Information Sciences 421 (2017) 237–253

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Faster compression methods for a weighted graph using locality sensitive hashing

Kifayat Ullah Khan[a], Batjargal Dolgorsuren[a], Tu Nguyen Anh[a], Waqas Nawaz[a,b], Young-Koo Lee[a,*]

[a] Department of Computer Science and Engineering, Kyung Hee University, Republic of Korea
[b] Department of Information System, College of Computer and Information Systems, Islamic University of Medina, Medina, Kingdom of Saudi Arabia

ARTICLE INFO

Article history:
Received 10 November 2016
Revised 19 July 2017
Accepted 26 July 2017
Available online 29 July 2017

Keywords:
Graph summarization
Set-based merging
Personalized summary graph
Min-wise hashing
Locality sensitive hashing

ABSTRACT

Weights on the edges of a graph can show interactions among members of a social network, emails exchanged in any organization, and traffic flow on roads. However, mining hidden patterns is difficult when the size of the graph is large. Creating a compact summary is useful if it preserves the structural and edge weight information of its underlying graph. Existing work in this context provides a pairwise compression strategy to create a summary whose decompressed version has minimum difference in edge weights compared to its initial state. The resultant summary graph is compact, but the solution has quadratic time complexity due to exhaustive pairwise searching. Therefore, we present a set-based summarization approach that aggregates sets of nodes. We avoid explicit similarity computations and directly identify the required sets via Locality Sensitive Hashing (LSH). LSH accelerates the summarization process, but its hashing scheme cannot consider the edge weights. Considering the edge weight during hashing is necessary when the objective of the required summary is altered to a personalized view. Hence, we propose a non-parametric hashing scheme for LSH to generate candidate similar nodes from the weighted neighborhood of each node. We perform comparisons with state-of-the-art solutions and obtain better results using various experimental criteria.

1. Introduction

Graph summarization is useful for creating a meaningful summary that preserves the intrinsic properties of a large graph. Various graph summarization techniques have been presented to create a compact summary of a large graph [1–10]. Compression of a weighted graph, among other types of graphs, is useful to demonstrate various real-life situations. Weights on the edges depict the number of interactions among individuals, number of co-authored papers, and traffic flow, among other application areas. We present a sample weighted graph in Fig. 1(a) where edge weight represents the number of messages sent from the sender node. Here, a large edge weight between any two nodes shows a strong social bond. We display a summary of our sample graph in Fig. 1(b), where nodes with a similar neighborhood are compressed and the difference in weight for every edge is less than a given threshold (0.25). Non-consideration of edge weights during

* Corresponding author.
E-mail addresses: kualizai@khu.ac.kr (K.U. Khan), bdolgorsuren@khu.ac.kr (B. Dolgorsuren), tunguyen@khu.ac.kr (T.N. Anh), wnawaz@iu.edu.sa (W. Nawaz), yklee@khu.ac.kr (Y.-K. Lee).

http://dx.doi.org/10.1016/j.ins.2017.07.033
0020-0255/

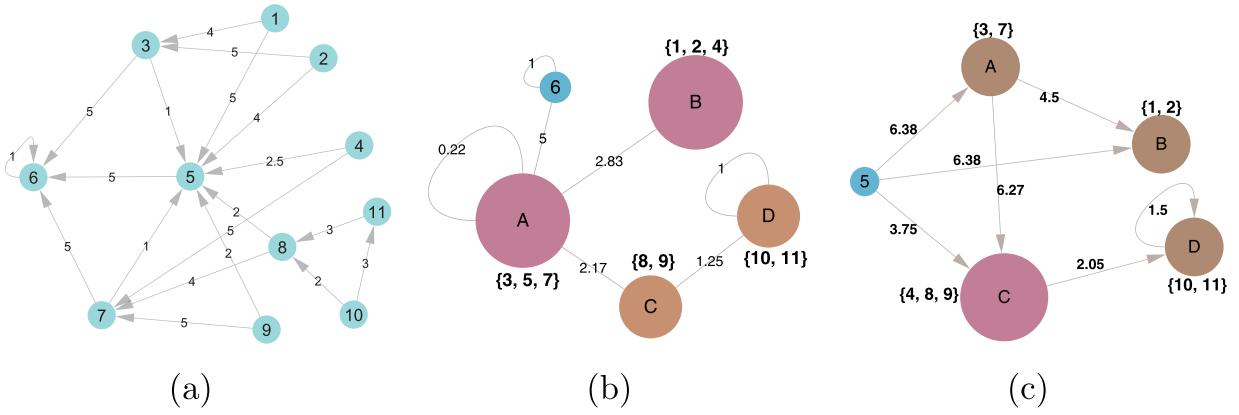


Fig. 1. (a) Sample graph (b) Summary for Basic Weighted Graph Summarization (BWGS) problem by considering our sample graph as undirected (c) Summary for Personalized Weighted Graph Summarization (PWGS) problem using node id 5, where direction of edges is reversed to show influence of from-node.

compression generates a summary graph in which strong and weak social bonding among nodes cannot be distinguished. We refer to this type of compression problem as a Basic Weighted Graph Summarization problem (BWGS)[1] and believe that focusing edge weights during compression is suitable to capture real-world semantics.

Recently, Toivonen et al. have proposed the solution to compress a weighted graph [6]. The authors create a summary graph where each super edge is assigned the mean weight from the edges of each super node so that the resultant summary graph has minimum edge weight difference from its non-compressed version[2]. They iteratively identify and aggregate a pair of nodes with minimum edge weight difference into a super node. A pairwise super node creation strategy generates a highly compact summary graph; however, it has quadratic time complexity. The authors optimize their solution by performing computations among 2-hop neighborhoods, but they aggregate only a pair of nodes in each iteration which is inefficient for a large graph. In this situation, we can improve the execution time by merging multiple nodes in each iteration and restricting similarity computations among highly similar nodes only. Hence, we adopt Locality Sensitive Hashing (LSH) [12], which is a fast similarity search algorithm for high-dimensional data. Using LSH, we avoid similarity computations among least similar nodes [8] and hence make our BWGS problem applicable on a large graph.

We observe that our BWGS problem generates a compact summary graph, but it does not allow a given user to visualize the summary from her personal perspective. A personalized summary graph, for example, can be created for a citation network where we can visualize the influence of a paper on the research community. Similarly, there are a huge number of followers of celebrities on Twitter. A personalized summary graph for a celebrity can provide a simple visualization of her fans/follower base. We call such compression a Personalized Weighted Graph Summarization (PWGS) problem. We illustrate a summary of our sample graph based on the PWGS problem in Fig. 1(c) to provide a personalized view for node id 5. Here, the nodes having equivalent or similar influence from a given input node are compressed into a super node to provide a useful visual analysis.

Influence-preserving graph summarization (IGS) has been recently proposed in [1]. They create a summary graph of a citation network for an influential paper to visualize its influence on the research community. They create a summary graph having k super nodes and l super edges that maximize the flow rate (Section 2.3). IGS is similar in spirit to our PWGS problem, but its nodes summarization solution uses a matrix decomposition-based algorithm having time complexity of $O(md^2)$, where m is total edges and d is average node degree. Furthermore, it requires quadratic space complexity, $O(n^2)$, where n represents the total nodes in the graph. Summarily, these issues make IGS inefficient for a large-sized graph. We observe that utilizing the solution of our BWGS problem for this cause is inappropriate since neighbors with high edge weights, having the potential to maximize the flow rate, are ignored during hashing. We can utilize the hashing solution of [13], but it requires a randomized threshold to filter out the subset of neighbors having higher edge weights for each node. Because correct selection of the threshold is difficult, we propose a modified hashing scheme for our BWGS problem. In this manner, not only can the resulting summary graph be computed efficiently, but highly influenced nodes are also captured.

In this paper, we present two types of compression problems, basic (BWGS) and personalized (PWGS), for a large weighted graph. We propose fast solutions to these problems by using a set-based super node aggregation strategy. To find the required set of nodes, we avoid explicit similarity computations among nodes by adopting LSH. LSH helps to restrict similarity computations among likely similar nodes only by matching hash codes from each node. To meet the objectives

[1] Preliminary work for the BWGS problem is presented in [11].

[2] A super node is a compressed representation of all the nodes having common neighbors, such as node B in Fig. 1(b). Similarly, a super edge is a single edge from a super node such as B that is a compressed representation of the merger of common edges, i.e., edges from nodes 1, 2, and 4 with node A.

of our PWGS problem, we find that utilizing the hashing scheme of our BWGS problem cannot identify the sets of nodes that could maximize the flow rate. Therefore, we present a non-parametric hashing scheme for LSH to generate candidate similar nodes based on similar neighborhoods and higher edge weights with their neighbors for flow maximization. We extensively evaluate our proposed solutions against state-of-the-art methods on three publicly available real-world graphs and one synthetic graph generated through a well-known data generator. The results verify the effectiveness of our proposals using variety of evaluation criteria.

2. Problem definition

2.1. Summary graph

The summary $S_G(V_s, E_s, w_s)$ of a weighted graph $G(V, E, w)$ is a compressed representation whose nodes are called super nodes V_s and edges are referred to as super edges E_s. w of every $(u, v) \subseteq E$ represents the edge weight, whereas the semantics of $w_s(u_s, v_s) \subseteq E_s$ differ for our two types of summarization problems. Each super node $v_s \in V_s$ either contains a pair or set of nodes $\{v_1, ..., v_l\} \in V$ or is preserved as is, i.e., $v_s \in V_s \Rightarrow v_s \in V$. Similarly, the edges from members of each super node are aggregated to form super edges in S_G.

Members of a super node in S_G may constitute a set of nodes from G that represents a clique, bi-clique, or bipartite subgraph. For a clique, every member node shares its entire neighborhood with every other member of the set. However, some of the neighbors may not be common to every other member in the case of a bi-clique or a bipartite subgraph. Similarly, edges from every member node may have different edge weights. Thus, each super node creation may introduce structural mistakes and errors due to edge weights. A structural mistake can further be of two types, i.e., creation of an edge that does not exist in G and deletion of an edge from G. In this situation, our generated S_G introduces the former type of edge mistakes, such as that from [6]. Alternately, we miss an edge if it violates constraints (Section 3.1) imposed for edge weight difference or neighborhood similarity.

To create a super node, we measure the neighborhood (N) similarity for every pair of nodes $(u, v) \in V$ in a set using the Jaccard coefficient *J.Sim* (Eq. (1)). The Jaccard coefficient is useful for measuring similarity between a pair of finite sets. In the case of a graph, each node is attached with a finite set of neighborhoods, hence we can directly benefit from it. Correspondingly, we find the edge weight difference between G and its decompressed (dcomp) graph from S_G using the distance measure [6] in Eq. (2), where G_1 and G_2 refer to the former and latter types of graphs, respectively.

$$J.Sim(u, v) = \frac{(N(u) \cap N(v))}{(N(u) \cup N(v))} \tag{1}$$

$$dist(G_1, G_2) = \sqrt{\sum_{\{u,v\} \in V \times V} (w_{G_1}(\{u, v\}) - w_{G_2}(\{u, v\}))^2} \tag{2}$$

2.2. Basic weighted graph summarization problem

The objective of the BWGS problem is to efficiently compute a concise summary, S_G, for a given large graph, G. The BWGS problem requires S_G to be computed in an in-time and concise manner so that it can support memory-based execution of various graph mining algorithms.

To efficiently compute S_G, we aggregate sets of nodes in each iteration during summary graph creation. This set-based merging traverses multiple nodes in each iteration and thus ends in a short amount of time. Similarly, we control the conciseness of S_G using the compression ratio cr from Eq. (3). In the case of edge weights, we set the weight of each super edge as the mean weight from its member edges to minimize the error.

$$cr(S_G) = \frac{|E_s|}{|E|} \tag{3}$$

Problem 1. Given an undirected graph G and compression ratio cr, $0 < cr < 1$, our objective is to compute its S_G in linear time having $cr(S_G) \leq cr$ and to minimize $dist(G, dcomp(S_G))$ and edge mistakes.

2.3. Personalized weighted graph summarization problem

In some situations, a given $v \in V$, referred to as v_{inf}, wants to visualize G to inspect its influence in the network. Creating an S_G from G may be a compact summary, but it uniformly aggregates every other $v \in V$ based on the given criteria of neighborhood similarity. Hence, the influence of v_{inf} cannot be demonstrated. The influence of v_{inf} on any $v_i \in V$ indicates how many times v_i interacts with v_{inf}, where the higher the interaction count, the higher the influence. We measure the influence of v_{inf} on every $v_i \in N_{v_{inf}}$ using the edge weight of v_i with it. Alternately, the influence of every other x hop $(x > 1)$ away v_j from v_{inf} is based on its respective edge weight with its $x - 1$-hop neighbors towards v_{inf}. Hence, our PWGS problem aims to aggregate nodes having higher edge weight from their neighbors to obtain highly influenced nodes for v_{inf}. We now define the various necessary concepts for our PWGS problem and its problem statement.

Definition 1 (Personalized graph)**.** Given a directed graph G and an influential node $v_{inf} \in V$, a personalized graph $P_G(V_p, E_p, w_p)$ is a directed subgraph of G where every $v_p \in V_p \Rightarrow v_p \in V$ and is accessible from v_{inf}.

We consider G to be a directed graph for our PWGS problem, where every $v \in V$ is attached with its in-links. The weight of every directed $(u, v) \subseteq E_p$ exhibits the influence of v on u, where u and v are source and destination nodes, respectively. As a result, P_G demonstrates a personalized graph whose every member node depicts influence from v_{inf}.

The size of S_G may be large depending upon its underlying P_G, therefore it is desirable to consider only k super nodes and l super edges for better visualization. This restriction generates a personalized summary graph that incorporates the least visual clutter. To select k super nodes from S_G of P_G, we apply a standard DFS or BFS search from v_{inf}. We continue the search until the required k super nodes are retrieved. The edge weight of each super edge is computed as Flow Rate using Eq. (4) [1]. Summarily, the resultant S_G provides a personalized view, where every super edge depicts the amount of influence flow from v_{inf}.

$$\frac{\sum_{v_i \in \varphi_s, v_j \in \varphi_d} w_{ij}}{\sqrt{|\varphi_s||\varphi_d|}} \tag{4}$$

φ_s and φ_d in Eq. (4) represent source and destination super nodes, respectively. The Flow Rate balances the sum of edge weights and aims to avoid inclusion of any node that has the least contribution towards maximizing total flow. We show the Flow Rate for every super edge of the summary graph in Fig. 1(c).

Problem 2. Given a personalized graph P_G and an influential node v_{inf}, we want to compute its S_G, having k super nodes and l flows (super edges), in linear time, to minimize $dist(P_G, dcomp(S_G))$, and to maximize flow rate.

3. Summary graph computation

3.1. Set-based merging for super node creation

The set-based super node creation strategy aggregates a set of nodes in each iteration to create an S_G. This set-based approach is efficient since merging multiple nodes reduces the total number of iterations.

In both of our summarization problems, a super node consists of a set of nodes that satisfy two criteria, i.e., they have similar neighborhoods and the difference in weights of every edge is minimum. Similar neighborhood structure is necessary so that their merger produces the least edge mistakes. Likewise, similar edge weights minimize errors due to edge weight differences between G and the decompressed version of its S_G. Utilizing pairwise checking ($\binom{n}{2}$ iterations) to identify sets of nodes is impractical for a large graph. Therefore, it is desirable to reduce pairwise checking and directly identify such sets.

To identify a set of nodes for compression, we select a query node q, where $q_i = v_i \in V$, to reduce all pairwise similarity computations. We then declare every $v_j \in V$ as compressible with q_i if their Edge Reduction Ratio (EDR) [4] is above a given threshold. Eq. (3) shows the EDR, where s is a prospective super node on a possible compression of q and v. The maximum value of the EDR is 0.5 when q and v share all of their neighbors with each other, while its minimum value can be any large negative number due to the large number of non-common neighbors between them. On setting the EDR threshold as 0.5, only the nodes having entirely common neighborhoods are merged with each other. Hence, the resultant summary graph is not highly compressed since only a limited number of nodes have such property. Alternately, a lower threshold for the EDR allows the merger of even those nodes that have the least-shared neighborhoods. In this manner, the summary graph is highly compressed, but it introduces a large number of edge mistakes. Let us explain computation of the EDR for the sake of illustration. We find that the EDR of nodes 2 and 4 in Fig. 1(a) is 0.25 since the resulting super node has three edges towards nodes 3, 5, and 7. We find that there is no edge between nodes 2 and 7, as is the case for nodes 4 and 3. Hence, this merger produces a super node that introduces two edge mistakes due to non-connectivity of its member nodes. We further verify the effect of varying the EDR threshold and its relationship with the resultant summary graph through experiments in Section 4.5.4.

$$EDR(q, v) = \frac{|N_q| + |N_v| - |N_s|}{|N_q| + |N_v|} \tag{5}$$

Using q, we approximate its candidate similar nodes and filter them using the EDR to obtain our desirable set for aggregation.

Definition 2 (Set of Candidate Similar Nodes)**.** Given a set of nodes $S = \{v_1, .., v_i, .., v_m\} \in V$, $q \in V$, an EDR threshold t_N, and an edge weight difference threshold t_{Ew}, we call S the Set of Candidate Similar Nodes $SCandN_q$ if the EDR of any $v_i \in S$ with q is below t_N or their difference in edge weights is above t_{Ew}.

Consider node 1 as q and a set of nodes {2, 4, 8, 9} in Fig. 1(a) as $SCandN_q$. We set t_N as 0.2 and t_{Ew} as 0.25. The EDR of q with every other candidate node satisfies t_N, but its t_{Ew} with node 8 (apart from others) is 3 because of their edge weight with node 5. As a result of this violation, aggregating this set into a super node produces a large error due to the difference in edge weights. To overcome these issues, we define our desirable set of nodes for compression as below.

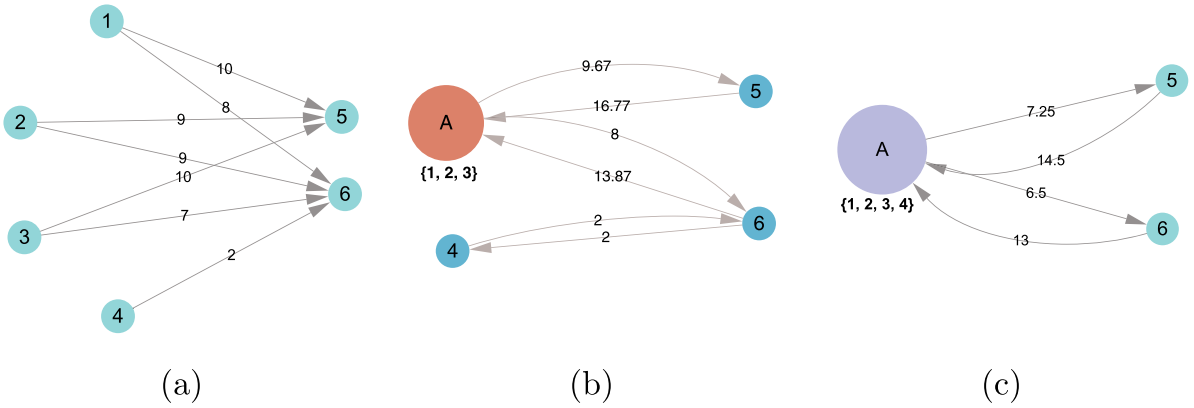


Fig. 2. Illustration to show effect of various thresholds on the summary graph. (a) Toy graph for detailed analysis of various utilized measures. (b) Summary obtained when the EDR threshold is 0.35 and the Edge Weight Difference Threshold is 6. (c) Summary obtained when the EDR threshold is 0.2 and the Edge Weight Difference Threshold is 9. Forward and backward edges show mean edge weight and the flow rate (from nodes 5 and 6) respectively in (b) and (c).

Definition 3 (Set of Compressible Nodes)**.** Given an $SCandN_q$, $q \in V$, an EDR threshold t_N, and edge weight difference threshold t_{Ew}, its largest subset is the Set of Compressible Nodes $SCompN_q$ if the EDR of every $v_i \in SCandN_q$ with q is above t_N and their difference in edge weights is below t_{Ew}.

Using the SCandN {2, 4, 8, 9} for query node 1 from Fig. 1(a) along with the same thresholds, we find that the subset {2, 4} is SCompN since it satisfies both the defined criteria. Therefore, merging this set into a super node generates the least number of edge mistakes as well as the lesser error due to the difference in edge weights. It is to be noted that there exists a 2-hop path between every member node of any *SCompN*; otherwise, they do not have any common neighbor(s).

We now provide another example to clearly demonstrate the effects of the edge weight, EDR threshold, and edge weight difference threshold on a subsequent summary graph using Fig. 2. To summarize the given sample graph from Fig. 2(a), we select node 1 as a query node, whereas nodes {2, 3, 4} are retrieved as its *SCandN*. We observe that node 4 does not satisfy both the criteria to be declared as its *SCompN*, in Fig. 2 (b), as its EDR with node 1 is 0.3 and edge weight difference is 8. On relaxing these thresholds, node 1 ultimately gets merged with its entire given *SCandN*, as shown in Fig. 2(c). We also demonstrate variation in flow rate by merging nodes with a high margin of difference in edge weights. For instance, the flow rate from node 6 to super node A in Fig. 2(b) is 14.11, but it drastically drops to 6.5 when a low-edge-weighted neighbor (node 4) also becomes part of the super node in Fig. 2(c). Summarily, we find that varying the threshold values significantly effect the structure of the resultant summary graph.

3.2. Creating a summary graph for the BWGS problem

The core of our proposed solution is based on LSH to identify similar nodes for compression. Therefore, we first provide justification for using this approach to validate the use of our similarity measure, i.e., Jaccard Similarity, and then the stepwise procedure of applying LSH on a graph.

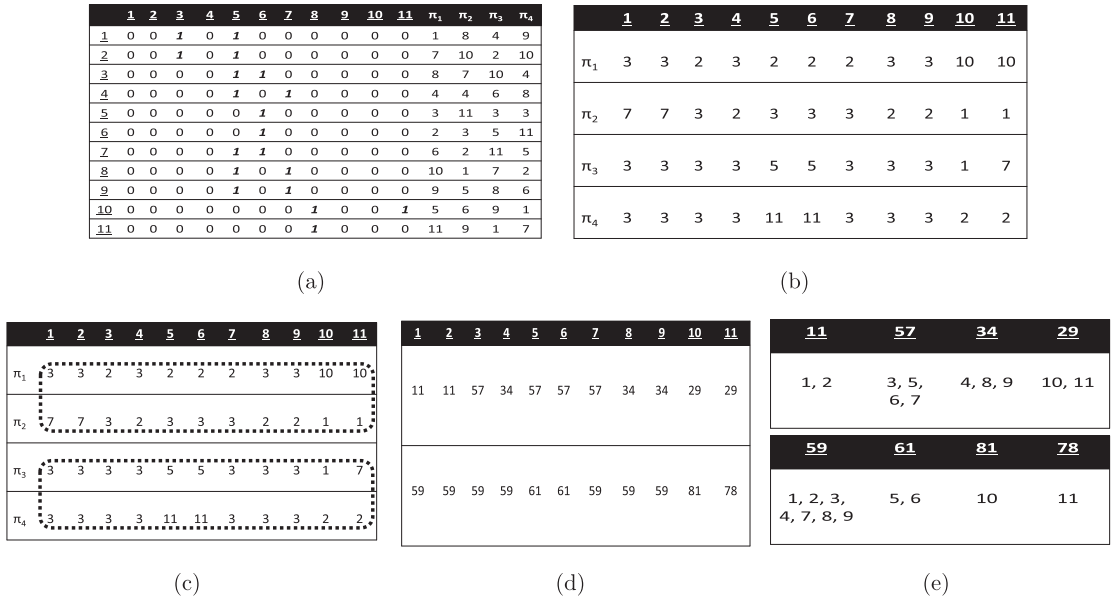
3.2.1. Validating the use of the Jaccard coefficient

Nodes can be declared as similar in a graph if their neighborhood structure is also similar. Particularly, a pair of nodes having many common neighbors are highly similar to each other, such as nodes 1 and 2 from Fig. 1(a) and vice versa. To demonstrate the notion of similarity among member nodes of a graph, the Jaccard coefficient truly reflects this aspect. High similarity among a given set of nodes is necessary so that a resultant super node of a summary graph produces the least number of errors due to non-common edges.

To identify nodes having a high Jaccard coefficient, a simple approach is to perform pairwise similarity computations between every pair of nodes in a graph. This strategy produces the required nodes, but it employs exhaustive searching with quadratic time complexity. To accelerate this process towards at least linear time complexity, we should avoid similarity computations among non-similar nodes and restrict them to among likely similar ones only. For this purpose, we utilize LSH, which approximates the candidate similar nodes with high accuracy. Fortunately, there exists an inherent magical relationship between the Jaccard coefficient and LSH that helps us to achieve this goal. Using LSH, the probability that two items (nodes in the case of a graph) will have the same hash value is exactly equal to their Jaccard coefficient [14].

3.2.2. Locating compressible nodes by using LSH

We now present a step-by-step process to apply LSH on our sample graph that comprises two key operations, i.e., min-hash matrix computation and generation of candidate similar nodes [8].



	1	2	3	4	5	6	7	8	9	10	11	π_1	π_2	π_3	π_4
1	0	0	1	0	1	0	0	0	0	0	0	1	8	4	9
2	0	0	1	0	1	0	0	0	0	0	0	7	10	2	10
3	0	0	0	0	1	1	0	0	0	0	0	8	7	10	4
4	0	0	0	0	1	0	1	0	0	0	0	4	4	6	8
5	0	0	0	0	0	1	0	0	0	0	0	3	11	3	3
6	0	0	0	0	0	1	0	0	0	0	0	2	3	5	11
7	0	0	0	0	1	1	0	0	0	0	0	6	2	11	5
8	0	0	0	0	1	0	1	0	0	0	0	10	1	7	2
9	0	0	0	0	1	0	1	0	0	0	0	9	5	8	6
10	0	0	0	0	0	0	0	1	0	0	1	5	6	9	1
11	0	0	0	0	0	0	0	1	0	0	0	11	9	1	7

(a)

	1	2	3	4	5	6	7	8	9	10	11
π_1	3	3	2	3	2	2	2	3	3	10	10
π_2	7	7	3	2	3	3	3	2	2	1	1
π_3	3	3	3	3	5	5	3	3	3	1	7
π_4	3	3	3	3	11	11	3	3	3	2	2

(b)

1	2	3	4	5	6	7	8	9	10	11
11	11	57	34	57	57	57	34	34	29	29
59	59	59	59	61	61	59	59	59	81	78

(c)

(d)

11	57	34	29
1, 2	3, 5, 6, 7	4, 8, 9	10, 11

59	61	81	78
1, 2, 3, 4, 7, 8, 9	5, 6	10	11

(e)

Fig. 3. Illustration of applying LSH on a graph (a) Adjacency matrix representation of sample graph from Fig. 1a, 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.

The minhash matrix has two main features. First, it can be memory resident, as its size is $k \times n$, where k and n are the total number of hash functions and total nodes in a graph, respectively, and $k \ll n$. Hence, it can be much smaller than the adjacency matrix (of size $n \times n$) of the same graph. Second, the minhash column for a node $v_i \in V$ from G is computed from the neighborhood of v_i, hence the matrix facilitates efficient Jaccard similarities estimation of its members. We demonstrate the minhash matrix computation for our sample graph from Fig. 1(a) using Fig. 3. Fig. 3(a) shows the adjacency matrix of the sample graph along with four hash functions of random permutations of node ids. Similarly, Fig. 3(b) shows the minhash matrix. Let us compute the minhash column of node id 1 using its adjacency list. Using neighbors as indexes for π_1, we find that neighbor id 3 is stored on index 5 and that the index of neighbor id 5 is 10. Because the value 5 is smaller than 10, 3 is chosen as the minimum as the first minhash value. Applying the same procedure for the remaining three hash functions on the neighborhood of node 1, we get 7, 3, and 3 as the remaining minimum values. In this manner, the minhash signature column is obtained for node id 1. The same process is applied for the rest of the nodes to compute the minhash matrix of the entire G.

All the nodes having the exact same minhash columns have almost exactly the same neighborhood. Alternatively, to find nodes having partially similar neighborhoods, known as candidate similar nodes, we divide the matrix into b bands of r rows each, as shown in Fig. 3(c). We then compute hash codes from portions of columns in each band of every node, as illustrated in Fig. 3(d). Finally, each band forms a hash table, and the nodes having the same hash code (in step (d)) are placed in the same bucket. In this manner, all the nodes that exist in the same bucket are termed as candidate similar nodes, and similarity computations are restricted among them only.

3.2.3. Merging compressible nodes

We start creating an S_G as soon as G is indexed using LSH. For each super node of S_G, we randomly select a q from G and retrieve all the nodes that appear with it from every hash table. This retrieved set of nodes is called $SCandN_q$. There are varying neighborhood similarities of q with every $v \in SCandN_q$ since even a single hash code collision declares them as candidate similar. For example, there is only one common neighbor with node 1 as q and node 9. Hence, they appear together in only one bucket in Fig. 3(e). Moreover, the difference in their edge weights is also large. Alternately, node 1 shares all of its neighbors with node 2, hence they appear together in both of the hash tables. Therefore, we extract its subset $SCompN_q$ so that the resulting edge mistakes and edge weight errors are lesser. For this purpose, we sort $SCandN_q$ based on EDR against q and then iteratively prune every candidate node that violates the thresholds of neighborhood similarity and edge weight difference. In this manner, we obtain a $SCompN_q$ that can be compressed into a super node.

It is observed that, although each member of a $SCompN_q$ satisfies both the given thresholds, directly merging the entire set into a super node may still generate large edge mistakes and edge weight errors. Therefore, we iteratively aggregate highly similar nodes first and continue adding the rest of the nodes sequentially into the super node. Using this iterative merging approach, we find that some of the member nodes of a given $SCompN_q$ do not get merged. This happens because

each merge alters the EDR of the rest of the nodes. As a result, the remaining non-compressible nodes are released so that they can be considered further in $SCandN_q$ for any other q. We now demonstrate this fact using Theorem 1.

Theorem 1. *Given a q and its $SCompN_q = \{v_1, ..., v_i, ...v_m\}$, where $EDR(v_i) > EDR(v_{i+1})$. If any $v_i \in SCompN_q$ cannot be merged with the super node s created by q and $\{v_1, ..., v_{i-1}\}$, then v_{i+1} also cannot be merged.*

Proof. Lets consider an $SCompN_q = \{v_1, ...v_4\}$ for q where $EDR(v_1) > EDR(v_2) > EDR(v_3) > EDR(v_4)$. Merging q with v_1 creates a super node s. This reduces EDR with v_2 as now s contains edges of q and v_1; and number of common neighbors of v_1 and v_2 is smaller to that of q and v_1. Merging s with v_2 also adds v_2 as member of s, however, it further reduces EDR of v_3 since overlap of common neighbors becomes very small. This small neighborhood sharing drops EDR of s with v_3 below the threshold, therefore s cannot be merged with v_3. Conclusively s also cannot be merged with v_4 due to low neighborhood sharing of s and v_4 as $EDR(v_2) >> EDR(v_4)$. □

3.3. Computing a summary for a PWGS problem

The objective of a PWGS problem is to create an S_G using highly influenced nodes by v_{inf}. As discussed in Section 2.3, a personalized S_G is more useful for v_{inf} when its every super edge is attached with a higher flow. Therefore, the technical challenge of a PWGS problem is to aggregate nodes having higher edge weights with their neighbors while ignoring those who also share the same neighbors but have lower edge weights with them.

We find that the desired computational model for a PWGS problem differs from that of BWGS in terms of flow maximization. PWGS aims to obtain an S_G containing super nodes that could maximize the flow. Alternately, BWGS creates an S_G based on uniform consideration of neighbors for every $v \in V$, hence it cannot handle flow maximization. An alternative is to create an S_G for v_{inf} using the computation model of BWGS and then perform post-processing to filter out k nodes. However, this process does not guarantee aggregation of nodes having higher edge weights with their neighbors. As a result, each super edge may be comprised of edges with varying edge weights, which not only effects flow maximization but also increases errors due to the differences in edge weights. In this situation, rather than optimizing the post-processing solution, we focus on flow maximization while indexing candidate similar nodes.

Weight Oriented LSH for Flow Maximization, WoLSH. It is observed that edge weights can only be focused until the minhash matrix computation in step (b) of Fig. 3, since the rest of the steps operate on minhash columns to generate candidate similar nodes. Therefore, we restrict our focus on this step to only the meeting of our objective of flow maximization. Here, our goal is to increase the probability of generating minhash values from a subset of neighbors, of every node, having higher edge weights.

Given $v_i \in V$ and its neighborhood N_{v_i}, we find that every $v_l \in N_{v_i}$ has uniform probability to be minimal against any of the hash functions, as shown in Eq. (6).

$$\forall v_j \in N_{v_i}, Pr(min(\pi(N_{v_i})) = v_l) = \frac{1}{|N_{v_i}|} \tag{6}$$

Eq. (6) confirms that minhash computation for v_i does not focus its subset of neighbors having higher edge weight with it. Therefore, we alter the neighborhood representation of each $v \in V$ based on the edge weight of every neighbor. We define this modified neighborhood in Definition 4.

Definition 4 (Weight Adjusted Neighborhood)**.** Given $v \in V$, upper bound (UB) of the normalized edge weights for E of G, and neighborhood of v as $N_v = \{v_1^{w_1}, ..., v_m^{w_m}\}$, where w_1 and w_m respectively show $w(v, v_1)$ and $w(v, v_m)$, the Weight Adjusted Neighborhood (WAN) contains representation r of each neighbor of v equal to its edge weight as $N_v^w = \{r_{v_1}^1, ..., r_{v_1}^{w_1}, ..., r_{v_m}^1, ..., r_{v_m}^{w_m}\}$, where $r_{v_i}^j$ is computed as $\mathrm{Id}(v_i) \times \mathrm{UB} + j$.

WAN adjusts the representation of every neighbor proportional to its edge weight, but every node has a unique representation in a hash function π as shown in Fig. 3(a). Therefore, we also place each node in every hash function, according to the maximum possible edge weight in G. If the edge weights are normalized between a range $1 \ldots z$ and the total number of nodes in G is n, then the representation r of every node is displayed in a sample hash function Π using Eq. (7).

$$\Pi = \left\{r_{v_1}^1, r_{v_1}^2, ..., r_{v_1}^z, ..., r_{v_n}^1, r_{v_n}^2, ..., r_{v_n}^z\right\} \tag{7}$$

Now, applying Π from Eq. (7) on WAN N_v^w of v, we find that a neighbor node with higher edge weights has a greater chance of becoming minimum to produce the minhash value. We prove this fact using Theorem 2.

Theorem 2. *Given $v \in V$ and its weight adjusted neighborhood $N_v^w = \{v_1^{w_1}, ..., v_m^{w_m}\}$, where $w_1 > w_m$ and $|N_v^w| > 1$. Then using a hash function Π of random permutation, we have*

$Pr(min(\Pi(N_v^w)) = v_1) > Pr(min(\Pi(N_v^w)) = v_m)$

Proof. Consider $N_v^w = \{v_1^{w_1}, ..., v_m^{w_m}\}$ of v, where $w_1 > w_m$. Since, v_1 and v_m appear in N_v^w equal to w_1 and w_m times respectively, so $\Pr(min(\Pi(N_v^w))) = v_l = \frac{w_1}{w_1+...+w_m}$ and $\Pr(min(\Pi(N_v^w))) = v_m = \frac{w_m}{w_1+...+w_m}$. Similarly, $\Pr(min(\Pi(N_v^w))) = v_l = \frac{w(v,v_l)}{\sum_{v_j \in N_v^w} w(v,v_j)}$ and $\Pr(min(\Pi(N_v^w))) = v_m = \frac{w(v,v_m)}{\sum_{v_j \in N_v^w} w(v,v_j)}$. As a result, $\Pr(v_1) > \Pr(v_m)$ to generate minhash value for v.

Using Theorem 2, we now show the probability for a pair of nodes v_i and v_j to pick the same neighbor for generation of the same minhash value in Eq. (8).

$$Pr\left(min\left(\Pi\left(N_{v_i}^w\right)\right)\right) = Pr\left(min\left(\Pi\left(N_{v_j}^w\right)\right)\right) = v_l = \frac{w(v_i, v_l)w\left(v_j, v_l\right)}{|N_{v_i}^w \cup N_{v_j}^w|} \tag{8}$$

Eq. (8) displays the probability that nodes v_i and v_j pick the same neighbor to generate the minhash value. We now display their overall probability of generating the same minhash values in Eq. (9).

$$Pr\left(min\left(\Pi\left(N_{v_i}^w\right)\right)\right) = Pr\left(min\left(\Pi\left(N_{v_j}^w\right)\right)\right) = \frac{\sum_{v_k \in \left(N_{v_i}^w \cup N_{v_j}^w\right)} w(v_i, v_k)w\left(v_j, v_k\right)}{|N_{v_i}^w \cup N_{v_j}^w|} \tag{9}$$

Because each super node consists of a set of nodes, the set of nodes $SCompN_q = \{v_i, ..., v_m\}$ obtained by having chosen v_l to generate the same minhash value is listed in Eq. (10). This probability is the same as in [13], which is obtained by generating minhash columns using a certain number of times.[3]

$$Pr\left(min\left(\Pi\left(\forall_{v_i \in SCompN_q} N_{v_i}^w\right)\right) = v_l\right) = \frac{\prod_{i \in SCompN_q} w(v_i, v_l)}{|\bigcup_{i \in SCompN_q} N_{v_i}^w|} \tag{10}$$

Using Theorem 2, it is confirmed that our proposed WoLSH scheme provides higher chances to nodes having higher edge weights towards generation of minhash values. In this manner, the nodes having higher edge weights have greater chances of aggregating with each other to produce super nodes whose super edges have higher Flow Rates. We now prove this fact using Theorem 3 based on the findings of Theorem 2. □

Theorem 3. *Given v and its sorted neighborhood based on their edge weight with v as $N_v = \{v_1, v_2, v_3, ..., v_j\}$, where $w(v, v_1) > w(v, v_2) > w(v, v_3) > ...w(v, v_j)$ and every neighbor satisfies edge weight difference threshold and EDR threshold, then on possible aggregation of v_1, v_2 into s_1 and $v_3, ..., v_j$ into s_2, we have Flow Rate $(e_s(v, s_1)) >$ Flow Rate $(e_s(v, s_2))$.*

Proof. Because the objective of Flow Rate (shown in Eq. (4)) is to create a balance between the sum of edge weights and the size of super nodes, it aims is to avoid the nodes with least contribution towards maximizing total flow. The edge weight of v_1 and v_2 with v is higher than the rest of the nodes, and the size of their resultant super node is also lesser, hence they are aggregated with each other to produce higher flow rate.

Using Theorems 2 and 3, we also confirm that WoLSH correctly identifies the nodes that are merged to have super edges containing higher Flows. □

3.4. Complexity analysis

We now present complexity analysis for the time and memory consumption of our solution. The time consumption is classified into hashing candidate similar nodes and their retrieval, along with aggregation. There is a difference in the computation model for both of the summarization problems, so we highlight the differences for clear understanding.

We use b hash tables and k hash functions to generate candidate similar nodes from the graph. Using each function, we find the minhash value for the node under consideration. Upon repeating this action k times, we obtain the minhash signature column of length k for the given node. Performing this action for the entire graph results in the minhash matrix. In the case of the PWGS problem, the minhash signature generation is slightly modified to consider the existence of the edge weight. In this case, for a given hash function and a node under consideration, we compute its minhash value equal to the number of times represented by its edge weight with the corresponding neighbor to give higher chances to those neighbors having higher edge weights in the generation of the minhash values. In this manner, the minhash computation is performed equal to the sum of the edge weights of the entire neighborhood for each query node. Finally, when the entire minhash matrix is generated, we divide it into b bands containing equal-sized portions of the columns to obtain the partially similar nodes. This step has been illustrated in Fig. 3(c) for clear understanding and is then followed by the steps to place candidate similar nodes together in buckets of b hash tables. In this manner, the total time consumed for candidate similar node generation is $O(bkn + Sum_{EW})$, where Sum_{EW} is the summation of the edge weights from every edge of the graph.

For summarization, we retrieve candidate similar nodes $SCandN_q$ for each q in $O(b)$ time from all hash tables. The retrieval time is b since there are b hash tables, where each contains a representation of every node in its buckets. The size of the retrieved set of candidate nodes $SCandN_q$ is m, where $1 \leq m \ll n$. This set is sorted, based on EDR, in $O(m \log m)$ time to bring highly similar candidate nodes to the top of the list. The same process is repeated for every q, hence the total repetitions are $O(Q^*(m \log m))$. Conclusively, the overall time complexity for the BWGS problem is $O(bkn + Q * (m \log m))$, and that of the PWGS problem is $O(bkn + Sum_{EW} + Q * (m \log m))$, where Q represents the total number of query nodes. It must be noted that the numbers of iterations to generate the summary graph are Q and $Q \ll n$ because each iteration aggregates multiple nodes. We consider the super node creation time to be constant.

[3] We discuss and perform comparisons against [13] in the experimental section.

The space complexity for both the BWGS and PWGS problems depends upon that of the index structure obtained by applying LSH. There are a total of b hash tables, and each node is stored b number of times, hence the space complexity is $O(bn)$.

3.5. Analyzing the completeness of the algorithm

We analyze the completeness of our algorithm by showing that our algorithm visits every node in the graph and that no node is left unvisited. For this purpose, we prove that all compressible nodes in a graph are compressed and that no such node remains uncompressed.

Considering Algorithm 1, we find that it randomly picks a query node q in every iteration (line 1 of Algorithm 1), so each such chosen node is marked as visited in this manner. Secondly, member nodes of the obtained set of nodes for q that are compressed into a super node are all marked as visited as well. When all the nodes are visited in this manner, the algorithm terminates. Below, we provide a theorem to show that all compressible nodes are compressed and that no such node is left uncompressed. We prove this statement by showing that each query node q does find a set of compressible nodes, if there is any, and creates a super node.

Algorithm 1 Set-based merging.

Require: Summary Graph S_G, EDR t_N, Edge Weight Difference t_w, Hash Tables HTs
Ensure: Summary S_G

1: **while** (*unvisited nodes in* S_G) **do**
2: Randomly select a query node q
3: $SCandN_q$ = Retrieve candidate nodes from HTs, appearing with q
4: $SCompN_q$ = Sort $SCandN_q$ based on EDR and prune the nodes which violate t_N and t_w
5: **for each** $v \in SCompN_q$ **do**
6: **if** $(EDR(u,v) > t_N$ and $(w_u - w_v) \leq t_w)$ **then**
7: $s = u \cup v$ // u=q in 1st iteration otherwise u=s
8: $V_s = V_s$ - $\{u, v\} \cup s$
9: remove u, v from S_G
10: **end if**
11: **end for**
12: **end while**
13: **Return** S_G

Theorem 4. *Given a query node $q \in V$ from G, it is compressible with any other node $v \in V$ from G, if they satisfy given thresholds of EDR and Edge Weight Difference.*

Proof. Given a query node $q_i \in V$ and any other node $v_j \in V$ from G, we find that the Jaccard similarity $J.Sim$ of q_i and v_j can be approximated as

$$Pr\big(min(\pi(N(q_i))) = min\big(\pi\big(N\big(v_j\big)\big)\big)\big) = \frac{\big(N(q_i) \cap N(v_j)\big)}{\big(N(q_i) \cup N(v_j)\big)} = J.Sim\big(q_i, v_j\big) \tag{11}$$

Here, $J.Sim(q_i, v_j)$ approaches its maximum value, i.e., 1, if their minhash values for k number of hash functions are the same, i.e., $\{minhash^1_{q_i} = minhash^1_{v_j}, minhash^2_{q_i} = minhash^2_{v_j}, ..., minhash^k_{q_i} = minhash^k_{v_j}\}$ or otherwise. Initially, q_i and v_j are declared as candidate similar using any fixed sized subset of length m, where $m \leq k$, of minhash values for q_i and v_j, if there occurs a hash code collision by applying an arbitrarily chosen hash function $f()$ i.e., $f(min^1_{q_i}, ..., min^m_{q_i}) = f(min^1_{v_j}, ..., min^m_{v_j})$, then q_i and v_j are placed together in the same bucket in a hash table. Similarly, if their hash codes collide b times, they are placed together in the same bucket of all b hash tables. Conclusively, v_j is merged with q_i into a super node upon retrieval if they also satisfy the constraint of edge weight difference. In this manner, both q_i and v_j are marked as visited nodes. Similarly, the same holds true for a set of l nodes $\{v_1, v_2, ..., v_l\}$ with each having b hash code collisions with q_i. Alternately, for any $v_i \in V$ having no or p hash code collision(s) with q_i, where $p < b$ and $EDR(q_i, v_i) < t_n$, v_i is left unmerged and remains as unvisited unless it is otherwise selected as a query node q_y or merged with any other query node q_z upon retrieval. □

4. Experimental evaluation

4.1. Experimental setup and datasets

We used an Intel Core i7-3960X 3.30 GHz processor and 36 GB of main memory, running 64-bit Windows 7 enterprise edition, for experiments. All the algorithms were implemented in Java and tested on three publicly available real-world

graphs (Astrophysics [15] with $V = 16{,}046$ and $E = 121{,}251$, Citation [16] with $V = 453{,}460$ and $E = 1{,}921{,}301$, and DBLP[4] with $V = 205{,}264$ and $E = 1{,}048{,}785$) and a synthetically generated dataset using a well-known graph generator with $V = 100{,}000$ and $E = 726{,}767$[5]. These are weighted graphs whose edge weights are normalized between 1 and 100.

4.2. Evaluation criteria for the BWGS problem

We evaluate our proposed algorithm for the BWGS problem using execution time and Root Mean Square Error (RMSE) for summary distance at different compression ratios. We skip the results of edge mistake analysis due to the shortage of space. We perform experiments against the state-of-the-art Randomized Semi-Greedy algorithm from [6]. We refer to our algorithm as Set-based Approach for Graph Summarization (SAGS) in continuation of our previous work [8]. It must be noted that we omit the parameter value Edge Weight Difference Threshold for generation of $SC_{omp}N_q$ (for Definition 3 and Algorithms 1 and 2) for experiments and rely on the natural edge weight similarity existing in each set of nodes.

Algorithm 2 Hashing and summarization.

Require: Graph G, Bands b, Hash Functions k, EDR Threshold t_N, Edge Weight Difference Threshold t_w, Degree Threshold t_{deg}
Ensure: Summary S_G

1: $S_G(V_s, E_s, w_s) = G(V, E, w)$
2: Create b Hash Tables HTs to store candidate similar nodes
3: **for each** $v_i \in S_G$ **do**
4: **if** (Degree$(v_i) < t_{deg}$) **then**
5: Declare 2-hop neighborhood of v_i as $SC_{and}N$ and filter best compressible node for aggregation.
6: **else**
7: **for** $cntr = 1$ *to* k **do**
//Execute Lines 8 and 9 for PWGS only
8: **for each** $v_j \in N_{v_i}$ **do**
9: N_{v_i} = Set Representations of v_j based on $w(v_i, v_j)$
10: $min_{cntr} = \min(\pi_{cntr}(N_{v_i}))$// Minhash value
11: $MinCol_{v_i} = MinCol_{v_i} \cup min_{cntr}$// Minhash Column
12: **end for**
13: **end for**
14: **end if**
15: Divide $MinCol_{v_i}$ into b bands to generate $\{HC_1, ..., HC_b\}$ hashes codes
16: **for** $cntr = 1$ *to* b **do**
17: Store v_i in bucket with id HC_{cntr} for HT_{cntr}
18: **end for**
19: **end for**
20: S_G = Set-based Merging(S_G, t_N, t_w, HT) // Algorithm 1
21: **Return** S_G

4.3. Evaluation criteria for the PWGS problem

We provide empirical evaluation of WoLSH against the hashing scheme of TopGC [13]. We also compare with the LSH indexing technique used for the BWGS problem, referred to as BasicLSH, where every neighbor for each node is uniformly considered for hashing. We provide comparisons for execution time evaluation, flow rate analysis, and RMSE for summary distance based on edge weight differences. Moreover, we also provide detailed comparisons of various features of the summary graphs for effectiveness analysis of WoLSH.

4.4. Setting LSH parameters

Utilizing LSH with different combinations of its parameters has a direct effect on its performance. These parameters (number of hash tables and hash functions) are used to approximate the Jaccard Similarity threshold using the formula $(\frac{1}{b})^{\frac{1}{r}}$ [14], where b is the number of bands to generate b hash tables and r is the number of rows ($|HashFunctions|/|HashTables|$) in each band. We use 10 bands and 30 hash functions for the BWGS problem, which generates a similarity threshold of 0.46. For the PWGS problem, we set a lower threshold of 0.2 by using 25 bands and 50 hash functions. A lower threshold is set in this case, as the main emphasis is on increasing the flow rate among super nodes in the summary graph. Hence, we do not want to miss merging similar nodes having higher edge weights with their common neighborhood.

[4] http://dblp.uni-trier.de/.
[5] https://sites.google.com/site/santofortunato/inthepress2/ Last accessed on 10/31/2016.

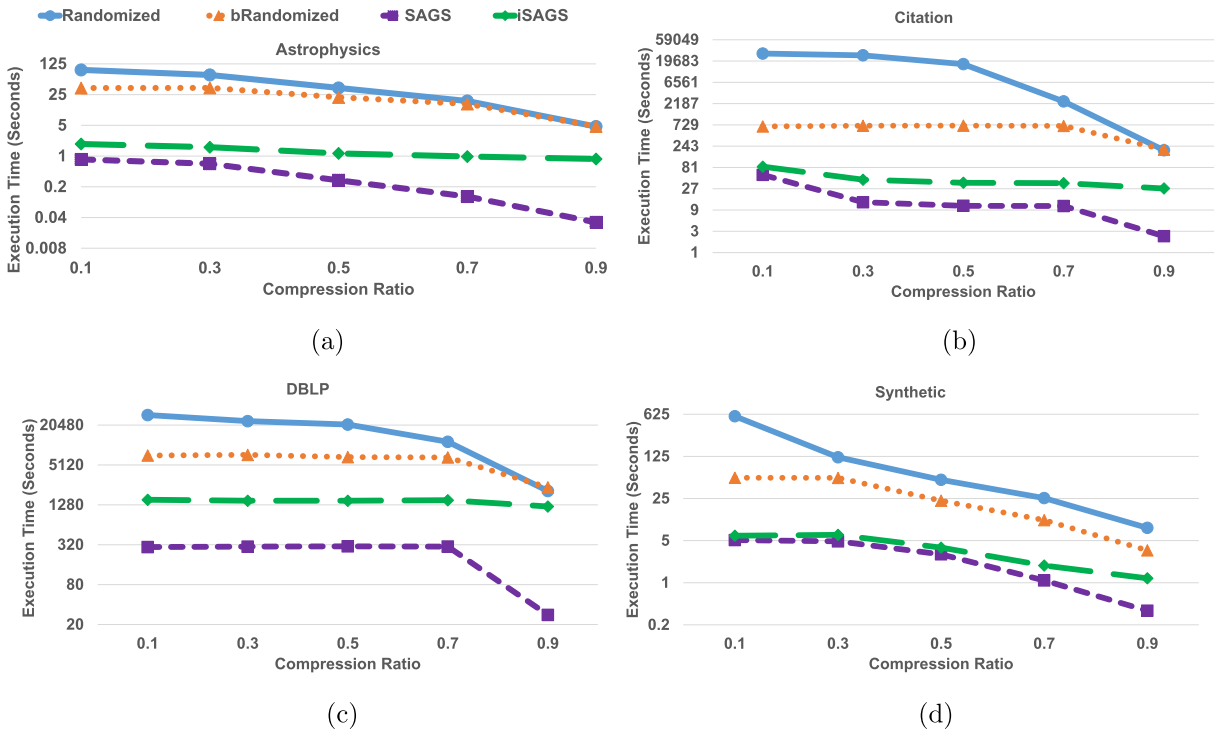


Fig. 4. Execution time comparisons at various compression ratio (Compression Ratio is defined in Eq. (3)).

4.5. Evaluations for the BWGS problem

4.5.1. Execution time analysis

We show execution time comparisons among algorithms in Fig. 4. Execution time by iSAGS demonstrates compression as well as indexing time for candidate similar node generation, whereas SAGS only shows the compression time. bRandomized is a variation of Randomized where we bound the number of super nodes in a summary graph equal to those by SAGS. In this manner, execution of bRandomized stops when the number of super nodes of the summary graph becomes equal to those by SAGS.

The results demonstrate that SAGS provides scalable performance in every dataset. Randomized has to explore the entire 2-hop neighborhood of each query node to search for a node that provides minimum difference in edge weight against it. The process is time-consuming, as each exhaustive search, which is proportional to the size of the 2-hop neighborhood of each query node, results in the merger of only a pair of nodes. Alternately, SAGS is independent of all such intensive comparisons. The prime reason is that it directly locates the nodes having similar neighborhoods using hash-code matching only. In this manner, no similarity-check computations are performed among dissimilar or least-similar nodes. Finally, a filtered set of nodes is aggregated into a super node. Theorem 1 is utilized by default to prune the least similar nodes when a large-sized set is retrieved against any query node.

4.5.2. Analyzing graph compression

The objective of Randomized is to produce a compact summary graph whose decompressed version has minimum distance due to edge weights from the input graph. Every iteration of Randomized opts to find a pair of nodes having minimum difference in edge weights only. In this manner, the entire focus is on minimization of error due to edge weights. This process continues until the required compression ratio is obtained. Hence, there is no focus on preserving the structural properties of a graph in its compressed version, and therefore our proposed approach creates a summary with lesser number of edge mistakes.

Randomized uses the termination condition for compression as the compression ratio only. Because there is no inclination towards considering structural similarities among any pair of nodes to be compressed, a lower ratio such as 0.1 outputs a highly compact graph with a minimum number of super nodes and super edges. SAGS, alternately, considers neighborhood similarities among nodes as the prime objective for compression. Therefore, structural properties remain intact in its generated summary graph. This is the reason that SAGS cannot compress as many nodes as Randomized, as shown in Fig. 5. We find that the number of super nodes produced by Randomized is much lower than that by SAGS. To rigorously evaluate

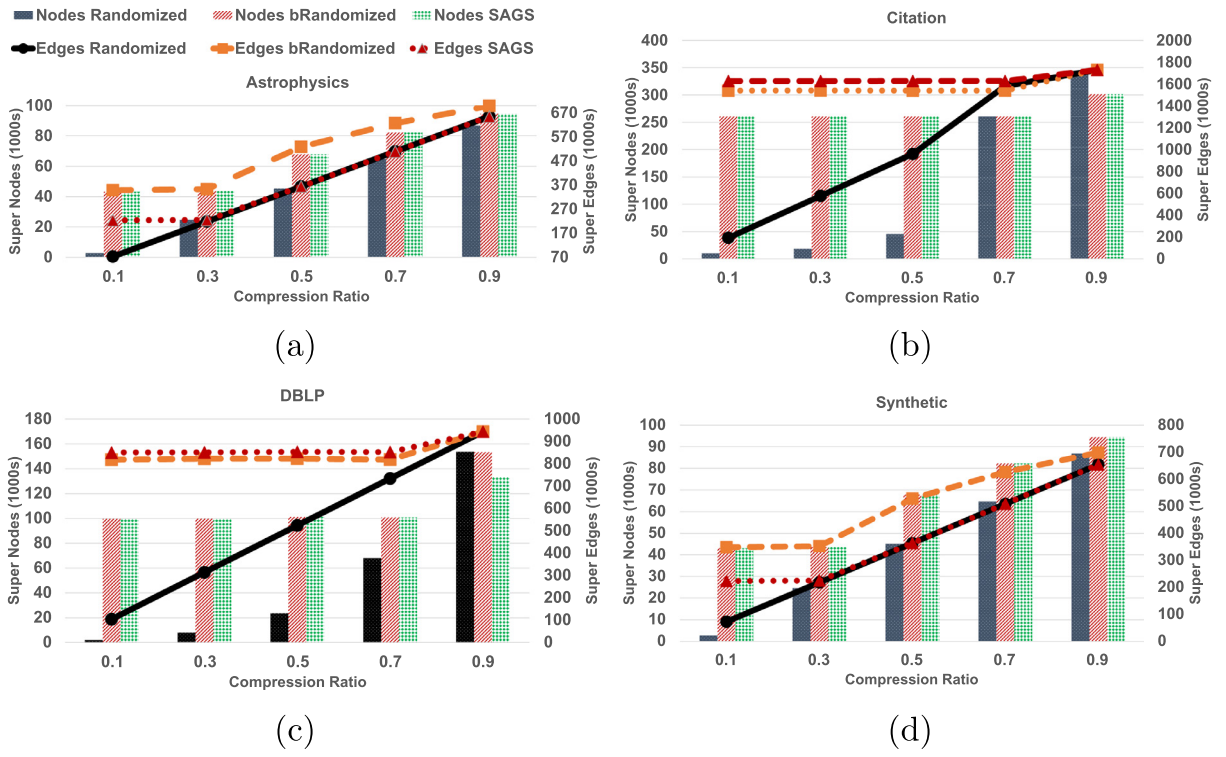


Fig. 5. Summary graph size at different compression ratio (Compression Ratio is defined in Eq. (3)).

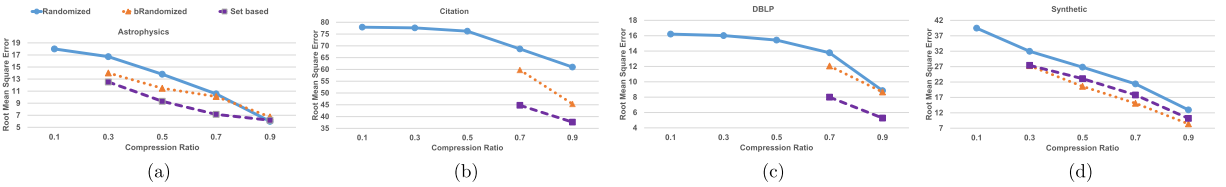


Fig. 6. RMSE (Root Mean Square Error) due to difference of edge weights, at different compression ratio (Compression Ratio is defined in Eq. (3)).

the behavior of SAGS, we modify Randomized to stop compression when the number of super nodes in the summary is equal to that by SAGS. The results of bRandomized demonstrate this variant version of Randomized.

It must be noted that, for each aggregation, we decide whether to generate edge mistakes or to create super edges by using the Minimum Description Length (MDL) rule as utilized by [4]. Therefore, Randomized, its variant bRandomized, and SAGS intend to output a compact summary graph with the least number of edge mistakes.

4.5.3. RMSE evaluation

Fig. 6 shows the difference in terms of RMSE among the given algorithms. We find that SAGS presents better results against its competitors in all of the comparisons. The main emphasis of SAGS is to ensure higher neighborhood similarities among the set of nodes for compression. Therefore, we rely on intrinsic edge weight similarities available in a graph for each set of nodes to compress. We observe that there is little variation in edge weight differences for each obtained set of nodes in a real-world graph with higher neighborhood similarities. This can be illustrated by the fact that nodes having many common neighbors also have strong communication among themselves. Consequently, there is a natural homogeneity in edge weights as well. Therefore, we find that RMSE by SAGS is less than that by bRandomized, although both have equal number of super nodes in their respective summary graphs. Lesser RMSE by SAGS in comparison with Randomized is obvious since SAGS compresses a lesser number of nodes than Randomized.

The distribution of edge weight in the synthetic graph, alternately, is highly diverse due to the random generation of edge weights. Therefore, we find that SAGS shows competitive performance in this dataset.

4.5.4. Effect of varying EDR threshold on summary graph

In this section, we provide an analysis of the effect of varying the EDR threshold on the resultant summary graph. We evaluate this effect in terms of the number of edge mistakes generated by the summary graph.

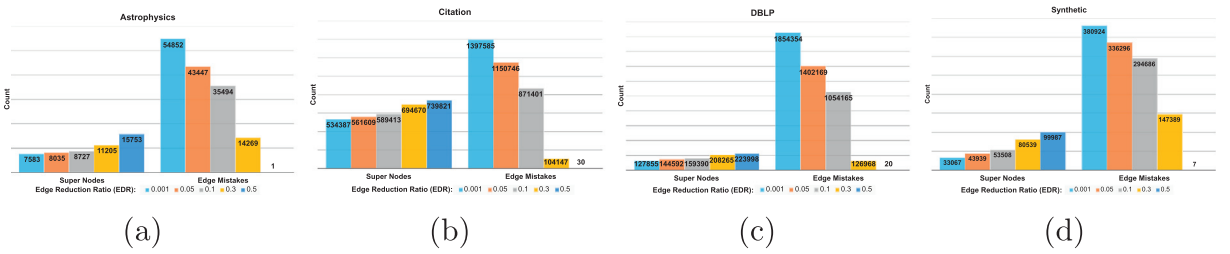


Fig. 7. Analyzing effect of varying EDR (Edge reduction ratio) thresholds on summary graph, where EDR is defined in Eq. (5).

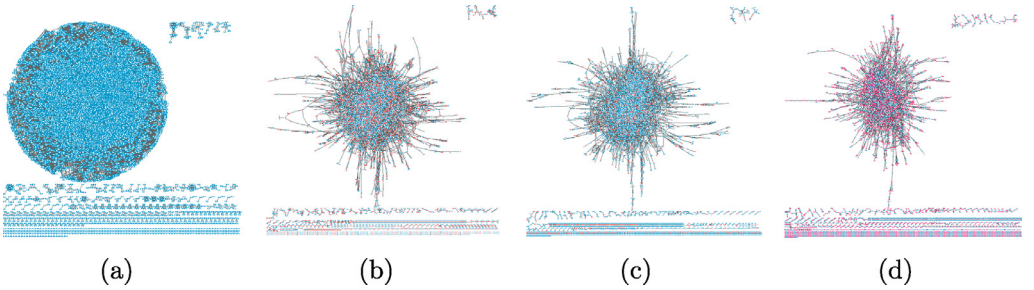


Fig. 8. Summary graph analysis through visualization of Astrophysics dataset. Blue and red colored nodes in summary graph show non-compressed and compressed nodes respectively. (a) Input graph. (b) Summary graph at EDR = 0.3 (c) Summary graph at EDR = 0.15 (d) Summary graph at EDR = 0.01.

We recall our explanation regarding EDR from Section 3.1 that a higher EDR threshold allows a merger of nodes having higher neighborhood similarity only. A lower number of edge mistakes is anticipated from the compression of such a type of nodes. The results in Fig. 7 validate this very aspect in the resultant summary graph. In each of the experiments, we find that the number of super nodes is much lower at a lower EDR threshold and that it increases with an increase of the EDR threshold. Edge mistakes, alternately, decrease in contrast to the aforementioned measures.

It should be noted that the EDR and compression ratio *cr* have no dependency on each other. *cr* controls the overall compression rate of the entire summary graph, i.e., to what extent a graph should be summarized. EDR, alternately, controls the merger of nodes based on their structural similarity with each other. If a higher value, such as 0.5, is set for EDR and a lower value, such as 0.1, is set for *cr*, it is not necessary that the obtained summary graph be compressed according to the requirements of *cr*. In this case, as soon as the traversal of the entire graph ends, the summarization process is stopped as well.

4.5.5. Visualization of the summary graph for the BWGS problem

In this section, we present the visualization of the summary graph to illustrate how the structure of its underlying graph looks after compression. We create the summary graph using varying threshold values of EDR (Eq. (5)) and visualize them in Fig. 8. We perform this experiment using the Astrophysics dataset only since the rest of the graphs are much larger in size, so their visualizations incorporate greater visual clutter.

The figures demonstrate the fact that, with a decrease in EDR threshold, the summary graph gets more compressed. At lower EDR threshold, nodes having lesser neighborhood similarities are merged with each other, hence the resultant summary graph is very compact. We find that there exist few disconnected regions in the input graph, such as the region in the top-right corner and many appearing in the bottom of Fig. 8(a). We traverse every node in the graph, hence every node in all such regions does get a chance of merging. Therefore, whether an input graph contains scattered or dispersed nodes, they are all compressed into super nodes provided they meet the required similarity constraints.

4.6. Comparisons for the PWGS problem

For evaluations of the PWGS problem, we choose a highly cited paper [17] as input from the citation network.

4.6.1. Execution time and RMSE analysis

We present performance comparisons of WoLSH against TopGC and BasicLSH in Fig. 9. The figure illustrates the results using three flavors of TopGC, generated at different values for its parameter *trials*. TopGC proposes a modification in LSH to generate the minhash signature column for each node using its subset of neighbors having higher edge weights. For this purpose, they generate minhash columns for each node *trial* number of times and randomly select a threshold value for each *trial* to filter out the subset of neighbors whose edge weights are higher than it. BasicLSH, alternately, uniformly considers every neighbor of a given node and hence does not consider the existence of edge weights. In contrast, our

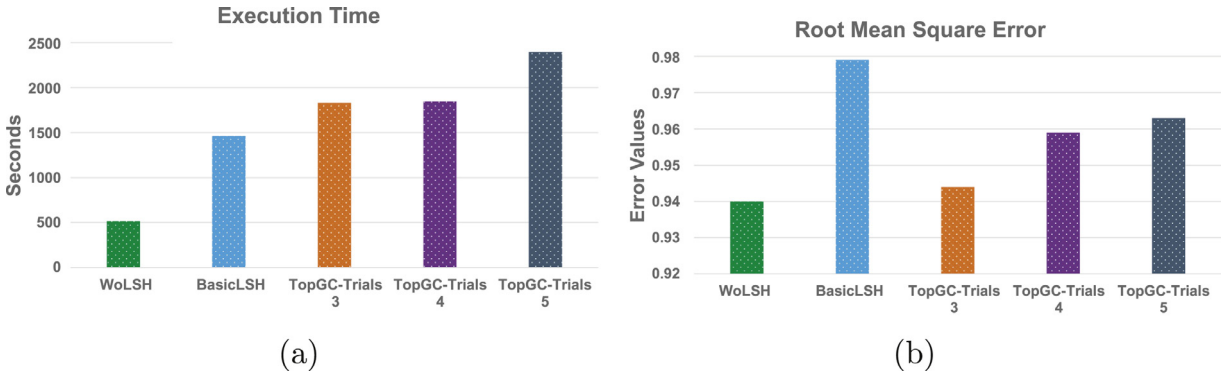


Fig. 9. Performance analysis of WoLSH. (a) Execution time (b) Root Mean Square Error (RMSE) due to difference of edge weights.

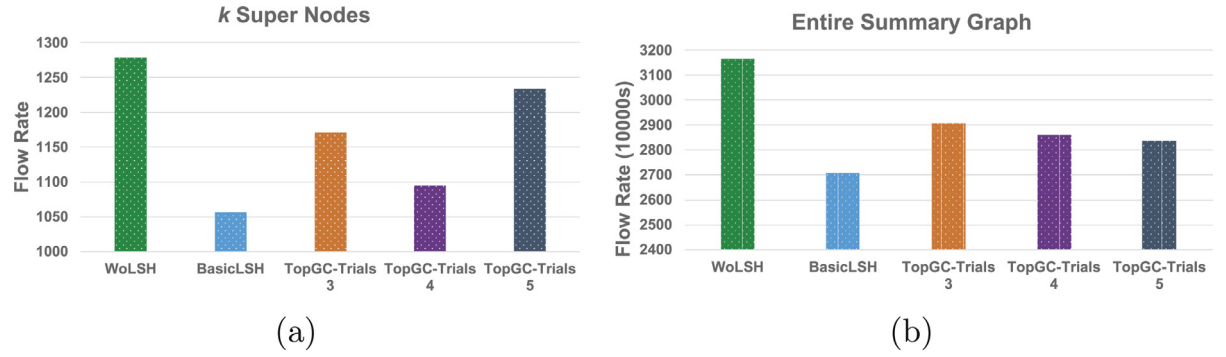


Fig. 10. Effectiveness of WoLSH for summary graph using flow rate. (a) Collective flow rate for k nodes Summary, where k is 23 (b) Collective flow rate for entire summary graph.

proposed hashing scheme for LSH is non-parametric, as we make use of edge weights to prune the subset of neighbors. All the low-edge-weighted neighbors of each node in WoLSH are least likely to generate minhash signatures, hence we automatically obtain a weighted neighborhood of each node.

We observe that WoLSH exhibits better performance in both evaluations, as shown in Fig. 9, against TopGC and BasicLSH. The difference in execution time between WoLSH and TopGC, for various values of its parameter *Trials*, is at least 3 times. This happens as the number of candidate similar nodes in TopGC increases proportionally with respect to the value of *Trials*. Therefore, TopGC consumes significant time in filtering out the required set of nodes for compression. The execution time of BasicLSH is also higher than that of WoLSH since we hash every node from its minor subset of nodes only. Hence, there are lesser number of hash code collisions, and few candidate similar nodes exist in the buckets of every hash table.

The RMSE comparison in Fig. 9(b) also demonstrates better results by WoLSH. A higher value of RMSE by BasicLSH shows that it does not consider the existence of edge weights. Hence, nodes having similar neighborhoods but varying edge weights are aggregated into super nodes. Alternately, generating minhash columns by filtering the subset of neighbors in both WoLSH and TopGC results in a summary graph having a lesser difference of RMSE. Interestingly, RMSE by TopGC is higher than WoLSH although it avails more chances to filter the required set of nodes for compression. We understand that the randomly generated threshold value for each trial is the main cause of this performance degradation. A random threshold selection for high-edge-weighted neighbors does not ensure the selection of the desired neighbors for minhash generation. Any random value can be generated in TopGC for neighborhood filtering, which drastically affects the performance. For instance, a lower threshold may choose almost the entire neighborhood, resulting in no pruning. Similarly, a very high threshold may only pick very few neighbors, resulting in no hash code collisions. Because WoLSH uses intrinsic properties of the graph by purely relying on already available information of edge weights, it provides better results in all cases.

4.6.2. Effectiveness of WoLSH

We now evaluate the effectiveness of our proposed hashing scheme against those by TopGC and BasicLSH. We provide a variety of results in Figs. 10 and 11. We show the results both for a summary graph having only k nodes and for the entire summary graph. Here, we set the value of *trial* to be 5, which is used by the authors in [13]. We observe that WoLSH consistently exhibits better results in every comparison. It exhibits maximized flow rate for both k nodes and the complete summary graph. Because TopGC suffers from random threshold generation to filter the weighted neighborhood, it is unable to search and utilize all those high-edge-weighted neighbors which are easily located by WoLSH. Minhash column generation from a mixture of neighbors of each node having varying edges weights results in super nodes having

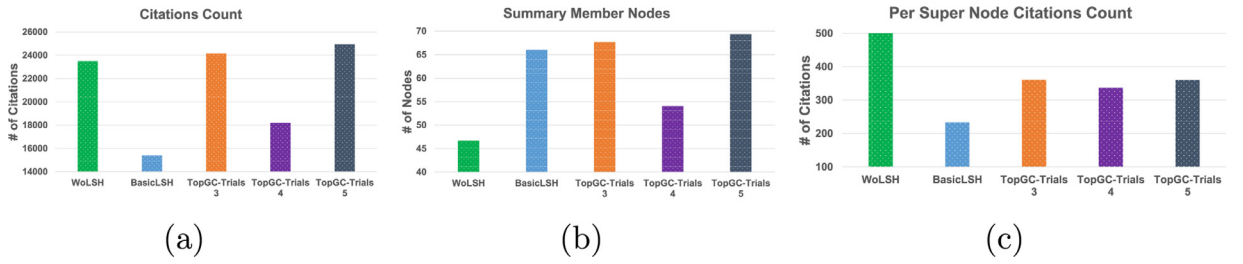


Fig. 11. Effectiveness of WoLSH for k nodes-based summary graph for citation count analysis. (a) Accumulated citations count (b) Total number of constituent/member nodes (c) Average citation count for each super node.

lower flow rate against their corresponding neighbors. This claim is particularly evident from results by BasicLSH, which are improved slightly by TopGC in Fig. 10. Citation count analysis in Fig. 11 also shows better performance by WoLSH. These results are from the summary graph having only k nodes against a given influential node. We find that WoLSH successfully aggregates all the nodes having significantly higher edge weights against their neighbors. Therefore, the total number of member nodes, Fig. 11 (b), is much smaller than that by its competitors, but their per node (Fig. 11(c)) as well as entire citation count (Fig. 11(a)) is still much higher. These results confirm the superior effectiveness of the WoLSH scheme.

5. Related literature review

In this section, we review existing works related to both key components of our work. For each of the sections, we first present non-technical details so that non-expert readers can easily understand the details.

Background for graph summarization

Automated data manipulation and its analysis have remained an important task for the last few decades. Buying an item from a store, for instance, is a database transaction that triggers many operations, such as inventory updates, orders and sales made, and so on. The same holds true for other types of relational data and also for any graph processing system as well. However, the major difference between relational and graph data is that the graph data holds relationships among its entities. For example, consider an order processing system; the relational database system stores the number of orders made, the stock of each item, and the amount earned via sales. Alternately, a graph processing system can show relationships in terms of which customers buy what types of items. Using this buying behavior, we can group customers based on various demographic factors. Therefore, we find that a graph holds actual relationships created using the necessities of life or a given phenomenon. Hence, graph data is more meaningful and semantically rich, resultantly leading towards interesting analysis. However, mining such useful data is difficult and time-consuming when the size of the underlying graph is too large. Therefore, it is necessary to create a compact summary so that any graph-mining algorithm can operate in a short period of time.

Review of existing graph summarization strategies

We now present a review of existing graph summarization approaches. Since the initial work for graph summarization by Saket et al. [4] and Tian et al. [5], there have been many studies on creating a summary of a large graph [6,7,18]. Saket et al. [4] compress nodes having similar neighborhood structures into super nodes and their corresponding edges into super edges. In contrast, Tian et al. [5] summarize nodes with the same attributes and the same neighborhood into super nodes and super edges. However, they relax the restriction of the same neighborhood for each super node due to the difficulty of the problem. Khan et al. present a unified approach in [8,19] that is a lesser constrained version of the problem [5]. Their approach aims to create a summary of a graph where each super node contains a set of nodes with similar attributes and similar neighborhood structure. Recently, a scalable pattern mining-based summarization approach [18] has been presented that iteratively identifies a dense bipartite subgraph for compression. These approaches create a concise summary graph but do not identify subgraphs of different structures. For this purpose, Shah et al. [7] utilize compression to highlight different shapes, such as star, chain, clique, and bi-partite subgraphs, in a large dynamic graph. As the objective of a compressed graph is to accelerate the mining process, Maccioni and Abadi [20] present a graph compression algorithm to accelerate the pattern matching process. In continuation of the virtual node idea from [18], the authors coin the notion of a compressor node to de-densify the dense regions in a graph. For this purpose, edges from every set of high-degree nodes are reduced with their exact common neighborhood by adding a new compressor node in the graph. In this manner, all the dense regions in the graph are sparsified to increase the scalability of the pattern matching process.

We observe that the above approaches provide useful graph summarization solutions but do not consider weights attached to edges. In this regard, Toivonen et al. [6] present compression of a weighted graph. The authors utilize the pairwise super node creation technique from [4] to create a summary graph. Each pair of compressible nodes provides better

compression than the rest of the pairs, but this technique is not applicable to a large graph. Therefore, we present a faster graph compression solution (BWGS problem) for a weighted graph. To compress a dynamic graph, Liu Wei et al. [3] propose a lossy compression scheme based on 3-dimensional tensor graph modeling. This modeling technique ($V \times V \times Time$) is adopted to incorporate changing edge weight information. Further, utilizing dynamicity, Qu Qiang et al. [2] summarize a graph to support diffusion of information. All of these approaches provide a concise view of the underlying graph, but they do not consider the preference of a certain influential node.

Lei Shi et al. [1] provide a flow-based summarization technique. They create a personalized summary graph for an influential node given as input. The authors introduce a new direction for graph summarization, but their non-negative matrix factorization-based solution is inefficient when applied on a large graph. Furthermore, their solution is aligned with a kernel k-means algorithm for graph clustering. We observe that, the larger the size of a cluster, the greater the number of non-shared neighbors among their member nodes, as observed in [19]. Moreover, large-sized clusters also introduce greater numbers of mistakes due to the differences in edge weights. The authors also present an extension of their work in [21] to show a connection between optimal summarization and eigenvector centrality of nodes in a graph.

All of the aforementioned strategies fall under the umbrella of aggregation-based summarization, where the result is a compact summary graph. Such a summary is directly made available for various types of analysis and visualizations. Compression-based methods, alternately, strive to obtain a succinct representation of a graph for low storage space requirements. For this purpose, they aim to find an intrinsic ordering of nodes available in a graph so that nodes having similar structural properties can be compressed [22,23] and stored in a compact data structure. Since the pioneering work by Boldi and Vigna [22], a number of improvements and extensions have been made in this line of work [24]. All such studies do not focus on node aggregation but rather on compressing the edges in the network.

Background of LSH

Finding the similarity (or distance) between data items is a primitive data processing task. An illustrative real-world application, apart from many others, can be similarities among residents of an area based on their demographic characteristics. Normally, attributes or dimensions in the aforementioned type of example are lesser in number. Therefore, the complete data can be processed in a short amount of time. However, domains such as geographic, spatial-temporal, and biological data and so on have 100s of dimensions. Similarity computations of every entity with every other one is truly a time-consuming job, especially when the numbers of entities are very large as well. In this situation, it is better to restrict similarity computations among likely similar ones and conserve computation time among all those entities that are absolutely not similar. The computation model of LSH is based on this very idea and relies on the collision of hash codes from entities under consideration. In this manner, similarity computations are performed only among those entities whose hash codes are the same. Thus, all the non-similar entities have different hash codes and are not compared at all. Based on this remarkable success, LSH has been extensively used in a variety of domains [25]. In this paper, we benefit from LSH for faster similarity computations among nodes in a large graph.

Fast similarity search by using LSH in a graph

Because LSH is an approximate technique for identifying similar objects in high-dimensional spaces [12], its effectiveness has been utilized to identify similar nodes in a graph by various researchers. The authors of [18] utilize min-wise hashing [26] to create a memory-resident minhash matrix. This matrix probabilistically identifies subgraphs with similar neighborhood structures. We provide a set-based approach for summarization of a non-attributed as well as multi-attributed graph in our previous work [8]. This approach adopts LSH to approximate sets of similar nodes in a graph for compression, hence we avoid explicit similarity computations to provide scalability.

Macropol and Singh utilize LSH to accelerate the graph clustering process [13]. Their objective is to efficiently locate clusters from a large graph, where each contains nodes having high edge weights with their neighbors. Because the adoption of LSH for a graph cannot consider weights attached to edges, they provide a modified hashing scheme for this purpose. This scheme generates multiple minhash signatures for each node using a probabilistic threshold. We understand that this LSH scheme [13] can be utilized for our PWGS problem. However, the correct selection of parameters is a well-known problem because it has a significant effect on the performance of any algorithm. Hence, we propose a non-parametric LSH scheme in this paper.

6. Conclusion

This paper presents efficient solutions for the summarization of a large weighted graph. Compression of a weighted graph is useful for accelerating the efficiency of various graph mining algorithms. Existing work utilizes a pair-wise node aggregation strategy that is inefficient for compressing a large weighted graph. To accelerate this process, we present a set-based summarization method that directly locates dense communities of nodes and aggregates them in the form of sets. We avoid an exhaustive search for each set of nodes by using LSH, which is a fast similarity-search algorithm. LSH adoption boosts the summarization process, but its hashing scheme based on the neighborhood similarity of nodes cannot consider

edge weight information. Therefore, we propose a non-parametric hashing scheme for LSH to consider the weighted neighborhood for every node. Rigorous evaluation of our proposals against state-of-the-art solutions using real-world datasets provides superior results. We understand that the combination of the set-based node aggregation strategy and LSH is suitable for efficiently creating a summary of any large graph. Moreover, a thorough investigation of the hashing scheme of LSH can further be altered for various types of similarity-search problems in graphs. Therefore, we intend to extend this very combination towards compression of a weighted streaming graph as our future work.

Acknowledgment

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

References

[1] L. Shi, H. Tong, J. Tang, C. Lin, Vegas: Visual influence graph summarization on citation networks, Knowl. Data Eng. IEEE Trans. 27 (2015) 3417–3431.
[2] Q. Qu, S. Liu, C.S. Jensen, F. Zhu, C. Faloutsos, Interestingness-driven diffusion process summarization in dynamic networks, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2014, pp. 597–613.
[3] W. Liu, A. Kan, J. Chan, J. Bailey, C. Leckie, J. Pei, R. Kotagiri, On compressing weighted time-evolving graphs, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ACM, 2012, pp. 2319–2322.
[4] S. Navlakha, R. Rastogi, N. Shrivastava, Graph summarization with bounded error, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, 2008, pp. 419–432.
[5] Y. Tian, R.A. Hankins, J.M. Patel, Efficient aggregation for graph summarization, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, 2008, pp. 567–580.
[6] H. Toivonen, F. Zhou, A. Hartikainen, A. Hinkka, Compression of weighted graphs, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, pp. 965–973.
[7] N. Shah, D. Koutra, T. Zou, B. Gallagher, C. Faloutsos, Timecrunch: interpretable dynamic graph summarization, in: Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1055–1064.
[8] 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, IEEE, 2015, pp. 255–259.
[9] K. LeFevre, E. Terzi, GraSS: Graph structure summarization., in: SDM, SIAM, 2010, pp. 454–465.
[10] M. Riondato, D. García-Soriano, F. Bonchi, Graph summarization with quality guarantees, in: 2014 IEEE International Conference on Data Mining, IEEE, 2014, pp. 947–952.
[11] K.U. Khan, W. Nawaz, Y.-K. Lee, Scalable compression of a weighted graph, 2016.
[12] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, ACM, 1998, pp. 604–613.
[13] K. Macropol, A. Singh, Scalable discovery of best clusters on large graphs, Proc. VLDB Endow. 3 (2010) 693–702.
[14] J. Leskovec, A. Rajaraman, J.D. Ullman, Mining of Massive Datasets, Cambridge University Press, 2014.
[15] M.E.J. Newman, The structure of scientific collaboration networks, Proc. Nation. Acad. Sci. 98 (2001) 404–409.
[16] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, Z. Su, Arnetminer: extraction and mining of academic social networks, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 990–998.
[17] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the internet topology, in: ACM SIGCOMM Computer Communication Review, volume 29, ACM, 1999, pp. 251–262.
[18] C. Hernández, G. Navarro, Compressed representations for web and social graphs, Knowl. Inf. Syst. 40 (2014) 279–313.
[19] K.U. Khan, W. Nawaz, Y.-K. Lee, Set-based unified approach for summarization of a multi-attributed graph, World Wide Web 20 (2017) 543–570.
[20] A. Maccioni, 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, ACM, 2016, pp. 1755–1764.
[21] L. Shi, S. Sun, Y. Xuan, Y. Su, H. Tong, S. Ma, Y. Chen, Topic: Toward perfect influence graph summarization, in: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), 2016, pp. 1074–1085.
[22] P. Boldi, S. Vigna, The webgraph framework i: compression techniques, in: Proceedings of the 13th International Conference on World Wide Web, ACM, 2004, pp. 595–602.
[23] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, P. Raghavan, On compressing social networks, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009, pp. 219–228.
[24] P. Liakos, K. Papakonstantinopoulou, M. Sioutis, Pushing the envelope in graph compression, in: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, 2014, pp. 1549–1558.
[25] J. Wang, H.T. Shen, J. Song, J. Ji, Hashing for similarity search: a survey, arXiv:1408.2927 (2014).
[26] A.Z. Broder, On the resemblance and containment of documents, in: Compression and Complexity of Sequences 1997. Proceedings, IEEE, 1997, pp. 21–29.