is far less than the dataset itself when $L = 1$, and the data can be handled efficiently.

Then, we define the *density ratio of m* of a certain value of m as the ratio of the density of the densest subgraph found by the current value of L divided by the density of the densest subgraph found by setting m from 1 to the number of vertices in initial graph. Figure 7 shows the density ratio of m versus the number of seeds selected when $L = 1$. As the figure shows, for most of the datasets, we can get the best results using the top 0.01% suitable degree seeds. For Dataset 10 that performs slightly worse, it achieves relatively higher performance with top 0.01% suitable degree seeds, which are still far less than 0.01% of all the vertices. Therefore, we conclude that using less than 0.01% vertices as seeds is sufficient for all the datasets for the densest subgraph discovery.

Based on the experimental results of Figure 4, Figure 5, Figure 6 and Figure 7, we can conclude that for all the datasets, we only need to set $L = 1$ and use less than 0.01% of all the vertices to get almost the best results with the different parameter values. Since the number of seeds is small and $L = 1$, the extra external memory is less than 0.01% of the size of the initial dataset.

C. Algorithm Efficiency and Effectiveness on Real-World DataSets

In this section, we evaluate the precision of the dentest subgraph discovery of our algorithm, and compare it with *GreedyAlg* and *PGreedyAlg* for discovering densest subgraph. First, we set the number of seeds as 0.01% of the number of all the vertices, $L = 1$, and run our algorithms on the datasets in Table I. Then, we compare the density of the densest subgraph discovered by our algorithm, *GreedyAlg* and *PGreedyAlg*. In *PGreedyAlg*, we set parameter $\varepsilon = 0$ for this algorithm to reach its best performance. Table II shows the comparison of the final results, where $\rho^2(G)$ presents the results of *PGreedyAlg*, $\rho^1(G)$ presents the results of *GreedyAlg*, and $\rho(G)$ presents the results of our algorithm. As shown in the table, our algorithm has the best performance on Dataset 1, Dataset 2, Dataset 3, Dataset 4 and Dataset 10. *GreedyAlg* has the best performance in Dataset 1, Dataset 2, and from Dataset 5 to Dataset 9. We can see that our algorithm performs the best

TABLE II
THE DENSITY OF THE DENSEST SUBGRAPH FOUND BY DIFFERENT ALGORITHMS

$G = (V, E)$	$\rho^2(G)$	$\rho^1(G)$	$\rho(G)$
Dataset 1	22.39	22.39	**22.39**
Dataset 2	69.97	77.35	**77.35**
Dataset 3	56.50	56.50	**56.56**
Dataset 4	4.08	3.76	**4.79**
Dataset 5	35.31	**37.33**	36.26
Dataset 6	35.31	**37.33**	32.09
Dataset 7	43.91	**46.25**	38.00
Dataset 8	38.65	**42.27**	40.74
Dataset 9	28.20	**30.17**	25.42
Dataset 10	78.43	78.43	**78.66**

TABLE III
THE COMPARISON OF THE NUMBER OF ITERATIONS AND EXECUTION TIME (UNIT/SECOND)

$G = (V, E)$	$i^2(G)$	$i(G)$	$t^2(G)$	$t(G)$
Dataset 1	8	**4**	160	**81**
Dataset 2	9	**4**	184	**82**
Dataset 3	10	**4**	257	**105**
Dataset 4	15	**4**	312	**84**
Dataset 5	10	**4**	2,175	**905**
Dataset 6	11	**4**	589	**84**
Dataset 7	9	**4**	184	**82**
Dataset 8	11	**4**	245	**90**
Dataset 9	11	**4**	238	**88**
Dataset 10	7	**4**	180	**108**

in 50% of the real datasets. Although our algorithm performs the worst in Dataset 6, Dataset 7 and Dataset 9, it still reaches at least 80% of the best results. *GreedyAlg* can achieve higher precision sometimes at the cost of low efficiency. However, *GreedyAlg* is not suitable for massive datasets due to the memory limitation, which is a critical requirement for Big Data. We emphasize the best performances with bold type in all the tables in the evaluation.

Another consideration for the comparison between our algorithm and other algorithms is the time efficiency. Since *GreedyAlg* cannot be parallelized and is not suitable for big datasets, we only compared the time efficiency between our algorithm and *PGreedyAlg* in the same situation. For the metrics, we not only compared the iteration times of the MapReduce process, but also compared the running time. Table III shows the comparison of the number of MapReduce iterations and the execution time, where $i^2(G)$ presents the number of MapReduce iterations of *PGreedyAlg*, $i(G)$ presents the number of MapReduce iterations of our algorithm, $t^2(G)$ presents the execution time of *PGreedyAlg* and $t(G)$ presents the execution time of our algorithm. As shown in the table, our algorithm is terminated by 4 iterations since $L = 1$, while *PGreedyAlg* is terminated in at least 7 iterations. Also, our algorithm is much faster than *PGreedyAlg*. Therefore, we can conclude our algorithm is more time-efficient than *PGreedyAlg*. On average, we reduce the running time by 62%.

V. CONCLUSION

In this paper, we designed a simple but efficient heuristic densest subgraph algorithm guided by the theorems, which can easily overcome existing problems in previous algorithms such as the neglect of connectivity and inefficiency for massive datasets. Then, we presented extensive experiments and compared with other algorithms based on real datasets and simulated graph datasets. Experimental results showed that our algorithm is capable of discovering the connected densest subgraph. Also, it can reduce the running times by 62% for average and discover denser subgraphs in 50% of the real datasets and 88% the simulated datasets comparing with previous algorithm in a MapReduce framework. To guarantee a good performance of our algorithm in an arbitrary situation, in near future, we will improve our algorithm by providing a theoretical guarantee for assortative natural graphs. Also, we will explore taking advantages of the features of natural graphs to efficiently discover multiple dense subgraphs, overlapping dense subgraphs, dense subgraphs in different scales according to the necessities of different applications.

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.," in *ADMA*, vol. 4093, pp. 606–617, 2006.

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

[4] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang, "Dense subgraphs with restrictions and applications to gene annotation graphs.," in *RECOMB*, vol. 6044, pp. 456–472, Springer, 2010.

[5] 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.

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

[7] S. Milgram, "The small world problem," *Psychology Today*, vol. 61, pp. 60–67, 1967.

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

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

[10] 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, pp. 29–123, 2009.

[11] C. Song, S. Havlin, and H. A. Makse, "Self-similarity of complex networks," *Nature*, vol. 433, pp. 392–395, 2005.

[12] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási, "Hierarchical organization of modularity in metabolic networks," *Science*, vol. 297, pp. 1551–1555, 2002.

[13] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. of KDD'05*, 2005.

[14] S. Zhou and R. J. Mondragn, "The rich-club phenomenon in the internet topology.," *IEEE Communications Letters*, vol. 8, pp. 180–182, 2004.

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

[16] U. Feige, G. Kortsarz, and D. Peleg, "The dense k-subgraph problem.," *Algorithmica*, vol. 29, pp. 410–421, 2001.

[17] Y. Asahiro, R. Hassin, and K. Iwama, "Complexity of finding dense subgraphs," *Discrete Applied Mathematics*, vol. 121, pp. 15–26, 2002.

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

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

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