Accelerating Community-Search Problem through Faster Graph Dedensification

Kifayat Ullah Khan, Tu Nguyen Anh,
Mostafijur Rahman Akhond, Waqas Nawaz*, Young-Koo Lee
Dept. of Computer Science and Engineering, Kyung Hee University, Republic of Korea
Email: {kualizai, tunguyen, mostafij, wicky786, yklee}@khu.ac.kr
*Islamic University Almadinah, Saudi Arabia

***Abstract*—Community-search is the problem of finding a densely connected subgraph from a large graph, for given set of query nodes. It is useful, for example, when some high profiled researchers want to arrange a conference or some celebrities wish to arrange a party, and are searching for other researchers and similar personalties respectively. Technically, the problem involves maximizing the minimum degree to find a highly connected subgraph. Well known $Greedy$ algorithm for this purpose, iteratively deletes nodes with minimum degree to meet certain objective function. We observe that Greedy operates in an in-efficient manner due to densely connected regions in a graph, referred as $Hot\ Spots$. In this paper, we provide a new concept of performing community-search on a dedensified graph. Our aim is to sparsify the hot spots to accelerate global searching method of Greedy to make it applicable on a large graph. Recently a graph dedensification approach has been proposed that adds $Compressor\ Nodes$ in a graph for dedensification. However, this method is in-efficient since it has to traverse entire graph to compress the hot spots. To solve this problem, we propose a faster graph dedensification algorithm by using Locality Sensitive Hashing (LSH). We improve time complexity of existing graph dedensification method from $O(|E|)$ to $O(|N|+|e_{HDN}|+k)$, where N, E, e_{HDN}, and k are nodes, edges, edges of High Degree Nodes (HDN), and hash functions respectively, and $|N| >> |HDN|$. Once the graph is dedensified, we use it to accelerate community-search operation. We perform experiments on two real world graphs, and observe significant improvement in execution time of Greedy algorithm, add lesser compressor nodes, and perform reduced traversals for graph dedensification.**

I. Introduction

Graph is a versatile data model to represent various real world concepts. Interaction between chemical compounds, relationships among users of online social networks, hyper-links in web pages, are all modeled and represented as graphs. This versatility is the main cause that researchers from different parts of the world, has keen interest in using them. Such overwhelming interest has also become source of increasing size of underlying graphs in respective domains. Hence, applying various mining techniques like community-search among others, is in-efficient. Therefore, it is necessary to compress dense regions of a large graph, to accelerate the mining process [13, 14, 10, 9].

Searching a community of nodes for given set of query nodes, is gaining attention in recent past [19, 5, 12, 18]. This is useful for variety of applications like arranging a research conference by certain researchers or planning a party by some celebrities. The problem involves, given some query nodes and a large graph, find a densely connected community of nodes, including the query nodes. The density criteria is based on well known minimum degree measure [17], where aim is to maximize the minimum degree of participating nodes. Sozio and Gionis [19] present the pioneering work for this task and present a Greedy algorithm, that is the extension of initial study by Asahiro et al. [1]. Greedy proceeds by removing every node from the graph, whose degree is minimum among rest of the nodes. The process is continued when degree of any of the query node is lesser than a node to be removed in current iteration or the subgraph becomes disconnected. It is observed that majority of real world graphs follow power law degree distribution [11, 2] containing comparatively smaller number of nodes, having high degree. Therefore, graph structure around these nodes is much denser. We call such dense regions as hot spots, encircled region in Figure. 1 (a), and understand that their sparsification reduces degree of large number of nodes. Consequently, execution time of Greedy algorithm can be improved by reducing number of Graph Edit Distance (GED) [21] operations. For example, consider a graph G and its dedensified graph $G^{/}$ (dedensification to be discussed shortly) in Figure 1. We find that for set of query nodes {8, 9, 10}, node 7 has minimum degree in hot spot. Applying Greedy on G requires four GED operations for its removal and that of its edges, whereas it requires only two operations in $G^{/}$ to generate required community structure for Figure. 1 (c). Therefore, we find a positive effect of hot spots sparsification for acceleration of community-search problem.

Recently, Maccioni and Abadi present state of the art solution for graph dedensification [14]. Their idea is to reduce congestion around HDNs of a graph by decreasing edges around them. For this purpose, they introduce a compressor node in the graph. A compressor node re-directs connectivity of HDNs from other low degree nodes, and reduces number of edges from the graph. For instance, consider Figure. 1 (a) where set of nodes $\{3, 4, 5, 6, 7\}$ are connected with our three query nodes {8, 9, 10}. A compressor node is added in the given graph that changes the connectivity type in it, thus reducing the number of edges, as shown in Figure. 1 (b). Such addition of compressor nodes significantly reduces congestion around HDNs and are source of sparsifying hot spots.

We find that graph dedensification technique by Maccioni and

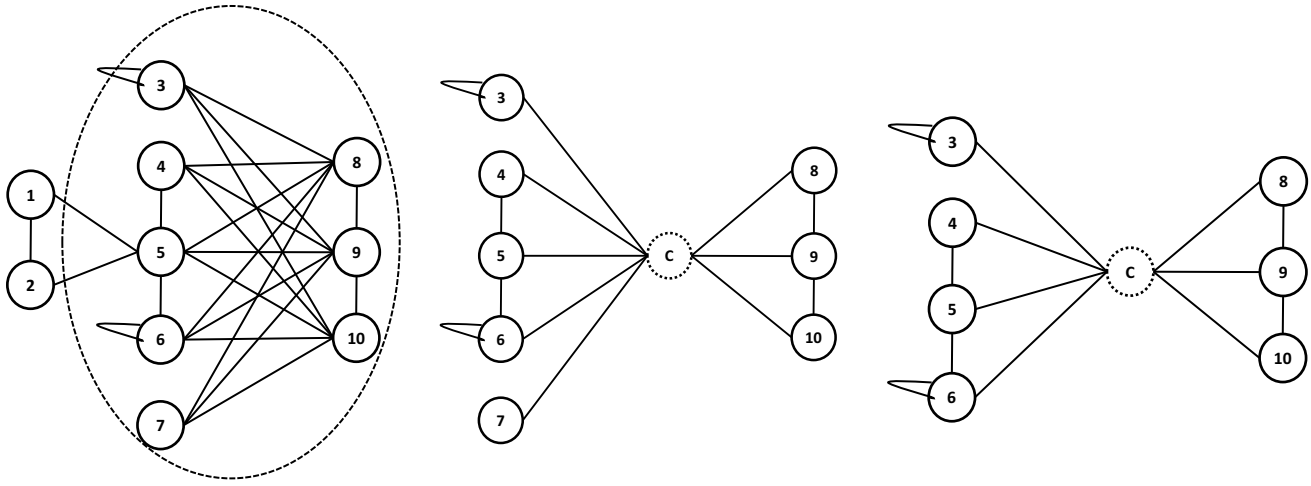


Fig. 1. (a) Original Graph G with encircled showing a hot spot (b) Dedensified Graph $G^{/}$ by addition of a Compressor Node (c) Community for Query Nodes $\{8, 9, 10\}$

Abadi [14] is useful, however, their proposed algorithm for this purpose is in-efficient. The authors proposed algorithm traverses every node in the graph and for each node, they investigate its neighborhood to locate HDNs. This makes time complexity for number of graph traversal operations as $O(|E|)$, which is high for a large sized graph having millions of nodes and edges. To solve this problem, we aim to locate hot spots by hashing query and HDNs only using LSH [3, 8]. LSH hashing technique helps avoid exhaustive search for hot spots with high accuracy. In our approach, we hash every query and HDN and then retrieve them in clusters to add compressor nodes. In this way, we save time to traverse neighborhood of all the nodes from underlying graph. Using this strategy, time complexity of our graph traversals reduces to $O\left(|N|+|\sum_{i=1}^{n<<N} Nbrs(u_i)|+k\right)$ which can be simplified as $O(|N|+|e_{HDN}|+k)$. Here, n are HDNs, $|N|$ are total nodes, $Nbrs(u)$ is neighborhood of a node u, and k are number of hash functions used. This is low as n and k are much lesser than total number of nodes and edges in a graph. Therefore, our proposed approach is much faster for graph dedensification.

We summarize contributions of our paper as below.

- We present a new notion of accelerating the community-search problem. Our proposal is to dedensify a large graph to speed-up the well known Greedy algorithm for aforementioned problem.
- We propose a fast dedensification algorithm where we directly locate Hot Spots in a graph for lossless compression. Our algorithm utilizes LSH to avoid exhaustive search for hot spots and reduces traversal of entire graph.
- We evaluate our proposals against state of the art solutions in terms of execution time, addition of compressor nodes for dedensification, and graph traversal operations. The results demonstrate superiority of our proposed approach in every evaluation aspect.

II. Related Works

This section presents overview of existing studies related to our work. We review existing studies related to both community-search problem and graph dedensification. Whereas focus of this paper is on accelerating community search process, however, we understand that summary graph generated by proposed dedensification scheme can be utilized for any other mining task as well.

1) Community-Search Problem: Community-search is a variant of classical community detection problem. Objective of community detection problem is to discover various clusters of nodes which can either be overlapped or isolated. Fortunato and Hric [6] present a nice guided tour of this problem for clear understanding. On the other hand, community-search problem requires finding a community of nodes for given set of query nodes. The result is a dense and connected subgraph structure, where all the nodes have strong connectivity with each other and also with query nodes.

Sozio and Gionis [19] pioneered the community-search problem which was later improved and extended by various researchers [5, 12, 18]. Sozio and Gionis [19] aim to maximize the minimum degree of each node, to obtain a dense subgraph. They present Greedy algorithm that performs global traversal on a graph for subgraph discovery. Whereas, Greedy generates an optimal solution, however, its global nodes traversal policy is in-efficient for a large graph. Hence, Cui et al. [5] present a local approach which is efficient. In this paper, we aim to accelerate global graph traversal strategy by Sozio and Gionis [19] to make it applicable for a large graph. We observe dense regions in a graph, are main bottleneck which effect traversal efficiency. Therefore, we perform lossless compression of such regions to meet our objective. In this regard, we are the first to present the concept of community-search problem on a dendensified graph for performance acceleration of Greedy algorithm.

2) Graph Dedensification: Graph dedensification or summarization is an interesting research direction, where aim is to generate a compact summary of a large graph for efficient graph mining operations [15, 20, 5, 10, 9]. Maccioni and Abadi [14] present the most recent research study to perform lossless graph dedensification. They identify dense subgraph structures and add a new compressor node in it for dedensification. Addition of compressor node is an extension of pioneering work by Buehrer and Chellapilla [4] which was later studied by Cecilia and Navarro [7]. On the other hand, various graph summarization studies also present compression techniques to minimize the description length of resultant summary graph [15, 10, 9].

In this paper, we focus on the most recent graph dedensification approach by Maccioni and Abadi [14]. We observe that the authors proposed method performs unnecessary traversals of the graph and has to add many compressor nodes for dedensification. We, therefore, aim to directly locate dense regions and perform their lossless compression. In this way, we not only reduce graph traversal operations but also add fewer number of compressor nodes.

III. Preliminaries and Problem Statement

In this section, we present overview of Greedy algorithm for community search problem, a state of the art approach for graph dedensification, and their issues. We also present formal definition of our notion of Hot Spots. Finally, we present the problem statement of our paper.

Algorithm 1 Community-Search by Greedy

Require: Graph $G\,(V, E)$, Query Nodes $Q = \{q_1, ..., q_k\} \in V$
Ensure: A dense subgraph containing query nodes
1: **for** ($i = 1$ to $|V|$) **do**
2: Delete v_i and its edges if $Degree\,(v_i)$ is minimum
3: $G_i = G_{i-1}$
4: **if** (degree($q_j < v_i$) or $(q_1, ..., q_k) \in Q$ become disconnected) **then**
5: Break
6: **end if**
7: **end for**
8: **Return** G_i

A. Greedy Algorithm for Community-Search Problem

Greedy algorithm by Sozio and Gionis [19] identifies a dense community structure for given query nodes. The authors maximize the minimum degree of each node for dense subgraph discovery. Given input graph G, Greedy iteratively deletes a node having minimum degree to produce G_t at t-th step. Every G_t is denser than each G_{t-1} and consequently generates required dense community structure for given query nodes. The iterative node deletion stops when either any of the query node has lesser degree than node currently selected or subgraph becomes disconnected. We present pseudocode of Greedy algorithm by [19] in Algorithm 1.

Greedy is an optimal solution to find a dense community structure, however, it incorporates two main issues in terms of its execution strategy. First, visiting every node in G is a global graph traversal approach where aim is to visit every $v \in V$ to evaluate its degree. A global approach ensures an optimal solution, however, it performs in-efficient traversal when G is large. Second, nodes connectivity in hot spots is very high. Hence, removing each minimum degree node requires large number of GED operations. We understand that reducing the search space for global traversal of G is appropriate for both of its key issues.

B. Graph Dedensification

Maccioni and Abadi [14] present a state of the art approach for graph dedensification. According to the authors, high connectivity around HDNs is the source of congestion around them. They identify such regions by locating HDNs in neighborhood of every node. In this way, edges of all the nodes having exact same connectivity with corresponding HDNs, are reduced by adding a new compressor node in the graph. This addition reduces congestion around HDNs and is useful to accelerate any mining operation.

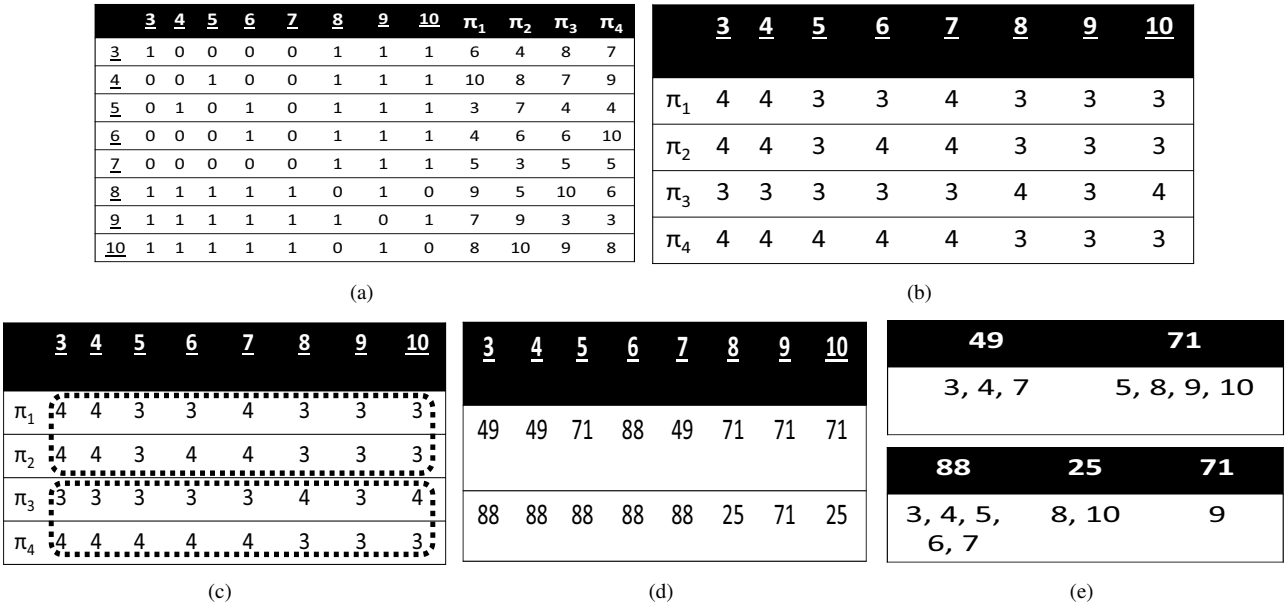
We understand that exhaustively searching HDNs in neighborhood of each node incorporates two main issues. First, there exists many nodes without any connection with any of the require HDN. However, the existing approach has to traverse every such node, thus consumes unnecessary CPU resources. Second, finding complete set of nodes having exact same connectivity with certain HDNs, is not performed perfectly. Hence, the algorithm generates many small clusters which ultimately requires large number of compressor nodes for dedensification. We believe that aforementioned issues can be sorted out if we directly access HDNs only and search complete set of corresponding nodes for dedensification.

C. Hot Spot

Hot spot is dense region of a graph and is created due to high connectivity of HDNs. It has been investigated that HDNs are frequently visited during graph traversal operations [16]. Therefore, sparsifying these dense region facilitate fast graph traversals. We now formally define our notion of hot spots in Definition 1. For sake of brevity, we split member nodes of a graph into HDNs and Low Degree Nodes (LDN).

Definition 1 $[Hot\ Spot]$ Given a directed graph $G(V, E)$ containing LDN and HDN, where $HDN = \{h_1, h_2, ..., h_h\} \in V$, $LDN = \{l_1, l_2, ..., l_l\} \in V$, in-degree of each $h_i \in HDN$ is above a given threshold $degree_t$ and that of every l_j is below $degree_t$, $HDN \cup LDN = V$, and $HDN \cap LDN = \phi$. Then, for each subset LDN_x of nodes from LDN and every subset HDN_y from HDN, if each node $l \in LDN_x$ has outlink with every node $h \in HDN_y$, the resultant subgraph structure is called a Hot Spot.

Compressing each hot spot is a lossless subgraph sparsification and does not introduce any connectivity mistake. We illustrate sample hot spot in Figure 1 (a). The example



	3	4	5	6	7	8	9	10	π_1	π_2	π_3	π_4
3	1	0	0	0	0	1	1	1	6	4	8	7
4	0	0	1	0	0	1	1	1	10	8	7	9
5	0	1	0	1	0	1	1	1	3	7	4	4
6	0	0	0	1	0	1	1	1	4	6	6	10
7	0	0	0	0	0	1	1	1	5	3	5	5
8	1	1	1	1	1	0	1	0	9	5	10	6
9	1	1	1	1	1	1	0	1	7	9	3	3
10	1	1	1	1	1	0	1	0	8	10	9	8

(a)

	3	4	5	6	7	8	9	10
π_1	4	4	3	3	4	3	3	3
π_2	4	4	3	4	4	3	3	3
π_3	3	3	3	3	3	4	3	4
π_4	4	4	4	4	4	3	3	3

(b)

	3	4	5	6	7	8	9	10
π_1	4	4	3	3	4	3	3	3
π_2	4	4	3	4	4	3	3	3
π_3	3	3	3	3	3	4	3	4
π_4	4	4	4	4	4	3	3	3

(c)

3	4	5	6	7	8	9	10
49	49	71	88	49	71	71	71
88	88	88	88	88	25	71	25

(d)

49	71
3, 4, 7	5, 8, 9, 10

88	25	71
3, 4, 5, 6, 7	8, 10	9

(e)

Fig. 2. Illustration of applying LSH on a graph (a) Adjacency matrix representation of sample graph from Fig. 1-a, along with four hash functions of random permutations based on node ids (b) Minhash matrix (c) Division of matrix into bands (d) Combined hash codes for each portion of columns (e) Hash tables containing buckets of candidate similar nodes.

illustration is an un-directed graph, however, is valid to describe a hot spot.

D. Problem Statement

Given a large directed graph $G(V, E)$, set of query nodes $Q = \{q_1, ..., q_k\} \in V$, and a degree threshold $degree_t$, our goal is to minimize number of graph edit distance operations to generate a dense community structure for Q, through effectively sparsifying hots spots using minimum number of graph traversal operations.

IV. The Proposed Solution

In this section, we present our proposed graph dedensification algorithm and explain how it accelerates community search problem.

A. Compressor Nodes

We utilize idea of compressor nodes by Maccioni and Abadi [14] to sparsify the hot spots. A compressor node is a new node added to the graph to redirect the connectivity among nodes. This redirection reduces the number of edges in each hot spot, hence useful. Edge reduction is lossless, therefore accuracy of any mining algorithm, remains intact. Every addition of a compressor node between LDN_i and HDN_j, reduces the number of edges from $|LDN_i|.|HDN_j|$ to $|LDN_i|+|HDN_j|$. In this way, the resultant graph is much sparser compared to its original version.

B. fastDedense, The Proposed Algorithm

We now present our proposed algorithm, $fastDedense$, for hot spots sparsification to dedensify a large graph. Key idea of fastDedense is based on two main points for efficient execution while providing same dedensification accuracy to state of the art solution. First, we avoid exhaustive searching for HDNs in neighborhood of each $v \in V$. For this purpose, we process only HDNs and avoid spending computational resources on LDNs. We use LSH, which is a fast similarity search technique that can approximate HDNs having common neighborhood, hence we gain speed-up. Next, we directly identify hot spots by retrieving clusters of HDNs. Common neighborhood of every such cluster is then filtered to obtain a hot spot, which is then sparsified by adding a new compressor node. We now present details of our proposed method using illustrations and present its pseudocode in Algorithm 2.

1) Generating Candidate Similar HDNs via LSH: To avoid exhaustive searching in neighborhood of each node, we only visit a node to check whether its indegree satisfies the degree threshold or not to declare it as HDN. On locating each HDN, we apply LSH on it to locate other similar HDNs. Applying LSH involves two main steps: (i) generating minhash signature matrix and (ii) generating candidate similar nodes. We explain both of these steps below from our previous work [10, 9], by considering graph in Figure 1 (a) for clear understanding.

Minhash matrix of a graph is compact alternative of its adjacency matrix representation. It has same number of columns to that of adjacency matrix, however, much lesser number of rows. Each column holds minhash signatures of each $v \in V$, computed from some hash functions. These signature columns approximate the Jaccard similarity among nodes with high accuracy as shown in Equation 1. u and v are a pair of nodes in Equation 1 and $Nbrs$ represent their respective neighborhood.

$$JaccardSimilarity\,(u, v) = \frac{(Nbrs(u) \cap Nbrs(v))}{(Nbrs(u) \cup Nbrs(v))} \quad (1)$$

We generate signatures columns of the minhash matrix by applying hash functions of random permutations on neighborhood of each HDN. Figure 2 shows step by step process to generate minhash columns. We show adjacency matrix of graph from Figure 1 (a) in Figure 2 (a) along with four hash functions $\{\pi_1, ..., \pi_4\}$ of random permutation. We then apply each hash function on neighborhood of each node to find the minimum value. For example, applying π_1 on neighborhood of node 3 outputs value 4 as the minimum value. Similarly, applying rest of the hash functions generates remaining of its minimum values. This process is repeated for every node to construct a complete minhash matrix.

Minhash matrix of G is then used to discover candidate similar HDNs. For this purpose, we split entire column of every node into b bands of r rows each, as shown in Figure 2 (c). Combined hash code for each portion of the column is then computed to filter out the nodes sharing same hash values, as shown in Figure 2 (d). Finally, each band forms a hash table containing buckets of candidate similar nodes. Key of the bucket is a unique hash code and its members are the nodes having the given hash code. For instance, Figure 2 (e) shows two hash tables having buckets of candidate similar nodes. We find that nodes $\{6, 7, 8\}$ have exact same neighborhood, hence they appear together in both hash tables. In this way, we finish generating candidate similar HDNs.

2) Hot Spots Sparsification: As soon as all the HDNs are hashed, we iteratively choose every HDN_i as a query node and retrieve its candidate similar HDNs from every hash table. Any HDN_j having even single hash code collision with HDN_i is declared as candidate similar and is retrieved. We then union all the retrieved candidate similar HDNs for HDN_i. Using this cluster of HDNs and neighborhood of HDN_i, we get a potential hot spot. We then intersect their respective neighborhood to get common neighbors. We then add a compressor node in G by dropping original edges between HDNs and their common neighborhood. In this way, each hot spot is sparsified using a compressor node. Any HDN visited in current iteration is not picked as query node, however, can be utilized with other compressor nodes.

C. Accelerated Community-Search

On obtaining a dedensified graph, we apply Greedy algorithm on it for our community search problem. Graph traversal mode of Greedy remains the same as described in Section III-A, however, its execution efficiency is drastically improved. The process is accelerated as each iteration requires lesser number of GED operations to remove a node, having minimum degree, and its corresponding edges. This happens only when a node to be removed is linked with a compressor node in a hot spot. Number of GED operations are reduced to only one operation in case of a node when it is connected with its entire neighborhood through a compressor node. Therefore, execution time of Greedy algorithm is significantly improved and becomes proportional to density of the underlying graph.

Algorithm 2 fastDedense

Require: Graph $G(V, E)$, Threshold for High Degree Nodes $degree_t$, Hash Tables l, Hash Functions k
Ensure: A Dedensified Graph $G^/$

```
for each (v_i ∈ V) do
    if (indegree(v_i) ≥ degree_t) then
        Declare v_i as high degree node hdn_i
        Compute its minhash signature column MinCol_{v_i} by using k hash functions
        Compute l hash codes from MinCol_{v_i} and store it in hash tables
    end if
end for
for each (hdn_i) do
    Retrieve all candidate similar nodes for hdn_i from every hash table
    Intersect neighborhood of retrieved set of high degree nodes to find their exact common neighbor nodes
    Declare obtained cluster of nodes as a Hot Spot
    Add a compressor node to sparsify the Hot Spot
end for
Return G^/
```

TABLE I
DESCRIPTION OF DATASETS USED FOR EVALUATIONS.

Dataset	Nodes	Edges
Stanford Web Graph	281,903	23,12497
Berkeley-Stanford Web Graph	685,230	76,00595

V. EXPERIMENTAL EVALUATION

We now present experimental evaluation of our proposed algorithm. We performed experiments on two real world graphs, obtained from Stanford Dataset Collection Library. Details of both datasets are shown in Table I. We implemented all the algorithms in Java and performed experiments on 64 bit Windows 7 enterprise edition server having Intel Core i7-3960X with 3.30 GHz processor and 36 GB main memory.

We compare our proposed idea against the most recent graph dedensification algorithm by Maccioni and Abadi [14]. Community-search evaluation is performed using Greedy algorithm by Sozio and Gionis [19]. We use two different criteria to compare effectiveness of our proposals. For community-search problem, we check execution time and GED operations of Greedy on original graph, dendensified graph by our and by [14]. Similarly, we compare our algorithm for dedensification, fastDedense, against Dedense from [14] in terms of number of compressor nodes and graph traversal operations (visiting count of unique edges) for dedensification.

A. Evaluations for Community-Search Problem

We demonstrate results for execution time and GED operation for community-search problem in Figures. 3 and 4 respectively. Execution time evaluation of Greedy proves that proposed approach for this problem is effective, as Greedy consumes higher time while running on an original graph.

Both the execution time and GED operations are reduced on dedensified graph as all the hot spots are already sparsified. In this way, degree of large number of nodes exists in reduced form. Performance of Greedy accelerates, specifically, when encountering compressor nodes during traversal since it requires only one edge removal operation. Thus significantly reducing number of GED operations and saving time.

We observe that GED operations on dedensified graph by our proposed algorithm are greater than those from others in Figure. 4 (b). Greedy removes same number of nodes for both the datasets, however requires large number of said operations when graph is dendensified by our proposed algorithm. This changed behavior exists, as edges of resultant community graph in this case, are much smaller compared to those generated by others. Hence, large number of GED operations are performed.

B. Comparing Graph Dedensification

We now describe performance evaluation for dedensification process. Figures. 5 and 6 respectively show comparison of graph traversal operations and addition of compressor nodes for dedensification.

Graph traversal analysis in Figure. 5 shows that proposed fastDedense visits lesser number of unique edges to identify required hot spots for compression. Each hash function is a random permutation of nodes in a graph, and is applied on neighborhood of each HDN to compute the minimum value for minhash signature column. Therefore, entire neighborhood of a node is visited, equal to number of hash functions. fastDedense restricts the traversals within neighborhood of HDN only. On the other hand, existing Dedense has to visit entire graph for this purpose, hence, is in-efficient. We understand that proposed approach is also highly suitable for any disk-based solution since it traverses certain adjacency lists only. Such adjacency lists once read, does not require further I/Os, hence is suitable for very large graphs which cannot reside in main memory.

Figure. 6 compares number of compressor nodes added by both algorithm for dedensification. We observe that fastDedense introduces significantly lesser number of compressor nodes, hence has better effectiveness. We illustrate this situation through a simple example for clear understanding. For instance, Dedense traverses neighborhood of a node x to find its HDNs. If it finds, say 3 HDNs, then an entry is saved in a map for x. If there is another node y, having exact same 3 HDNs in its neighborhood, then above described entry in the map is updated. Now consider the scenario, when a node z appears in a subsequent iteration, having connectivity with only 2 of the 3 above mentioned HDNs. Now, a new entry is added in the map, rather than updating the previous entry for nodes x and y. Existing entry cannot be updated since doing so violates the definition of hot spot, resulting in introduction of an edge mistake. Therefore, Dedense has to add multiple compressor nodes for sparsification. On the other hand, fastDedense directly identifies clusters of HDNs at once. Each of the cluster is directly sparsified, hence, lesser number of compressor nodes are introduced.

VI. Conclusion

In this paper, we present a new notion of performing community searching for given set of query nodes, on a dedensified graph. We understand that this novel concept can facilitate search for an optimal community in various cases like using streaming graph, discovering influence-based communities, and so on. To accelerate community-search, we reduce size of underlying graph by providing a fast algorithm that performs lossless dedensification of dense regions in a graph. The proposed algorithm avoids traversal of entire graph and effectively discovers the dense regions for dedensification. Experiments on real world and public-ally available graphs, confirm the effectiveness of our proposals. As a future work, we plan to extend our proposed approach for a streaming graph with the aim of analyzing varying structure of communities over the passage of time to search for an optimal dense subgraph.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2015R1A2A2A01008209).

References

[1] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[2] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[3] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[4] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 95–106. ACM, 2008.

[5] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 991–1002, New York, NY, USA, 2014. ACM.

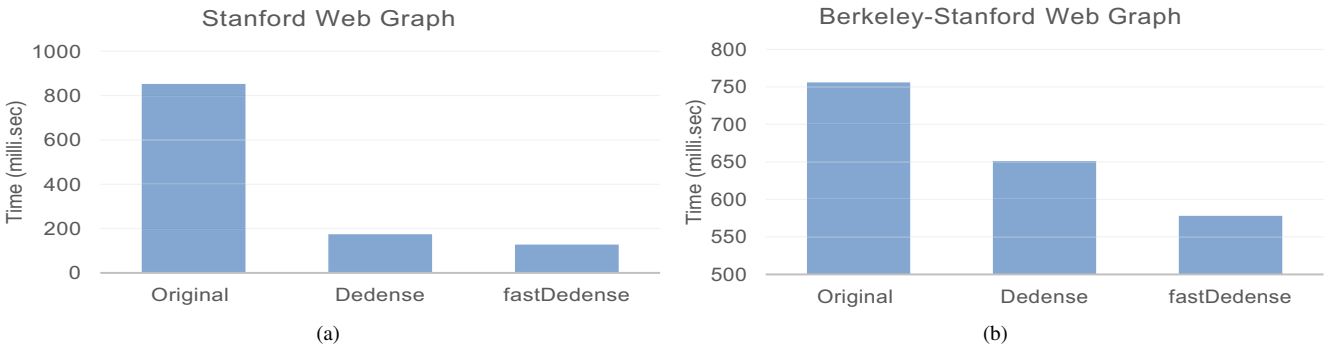
[6] S. Fortunato and D. Hric. Community detection in networks: A user guide. *Physics Reports*, 2016.

[7] C. Hernández and G. Navarro. Compressed representations for web and social graphs. *Knowledge and information systems*, 40(2):279–313, 2014.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

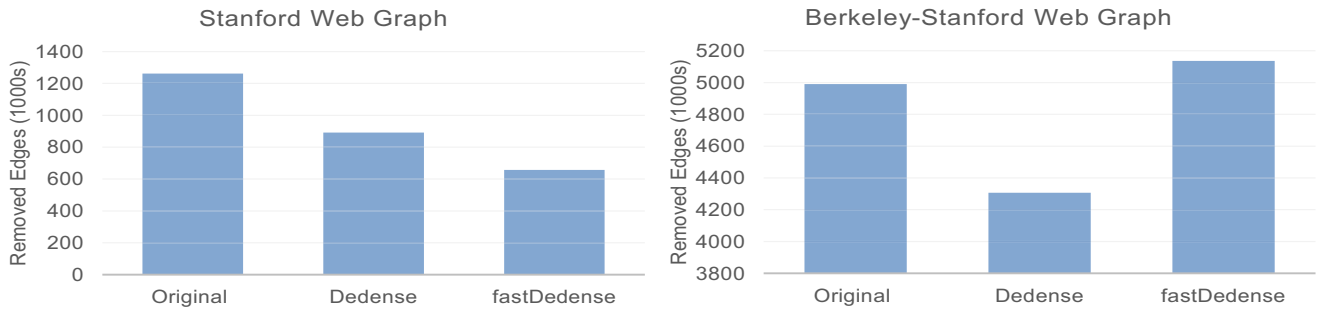
[9] K. U. Khan. Set-based approach for lossless graph summarization using locality sensitive hashing. In *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pages 255–259. IEEE, 2015.

[10] K. U. Khan, W. Nawaz, and Y.-K. Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.



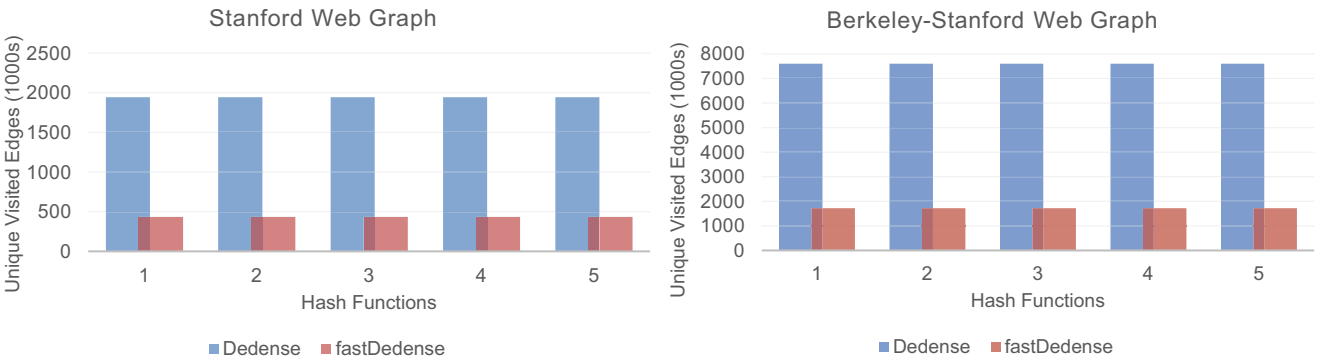
(a) (b)

Fig. 3. Execution time evaluation of Greedy algorithm on original and dedensified graph by existing and proposed method.



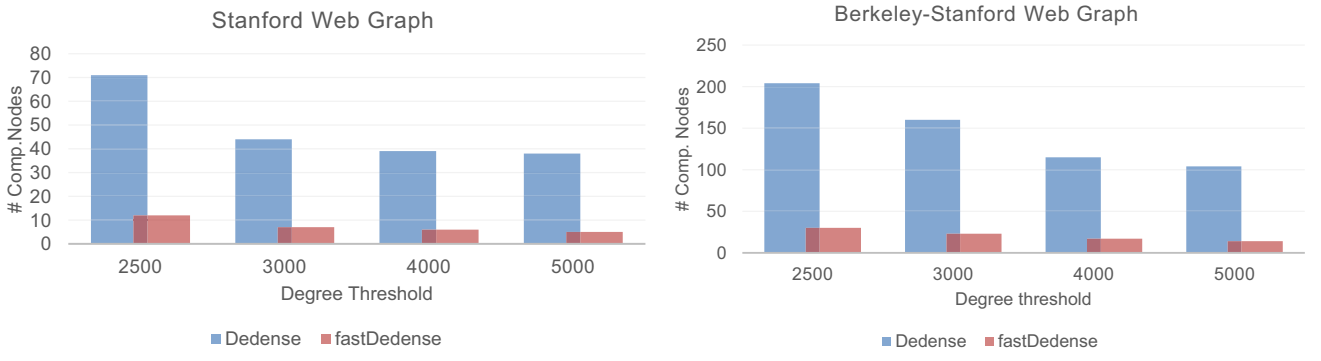
(a) (b)

Fig. 4. Comparing graph edit distance operations based on edges reduction to compute a community from original and dedensified graph.



(a) (b)

Fig. 5. Evaluation of graph traversal operations by varying number of hash functions for LSH.



(a) (b)

Fig. 6. Comparing addition for compressor nodes for graph dedensification at different degree thresholds for discovery of high degree nodes.

[11] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[12] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *Proceedings of the VLDB Endowment*, 8(5):509–520, 2015.

[13] Y. Lim, U. Kang, and C. Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *Knowledge and Data Engineering, IEEE Transactions on*, 26(12):3077–3089, 2014.

[14] A. Maccioni and D. J. Abadi. Scalable pattern matching over compressed graphs via dedensification. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1755–1764, New York, NY, USA, 2016. ACM.

[15] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432. ACM, 2008.

[16] W. Nawaz, K.-U. Khan, and Y.-K. Lee. Spore: shortest path overlapped regions and confined traversals towards graph clustering. *Applied Intelligence*, 43(1):208–232, 2015.

[17] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[18] J. Shan, D. Shen, T. Nie, Y. Kou, and G. Yu. Searching overlapping communities for group query. *World Wide Web*, pages 1–24, 2015.

[19] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 939–948, New York, NY, USA, 2010. ACM.

[20] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973. ACM, 2011.

[21] Z. Zeng, A. K. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: on approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.