IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON DATA MINING FOR INTERNET OF THINGS

Received August 10, 2019, accepted September 12, 2019, date of publication September 16, 2019, date of current version October 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2941766

HcRPC: Highly Compact Reachability Preserving Graph Compression With Corrections

RUI BING[1], HUIFANG MA[1,2,3], XIANGCHUN HE[1], ZHIXIN LI[3], AND LIJUN GUO[4]

[1]College of Computer Science and Engineering, Northwest Normal University, Lanzhou 730070, China
[2]Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China
[3]Guangxi Key Lab of Multi-Source Information Mining and Security, Guangxi Normal University, Guilin 541004, China
[4]College of Information Science and Engineering, Ningbo University, Ningbo 315000, China

Corresponding author: Huifang Ma (mahuifang@yeah.net)

This work was supported in part by the National Natural Science Foundation of China under Grant 61762078, Grant 61363058, Grant 61762079, and Grant 61966004, in part by the Guangxi Key Laboratory of Trusted Software under Grant kx201910, in part by the Research Fund of Guangxi Key Lab of Multi-Source Information Mining and Security under Grant MIMS18-08, in part by the Ningbo Municipal Natural Science Foundation of China under Grant 2018A610057, and in part by the Zhejiang Provincial Natural Science Foundation under Grant LY17F030002.

ABSTRACT Graphs are used in numerous applications to model real-world systems and phenomena. The ever increasing size of graphs makes them difficult to query and analyze. In this paper, we propose *HcRPC*, a Highly compact Reachability Preserving Graph compression algorithm with Corrections, which is capable of preserving the reachability relations between the nodes in original graph. The highly compressed representation of a given graph consists of a compressed graph and a set of corrections. The original graph is compressed on the basis of equivalence class obtained via the reachability relations between nodes in the original graph. In the compressed graph, each node corresponds to a set of nodes from the original graph with similar ancestors and descendants, and each edge represents linkage between the original nodes in any two node sets. The corrections portion specifies the set of corrections, including equivalent class-node corrections and node-node corrections. MinHash technique is utilized to speed up checking whether equivalence classes are structure-similar and the pair of equivalence classes with high similarity are thus merged to acquire a highly compressed graph. Besides, we develop an algorithm for preserving compressed graph with a set of corrections in response to changes to the original graph. We evaluate our algorithms on real-life graph data sets and the results indicate that graph data sets can be highly compressed while preserving the reachability relations between nodes.

INDEX TERMS Graph compression, reachability query, MinHash, dynamic graph.

I. INTRODUCTION

Nowadays, it is increasingly common to find graphs with millions of nodes in various domains where nodes in the graph represent entities in the real-world, while edges represent the relationship between entity and entity, such as the relationships between users in social networks and web pages in Web graphs. Due to the increasing scale of graph data, it is impossible to process and analyze these graph data directly. In order to save storage space and facilitate the analysis of graph, it is necessary to compress large graph into smaller-scale graph. Therefore, graph compression has recently drawn intense research interest [1] and has been wildly applied in many scenarios and domains [2]–[10], such as community influence analysis [11]–[15], graph visualization [16], [17] and pattern discovery [18]–[20]. After compressing graph, the complexity of the graph is reduced while certain characteristics of the graph are preserved.

Consider a semantic network that represents people as nodes in the graph and relationships among people as edges in the graph. There are needs to understand whether two people are related for security reasons [21]. On biological networks, where nodes are either molecules, or reactions, or physical interactions of living cells, and edges are interactions among them, an important question is to find all genes whose expressions are directly or indirectly influenced by a given molecule [22]. All the above questions can be mapped into reachability queries, which make graph reachability queries to be a very basic type of graph queries for many applications. For reachability queries, it always takes $O(|V| + |E|)$ time

The associate editor coordinating the review of this manuscript and approving it for publication was Shirui Pan.

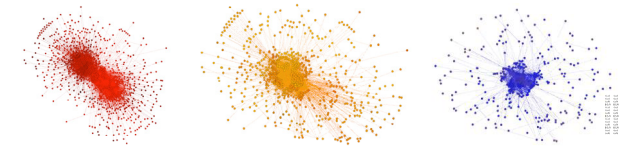


FIGURE 1. Compressing a real-life wiki-Vote social network.

via DFS/BFS search to determine whether there is a path between any two nodes in the graph $G = (V, E)$. Although indexes can be constructed to speed up evaluation, they can still incur additional costs [23]. Therefore, evaluating queries on graphs with millions of nodes and billions of edges is often prohibitively expensive, and we are unlikely to reduce the computational complexity. We may not change the complexity of graph queries algorithms, but we can reduce the size of the data graphs to obtain faster query speed. The method proposed in [24] uses equivalence classes to partition nodes in graphs, which effectively reduces the number of nodes in graphs. For instance, a real-life wiki-Vote social network can be highly compressed for reachability queries, as depicted in Figure 1. What is more, in many applications such as online social networks, the graph evolves over time. We therefore consider the problem of updating the compressed graph given the graph changes over time.

To this end, in this paper, we introduce *HcRPC*, a *Highly compact Reachability Preserving* graph *Compression* with corrections. Figure 2 gives the framework of our methods. It first divides nodes from the original graph into equivalent classes via the equivalence relation of reachability. Then each pair of equivalence classes are further compressed based on the similarities of their ancestors and descendants. What is more, a set of corrections is decided to maintain the reachability of the nodes in the original graph. *HcRPC* is relative to the reachability queries, generating smaller graphs while preserving information only relevant to reachability queries rather than the entire original graphs, and therefore, achieves a better compression ratio. In addition, any algorithm available for evaluating reachability queries can be directly performed to query the compressed graphs, as is, without decompressing. In order to handle the dynamic changes of original graph, we also propose an *Incremental Reachability Preserving Compression* algorithm, i.e. *IncRPC*, to update both the compressed graph and corrections in the dynamic graph, allowing us to maintain the reachability of original graph without decompressing after compress the graph once. We verify the effectiveness and efficiency of our compression techniques through experiments using real-life data.

The contribution can be summarized as follows:

(1). We propose a reachability query preserving compression method for querying reachability relations on large real-life graphs. Contrary to general graph data compression strategies, our method only retains the information needed for reachability queries, and thus achieves better compression ratio by merging similar equivalence classes via MinHash. Any algorithm for calculating the reachability query of the original graph can be directly performed on the compressed graph without decompression.

(2). As is known that merging similar sets will inevitably incur reachability relation errors, a set of corrections is established to maintain the same reachability relations on the compressed graph as in the original graph. For reachability queries, by calculating similarities and merging equivalence classes to achieve better compression ratio, the reachability relations in the compressed graph are kept unchanged through the set of corrections. This not only achieves preferable compression ratio, but also maintains the reachability relationship in the original graph.

(3). In order to cope with the dynamic changes of real-life graphs, we propose an incremental reachability preserving compression algorithm. Given a graph G, its compression G_s, and updates ΔG to G, the aim is to utilize similarity calculation and corrections to make the reachability relations on the updated compressed graph G_s consistent with the original graph G. This allows us to compute the compressed G_s once and maintain it incrementally according to the changes in G.

II. RELATED WORKS

Graph compression methods can be roughly categorized into two groups: general graph compression and query-friendly graph compression.

A. GENERAL GRAPH COMPRESSION

General methods preserve the information of the entire graph, and highly depend on extrinsic information, coding mechanisms and application domains [25], However, these methods need decompression before querying the graph. Existing general graph compression algorithms can be summarized into the following four categories: (a). Attribute-based graph compression algorithms: On-Line Analytical Processing (OLAP) technology is developed to form the graph OLAP method, collecting graph data from both information OLAP and topology OLAP [26], and probability is considered to compress graphs [27]. (b). Structure-based graph compression algorithms: There are currently two strategies, one is probability-based graph compression method [28], which takes zero reconstruction error as the benchmark, and constructs reconstruction error according to the differences between the expected adjacency matrix calculated by the super-graph and the adjacency matrix of the original graph. The smaller the reconstruction error is, the better the accuracy is. The other is the graph compression method based on maximum compression, aiming to minimize the size of the super-graph. The most notable algorithm is the Minimum Description Length (MDL) representation method [29]. These methods are compressed only according to the structure of the nodes, ignoring the edge weights corresponding to the attribute information on the nodes and the compactness between the nodes. (c). Structure and attribute-based graph compression algorithm: At present, there are two different

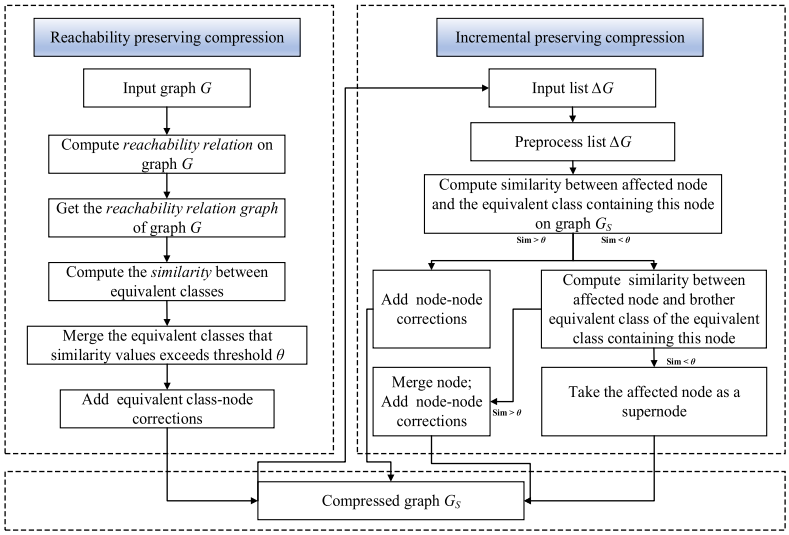


FIGURE 2. Framework for reachability preserving graph compression.

strategies of graph compression algorithm based on structure and attributes. One is to utilize attribute similarity to partition nodes set, and then utilize structural similarity to partition attribute similarity groups. The other is to transform attribute information into structural information and measure both of them into the same measure function, to directly obtain the final compressed graph [30]. The representative graph compression algorithms are: graph compression technology based on entropy model in information theory [31], SNAP/k-SNAP compression algorithms [32], [33], generating compression graphs according to the attributes of nodes and edges selected by users. (d). Graph compression algorithm based on weighted graph: Weighted graph compression algorithms aim at calculating the weights of the superedges between node sets formed by node compression [34].

Our method is different from the general graph compression methods from the following aspects: the compression method only depends on the reachability relationship in the graph, and the compressed graph is only used for a specific class of queries, i.e., the reachability queries. The compressed graph obtained by *HcRPC* can be directly used to perform reachability queries without decompressing the compressed graph.

B. QUERY-FRIENDLY GRAPH COMPRESSION

Closer to our work, query-friendly graph compression approaches target specific classes of queries. Queries can be summarized in the following categories:

Neighborhood queries. The purpose of neighborhood queries is to find nodes connected to the specified node in the graph [29], [35], [36]. The idea of query-able compression (no decompression query) for such queries is proposed in [35], which adopts the compressed data structures by exploiting Eulerian paths and multi-position linearization. In [36], an S-node representation is introduced to solve neighborhood query on network graph. The aim of graph summarization [29] is to sketch graphs with small subgraphs and construct super-graph abstraction. These methods construct compact data structures that should be decompressed to answer queries [25]. In addition, the query evaluation algorithms on the original graph should be modified to answer the query in the compact data structures.

Distance-based queries. The distance-based queries are usually executed on the weight graphs. The purpose of distance-based query is to obtain the shortest path between any two nodes in the graph [37]–[39]. [37] propose a novel strategy to simplify weighted graphs by pruning least important edges from them by defining a graph connectivity function based on the best paths between all pairs of nodes. Given the number of edges to be pruned, the problem is then to select a subset of edges that best maintains the overall graph connectivity. Reference [38] introduces a new approach 'gate graph' from a large graph so that for any 'non-local' node pair (distance greater than some threshold) in the original graph, the shortest-path distance can be recovered by consecutive 'local' walks through the gate vertices in the gate graph, to perform graph simplification. Reference [39] introduces a novel distance preserving compression method called Shrink, which can be used to query and store both weighted and unweighted graphs. Compressing with Shrink has the least effect on the distances between nodes because a system of equations is introduced to minimize the distance variations caused by nodes merge.

Reachability queries. The aim of the reachability queries is to determine whether any two nodes in the original graph are reachable [24], [40]–[43]. To answer these queries, [40] computes the minimal subgraph using the same transitive closure as the original graph, and [41] reduces the graph by replacing a simple cycle for each strongly connected component. These methods allow reachability queries to be evaluated on a compressed graph without decompression. Bipartite compression [42] reduces the graphs by introducing virtual

nodes and compressing bicliques. However, its compression method is to establish the bijection between the original graph and its compression graph, so that they can be converted to each other. In contrast, our method does not need to restore the original graph; and the algorithms for reachable queries must be modified before they are applied to the compressed graphs [42], [43] calculates a compresses bit vector to encode the transitive closure of a graph. In contrast, the reachable algorithm and the compression scheme in [43] can be directly applied to the compressed graph. Incremental maintenance of bit vectors is not mentioned in [43]. Compared with the reachability query methods discussed above, our method achieves a better compression ratio, since our compressed graph do not necessarily be a subgraph of the original graph and some of the compression methods only reduce the number of the edges [40], [42], [43] while our method reduces the number of nodes and edges by merging nodes into super-nodes.

Unlike previous graph compression methods, $Compress_R$ [24] preserves information needed for answering reachability queries which divides the nodes in the original graph into several equivalence classes by equivalence relation of reachability to reduce the graph and achieves better compression effect.

It is worth noting that our work is a significant extension of $compress_R$. $Compress_R$ obtains equivalence classes through equivalence relation of reachability while the aim of our proposed *HcRPC* is to obtain highly compressed graph via merging sets with high similarities, and corrections are introduced to maintain the reachability of the original graph, which makes *HcRPC* substantially different from $compress_R$ in terms of both key ideas and techniques. Meanwhile, *IncRPC* is to utilize similarity calculation and corrections to make the reachability relations on the updated compressed graph G_s consistent with the original graph G instead of separating nodes each step in the incremental algorithm as that in $compress_R$ [24]. Furthermore, our method is able to achieve higher compression rate through merging similar equivalent classes.

III. PRELIMINARIES

In this section, we introduce some necessary notations to describe the essential basic concepts and the MinHash technique used in similarity computing.

A. GRAPH AND REACHABILITY QUERY

Given a directed graph consisting of nodes and edges denoted by $G = (V, E)$ where V is a set of nodes, $E \subset V \times V$ is a set of edges, $\langle u, v\rangle \in E$ denotes a directed edge from node u to v.

A path p from node u to v in G is a sequence of nodes ($u = u_o, u_1, u_2 \ldots u_n = v$), for each $i \in [1,n]$, $\langle v_{i-1}, v_i\rangle \in E$. The length of path p denoted by $len\,(p)$, is the number of edges in path p. A node u can reach v (or v is reachable from u) if and only if (iff) there exists a path from u to v in G.

A graph $G_s = (V_s, E_s)$ is a compressed graph of G, where V_s is a set of supernodes and each supernode is composed of a set of nodes in G, while E_s is a set of superedges between supernodes and each superedge represents the edges between the original nodes in the supernodes. In the following section, we will detail how to create supernodes based on the similarity between sets, and how to establish superedges via the connection of the original nodes in the supernodes.

A reachability query on a graph G is $Q_R\,(u, v)$, which is a boolean query asking whether node u can reach node v in G.

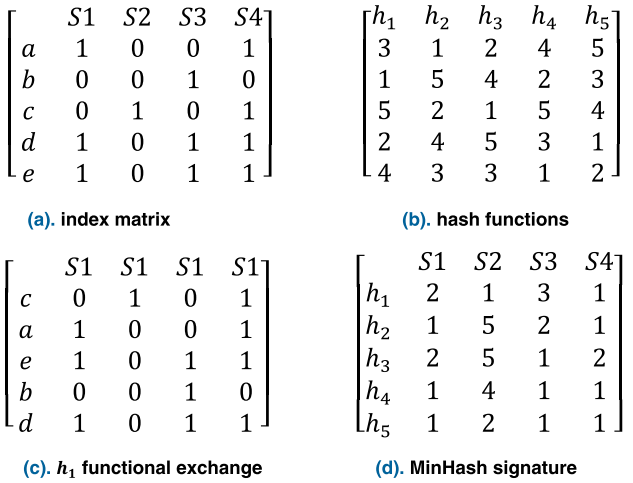


FIGURE 3. MinHash computing process.

B. MINHASH

MinHash is a technique for quickly estimating how similar two sets are [44]. Initially used in AltaVista search engine to detect duplicate web pages and delete them from search results. The goal of MinHash is to quickly estimate Jaccard similarity between sets, without explicitly computing the intersection and union.

Specifically, an index matrix $\boldsymbol{I}$ size of $n \times k$ is constructed, where n is the total number of distinct elements in the union of all the sets to be compared and k is the number of sets. The entry $\boldsymbol{I}_{ij}$ in the index matrix represents whether the jth set contain the element i in the union of all the sets. Different hash functions $h_1, h_2, \ldots, h_l$ are defined and each hash function is a random permutation on n elements. That is to say, each hash function is a random row order exchange of the index matrix $\boldsymbol{I}$. According to the row order exchange of the index matrix $\boldsymbol{I}$ based on hash function h_i, the row number of the first non-zero value of each column in the exchanged index matrix is the MinHash value of the set represented by this column. Therefore, after exchanges l times, each set is attached with l MinHash values. In terms of the l different hash functions and k sets, a MinHash signature matrix $\boldsymbol{M}$ $(l \times k)$ is created where $\boldsymbol{M}_{ij}$ represents MinHash value on hash function h_i for the jth set.

The similarity between columns in the MinHash signature matrix is considered as the similarity of the set represented by this column. Clearly, the similarity values between two sets are real numbers within the interval [0,1].

Example 1: Take four sets, $S1$, $S2$, $S3$ and $S4$ as an example, where $S1 = \{a, d, e\}$, $S2 = \{c\}$, $S3 = \{b, d, e\}$, $S4 = \{a, c, d, e\}$, the union set $U = \{a, b, c, d, e\}$. Next we show the calculation of the similarity between the sets. We create an index matrix shown in Figure. 3(a) and five different hash functions, h_1, h_2, h_3, h_4, h_5 are generated randomly as shown in Figure. 3(b) to exchange row order in index matrix. Figure. 3(c) indicates h_1 functional exchange on index matrix and the MinHash values of $S1$, $S2$, $S3$ and $S4$ are 2, 1, 3, 1, respectively. The exchange process if the rest hash functions are similar with h_1, based on which the MinHash values of these sets are obtained. Finally, we create the MinHash signature matrix shown in Figure. 3(d). The calculation of similarity between two sets is conversed as the similarity between the two corresponding columns in the MinHash signature matrix. For instance, the similarity between $S1$ and $S4$, $S3$ and $S4$ are 0.8 and 0.4, respectively.

IV. COMPRESSION FOR REACHABILITY

In this section, we propose the highly compact two-part representation of a given graph G consisting of a graph summary and a set of corrections. In contrast to reachability preserving algorithm proposed by Fan *et al.* [24], our highly compact reachability preserving compression algorithm can effectively reduce the number of equivalent classes and save storage costs.

A. REACHABILITY RELATION GRAPH

The *reachability relation* on graph $G = (V, E)$ is defined as a binary relation $R_e \subseteq V \times V$, for each $(u, v) \in R_e$ and any node $x \in V$ such that x can reach u if and only if x can reach v; and u can reach x if and only if v can reach x. Put it another way, $(u, v) \in R_e$ if and only if they have the same set of ancestors and the same set of descendants.

We use $[u]_{R_e}$ to denote equivalence class containing node u and other nodes in the same equivalence class with the same set of ancestors and the same set of descendants, indicating each equivalence class owns a set of ancestors and set of descendants.

The reachability relation graph of graph $G = (V, E)$ is defined as $G_r = (V_r, E_r)$ where $V_r = \left\{ [v]_{R_e} \middle| v \in V \right\}$ is a set of equivalence classes partitioned via reachability relation and E_r is a set of edges. There is an edge $\left\langle [u]_{R_e}, [v]_{R_e} \right\rangle$ in E_r if there exist nodes $u' \in [u]_{R_e}$ and $v' \in [v]_{R_e}$ such that $\left\langle u', v' \right\rangle \in E$. Therefore, the reachability relation graph can be regarded as the initial compressed graph for the original graph.

B. SIMILARITY CALCULATION AND CORRECTIONS

Suppose there are k equivalence classes in V_r for the reachability relation graph for graph $G = (V, E)$, denote as $(S_1, S_2, \ldots, S_k)$. Two equivalent class-node matrices based on the reachability relation, $\boldsymbol{A}$ and $\boldsymbol{D}$, are created respectively, with both sizes of $n \times k$, where n is the total number of nodes in graph $G = (V, E)$. Entries in Matrix $\boldsymbol{A}$ denote which nodes are ancestors of the nodes in each equivalent class in the reachability relation graph. Thereafter, each row of matrix $\boldsymbol{A}$ indicates the node in graph G, and each column indicates the equivalent class in the reachability relation graph. $\boldsymbol{A}_{ij}$ is a binary value indicating whether node i is an ancestor of the nodes in the equivalent class S_j. Similar to matrix $\boldsymbol{A}$, matrix $\boldsymbol{D}$ denotes which nodes are descendants of the nodes in each equivalent class. $\boldsymbol{D}_{ij}$ in matrix $\boldsymbol{D}$ represents whether node i is the descendant of nodes in the equivalent class S_j.

MinHash is adopted to calculate the similarities of ancestors and descendants between each pair of equivalent classes S_i, S_j denoted by $sim_{an}(S_i, S_j)$ and $sim_{des}(S_i, S_j)$, respectively. Next, we propose a joint similarity to measure the similarity between S_i, S_j.

$$sim(S_i, S_j) = \frac{1}{2}\left(sim_{an}(S_i, S_j) + sim_{des}(S_i, S_j)\right) \quad (1)$$

A pair of equivalent classes S_i, S_j is merged to create a supernode if $sim(S_i, S_j) \geq \theta$, where θ is the similarity threshold to determine the similarity between two equivalent classes. Otherwise we create two supernodes to represent two equivalence classes S_i, S_j. In the experiments, we will detail the setting of threshold θ.

To eliminate reachability errors caused by merging equivalence classes, we introduce a set of corrections C to preserve the reachability relations in the original graph $G = (V, E)$.There are two types of equivalent class-node corrections, $-r()$ and $+r()$, where $+r()$ represents that the equivalent class are reachable to a node while $-r()$ represents that the equivalent class are unreachable to a node. For instance, the equivalence class S_j has the descendant u and the ancestor v while S_i does not have the descendant u and the ancestor v. We create corrections to preserve reachability by merging equivalence classes S_i, S_j, where each equivalent class-node correction is a tuple denoted by $-r(S_i, u)$ or $-r(v, S_i)$. The $-r(S_i, u)$ represents the equivalence class cannot reach the descendant u because node u is the descendant of S_j, and the $-r(v, S_i)$ represents the ancestor v cannot reach equivalence class S_i since node v is the ancestor of S_j.

C. REACHABILITY PRESERVING COMPRESSION

We next present the highly compact reachability preserving compression algorithm, *HcRPC*. Given a graph $G = (V, E)$, the objective is to calculate the compressed graph $G_s = (V_s, E_s)$ and corrections C. The algorithm is shown in Figure. 4.

Given a graph $G = (V, E)$, *HcRPC* firstly computes its reachability relation R_e and obtain the partition $Par = V/R_e$ of G consisting of equivalent classes (lines 2-3). After this, equivalent classes are used to form the reachability relation graph G_r (lines 4-10). According to (1), *HcRPC* computes the joint similarity between each pair of equivalent classes S_i, S_j in graph G_r (lines 11-14). At this point, the algorithm selects the pair of equivalence classes whose joint similarity exceeds the threshold θ to merge. For the equivalence classes whose joint similarity does not exceed the threshold, they

Algorithm 1. ***HcRPC***

Input : $G = (V, E)$

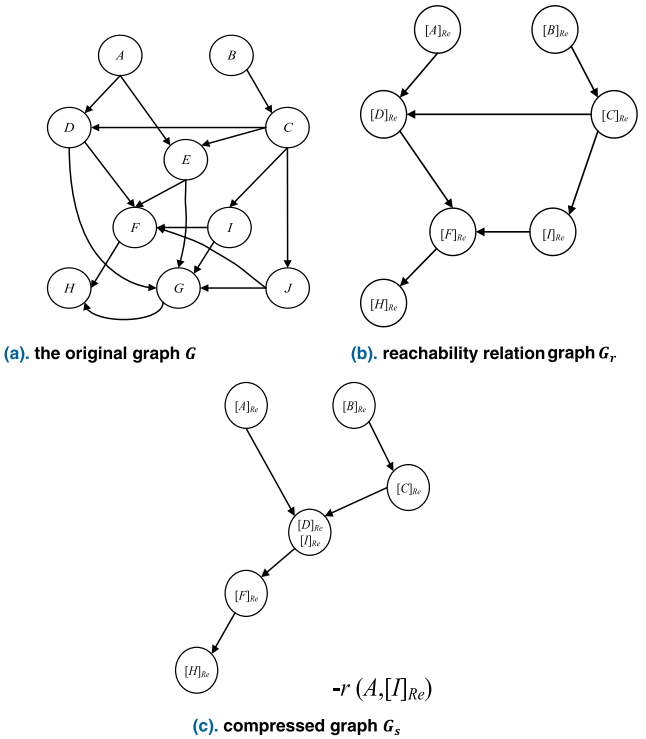
Output : $G_s = (V_s, E_s), C$

01: initialize $V_s = \emptyset, E_s = \emptyset, V_r = \emptyset, E_r = \emptyset$;
02: compute reachability preserving relation R_e
03: compute the partition $Par = V \,/\, R_e$ of G
04: **for each** $S_i \in Par$ do
05: $\quad V_r = V_r \cup \{S_i\}$
06: **for each** $S_i, S_j \in V_r$ do $\langle u, v\rangle$
07: $\quad$ **if** there exist $u \in S_i$, $v \in S_j$ such that
08: $\quad\quad \langle u, v\rangle \in E$ but S_i does not reach S_j
09: $\quad$ **then** $E_r = E_r \cup \{\langle S_i, S_j\rangle\}$
10: **return** reachability relation graph G_r of graph G
11: create equivalent class-node matrices, $\boldsymbol{A}$ and $\boldsymbol{D}$
12: **for each** $S_i, S_j \in V_r$ do
13: $\quad$ compute similarity of ancestors and descendants between S_i, S_j using (1)
14: $\quad$ compute joint similarity between S_i, S_j
15: $\quad$ **if** $sim(S_i, S_j) \geq \theta$
16: $\quad\quad$ Merge (S_i, S_j)
17: $\quad\quad$ create a supernode v_{si}, $V_s = V_s \cup \{v_{si}\}$
19: $\quad$ **else**
20: $\quad\quad$ create two supernodes v_{sx}, v_{sy} $V_s = V_s \cup \{v_{sx}, v_{sy}\}$
21: **for all** pairs $(v_{si}, v_{sj}) \in V_s$
22: $\quad$ **if** exist $S_i \in v_{si}$, $S_j \in v_{sj}$ such that $\langle S_i, S_j\rangle$
23: $\quad\quad$ **then** $E_s = E_s \cup \{\langle v_{si}, v_{sj}\rangle\}$
24: $\quad\quad$ Add equivalent class-node corrections to C
25: **return** G_s, C

FIGURE 4. Algorithm ***HcRPC***.

become supernodes independently. Meanwhile, supernodes are created to represent the merged equivalence classes and single equivalence classes, and they are then added to set V_s (lines 15-20). The strategy of adding superedge for each pair S_i, S_j is that if there is an edge between elements in the equivalence classes S_i, S_j, the superedge between S_i, S_j is created. After this, a set of equivalent class-node corrections is generated to preserve the reachability relations on compressed graph G_s (lines 21-24). Finally, the compressed graph G_s and corrections C are constructed and returned (line 25).

Example 2: Consider the graph G given in Figure. 5(a), in terms of reachability relations on the graph G, nodes D, E are in the same equivalence class, F, G are in the same equivalence class and I, J are in the same equivalence class



(a). the original graph G (b). reachability relation graph G_r

(c). compressed graph G_s

FIGURE 5. Reachability preserving compression.

as shown in Figure. 5(b) since they own same ancestors and descendants. Let the threshold θ equals to 0.75, after calculating the joint similarity between each equivalent classes in the reachability relation graph G_r, the joint similarity between $[D]_{R_e}$ and $[I]_{R_e}$ exceeds the threshold θ. Therefore, we merge $[D]_{R_e}$ and $[I]_{R_e}$ together and create a supernode for V_s. In addition, node A cannot reach $[I]_{R_e}$ in the graph G_r, we thus add an equivalent class-node correction to preserve the reachability of graph G shown in Figure. 5(c).

V. INCREMENTAL REACHABILITY PRESERVING COMPRESSION

In order to deal with the dynamic changes of graphs, in this section we propose an incremental reachability preserving compression algorithm for compressed graph. The updates ΔG is defined as a list of edge deletions and insertions to the original graph G. The algorithm we proposed can preserve the reachability of the updated original graph directly through the compressed graph G_s by using node corrections without decompressing the compressed graph G_s and separating each affected node in each step.

A. PREPROCESSING

Before we deal with edge operations in updates ΔG, we need to preprocess the updates to remove edge operations that have no affect to reachability on compressed graph G_s. The strategy of removing edges is similar with [24]: According to the compressed graph G_s and corrections C, the preprocessing removes (a) edge insertions $\langle u, v\rangle$ where $[u]_{R_e} \neq [v]_{R_e}$, and $[u]_{R_e}$ can reach $[v]_{R_e}$ in graph G_s; and (b) edge deletions $\langle u, v\rangle$

if either $[u]_{R_e}$ can reach $[v]_{R_e}$ through a path with length no less than two in G_s.

Two types of node-node corrections are adopted denoted by $-()$ and $+()$, respectively, where $+()$ represents that the pair of nodes have an edge between each other and $-()$ represents that there is no edge between this pair of nodes.

B. INCREMENTAL REACHABILITY PRESERVING COMPRESSION

Here, we present the incremental reachability preserving compression algorithm that given a graph $G = (V, E)$, compressed graph $G_s = (V_s, E_s)$ of G and updates ΔG, calculate its updated compressed graph $G_s = (V_s, E_s)$ and corrections C. The algorithm is shown in Figure. 6.

Given a graph $G = (V, E)$, compressed graph $G_s = (V_s, E_s)$ of G and updates ΔG, the algorithm first removes redundant edges in ΔG (line 1). Next, for each edge in the updates ΔG, the similarity between the affected node and its equivalent class (lines 2-3) is calculated. The algorithm adds node-node corrections if the joint similarity between the affected node and its equivalent class exceeds the threshold θ, and separates the affected node from its equivalent class if the joint similarity between the affected node and its equivalent class below the threshold θ (lines 4-31). Then, for the separated affected node, the algorithm calculates the joint similarities between the node and its equivalent class' brother equivalent classes in set B (lines 9,18,26). The separated affected node is merged into equivalent class in set B if the joint similarity exceeds the threshold θ (lines 10-11, 19-20, 27) and a supernode is created to represent this separated affected node if the joint similarity is below the threshold θ (lines 12-14, 21-23, 29-31). $N(u)$, $N(v)$ represent the neighbor node set of nodes v, respectively. Thus, the updated compressed graph G_s and corrections C are established and returned (line 32).

Example 3: Recall the graph G proposed in Figure 5. Now we update graph G with ΔG, where ΔG is composed of two edge insertions $\langle D, H\rangle$, $\langle A, I\rangle$ and one edge deletion $\langle E, G\rangle$ as shown in Figure. 7(a). According to the reachability relation of graph G_s, we can see that the edge insertion $\langle D, H\rangle$ can be removed in preprocessing phase since $[D]_{R_e}$ can reach H. Next, we perform the edge insertion $\langle A, I\rangle$. After inserting $\langle A, E\rangle$, we compute the similarities between node A and its equivalent class $[A]_{R_e}$, node E and its equivalent class $[I]_{R_e}$. Let the threshold θ equals to 0.75, it is clear that both the similarities between node A and its equivalent class $[A]_{R_e}$ node I and its equivalent class $[I]_{R_e}$ exceed the threshold θ. Therefore, there is no need to separate node A and I from their equivalent classes, and a node correction $+(A, I)$ is added to denote that there is an edge from A to I. Same as the edge insertion, we perform the edge deletion $\langle E, G\rangle$ and then compute the similarities between node E and its equivalent class $[D]_{R_e}$, node G and its equivalent class $[F]_{R_e}$. The result in Figure. 7(b) indicates that there is no need to separate node E and G from their equivalent classes, and a node correction $-(E, G)$ is supplemented to denote the deletion of edge from E to G.

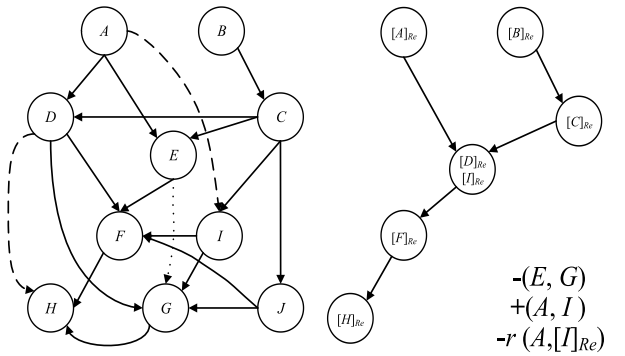
Algorithm 2. ***IncRPC***

Input: $G = (V, E)$, $G_s = (V_s, E_s)$, updates ΔG
Output: the updated graph $G_s = (V_s, E_s)$, C
01: remove redundant edges in ΔG
02: **for each** update $\langle u, v\rangle$ in ΔG
03: compute $sim(u, [u]_{R_e})$ and $sim(v, [v]_{R_e})$
04: **if** $sim(u, [u]_{R_e}) \geq \theta$ and $sim(v, [v]_{R_e}) \geq \theta$
05: add node-node correction to C
06: **elif** $sim(u, [u]_{R_e}) < \theta$ and $sim(v, [v]_{R_e}) \geq \theta$
07: separate u from $[u]_{R_e}$
08: **for each** $[x]_{R_e} \in B([u]_{R_e})$
09: compute $sim(u, [x]_{R_e})$
10: **if** $sim(u, [x]_{R_e}) \geq \theta$
11: merge $(u, [x]_{R_e})$ and add node-node corrections to C
12: **else**
13: create a supernode v_{si}, $V_s = V_s \cup \{v_{si}\}$
14: create superedges between v_{si} and nodes in $N(u)$
15: **elif** $sim(u, [u]_{R_e}) \geq \theta$ or $sim(v, [v]_{R_e}) < \theta$
16: separate v from $[v]_{R_e}$
17: **for each** $[x]_{R_e} \in B([v]_{R_e})$
18: compute $sim(v, [x]_{R_e})$
19: **if** $sim(v, [x]_{R_e}) \geq \theta$
20: merge $(v, [x]_{R_e})$ and add node-node corrections to C
21: **else**
22: create a supernode v_{si}, $V_s = V_s \cup \{v_{si}\}$
23: create superedges between v_{si} and nodes in $N(v)$
24: **else**
25: separate u from $[u]_{R_e}$ and v from $[v]_{R_e}$
26: **if** $sim(v, [y]_{R_e}) \geq \theta$ or $sim(u, [x]_{R_e}) \geq \theta$, $[x]_{R_e} \in B([u]_{R_e})$, $[y]_{R_e} \in B([v]_{R_e})$
27: merge $(u, [x]_{R_e})$ or merge $(v, [y]_{R_e})$
28: add node-node correction to C
29: **else**
30: create two supernodes v_{sx}, v_{sy} $V_s = V_s \cup \{v_{sx}, v_{sy}\}$
31: create superedges between v_{sx} and nodes in $N(u)$, v_{sy} and nodes in $N(v)$
32: **return** G_s, C

FIGURE 6. Algorithm *IncRPC*.

TABLE 1. Datasets.

dataset	type	nodes	edges
soc-Epinions1	social networks	75,879	508,837
soc-Pokec	social networks	1,632,803	30,622,564
wiki-Vote	social networks	7,115	103,689
wiki-Talk	Communication networks	2,394,385	5,021,410
amazon	Product co-purchasing networks	403,394	3,387,388
p2p-Gnutella	Internet peer-to-peer networks	62,586	147,892



(a). the updated original graph G compressed graph G_s (b). the updated

FIGURE 7. Incremental reachability preserving compression.

VI. EXPERIMENTS

In this section, we present the experimental evaluation on real-life data sets using algorithms introduced in the previous section. Furthermore, we investigate the impact of the graph size and graph type.

A. EXPERIMENT SETTING

We run our algorithms on the following real-life data sets with different graph type to evaluate the proposed techniques. We adopt six real-world networks collected by SNAP[1] with four graph types shown in Table 1.

Three *social networks*. (a). soc-Epinions1 is a who-trust-whom online social network of a general consumer review site Epinions.com. Members of the site can decide whether to 'trust' each other. All the trust relationships interact and form the Web of Trust which is then combined with review ratings to determine which reviews are shown to the user. (b). soc-Pokec: Pokec is the most popular online social network in Slovakia. In the soc-Pokec, a node represents a user and edges represent friendships between users. Datasets contains anonymized data of the whole network and friendships in Pokec are oriented. (c). wiki-Vote: This network contains all the Wikipedia voting data from the inception of Wikipedia. Nodes in the network represent Wikipedia users and a directed edge from node i to node j represents that user i votes on user j.

[1]http://snap.stanford.edu/data/

Communication network. wiki-Talk: The network contains all the users and discussion from the inception of Wikipedia. Nodes in the network represent Wikipedia users and a directed edge from node i to node j represents that user i at least once edited a talk page of user j.

Product co-purchasing network. amazon: In this network, a node represents a product and edges represent co-purchasing relations. If a product i is frequently co-purchased with product j, the graph contains a directed edge from i to j.

Internet peer-to-peer network. p2p-Gnutella: In this network, nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts.

B. EVALUATION METRICS

We utilize the compression ratio widely used in numerous graph compression methods to measure the performance of the reachability preserving compression algorithm. The compression ratio of G_s is defined as follow:

$$cr\,(G_s) = \frac{|G_s|}{|G|} \tag{2}$$

In (2), $|G_s|$ denotes the sum of the number of supernodes and superedges in the compressed graph G_s of the original graph G and $|G|$ denotes the sum of the number of nodes and edges in the original graph G.

In addition, we define storage cost to measure the memory cost of the compressed graph, where the storage cost is defined as follow:

$$c\,(G_s) = |G_s| + |C| \tag{3}$$

Same as in (2), $|G_s|$ in (3) denotes the sum of the number of supernodes and superedges in the compressed graph G_s and $|C|$ denotes the number of two type corrections in graph G_s.

We conduct a series of experiments to verify our algorithm: the effectiveness of the reachability preserving compression algorithm and incremental reachability preserving compression algorithm, the performances are measured by compression ratio; the effectiveness of the query processing measured by query evaluation time over original and compressed graphs; the efficiency of the incremental reachability preserving compression algorithm measured through update execution time and the performance of storage cost

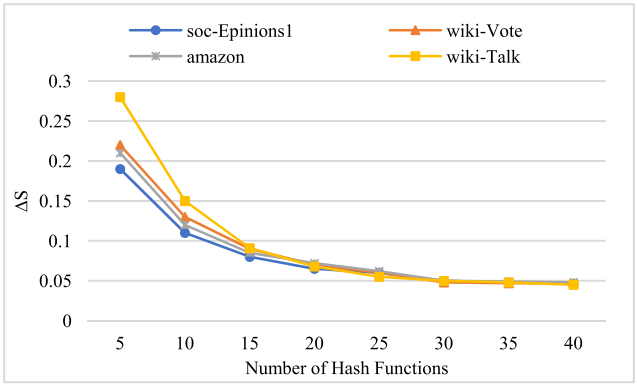


FIGURE 8. Hash function setting.

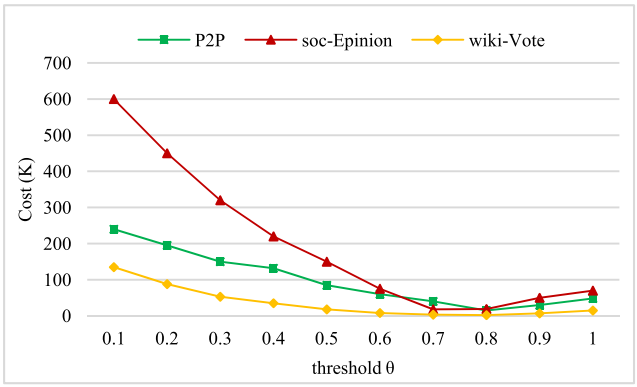


FIGURE 9. Threshold setting.

with corrections. Besides, we conduct experiment to select reasonable parameter θ and number of hash functions.

We compare our method performance with one of the state-of-the-art graph compression method introduced in [24], where the reachability preserving compression in [24] is named after $compress_R$ and the incremental compression is named after *incRCM*.

C. EXPERIMENTAL RESULTS

1) HASH FUNCTION SETTING

Firstly, we introduce how to choose the optimal number of hash functions on four datasets. The result is shown in Figure. 8, where the X-axis represents different numbers of hash functions and ΔS in the Y-axis represents the difference of similarities between the same sets measured by the previous number of hash functions and the current number of hash functions. From Figure 8, we can see that when the number of hash functions reaches 30, ΔS tends to be stable. Therefore, we set the number of hash functions to 30 for calculating similarities.

2) THRESHOLD SETTING

In this set of experiments, we introduce how to choose the optimal threshold θ according to the storage cost. Figure 9 shows how the compression performance of the *HcRPC* algorithm changes as we change the values. Three different colors indicate three different datasets, P2P, soc-Epinions1 and wiki-Vote, respectively. In Figure. 9, The X-axis represents different threshold values ranging from 0.1 to 1 with 0.1 increments and the Y-axis represents the storage cost of compressed graph. For each θ, we run the algorithm five times and pick the result which is associated with the best storage cost. The result reveals that when the threshold θ is 0.75 on average, the storage cost $c(G_s)$ of the compressed graph is the smallest since exorbitant setting of similarity threshold θ brings about few or even no equivalent classes to be merged. Similarly, excessively low setting of similarity threshold θ allows any two equivalent classes to be merged, too much corrections, even more than the edges in the original graph, and thus the compressed graph is with a highly expensive storage cost. Hence, we set the threshold θ to 0.75 when we merge equivalent classes. From Figure 9, it is clear that the P2P data set with smaller size than soc-Epinions1 achieves a slightly higher storage cost than that of soc-Epinions1 in the case of choosing the optimum threshold θ, while the compression effectiveness of P2P data sets is slightly lower than that of the other two data sets, because soc-Epinions1 and wiki-Vote have higher connectivity than P2P networks.

3) EFFECTIVENESS OF REACHABILITY COMPRESSION

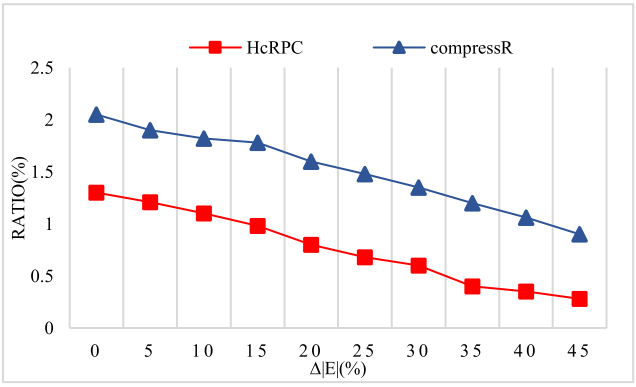
We then evaluate the effectiveness of *HcRPC* using compression ratio on real-life datasets. We treat the compression ratio as a measurement for representation of compression effectiveness, which differs from the ratio measuring the memory cost reduction. The smaller the compression ratio is, the more effective the compressing method is. The compression ratios of reachability preserving compression are reported in Table 2. The table shows that: (a) *HcRPC* can highly compress real-life graphs. Indeed, $cr(G_s)$ obtained by *HcRPC* is in average 2.2% over the six datasets. For wiki-Vote data set, it is up to 99.8% reduction in original graph size. (b) Comparing with $compress_R$, the compression ratios produced by the *HcRPC* are about 40% lower than $compress_R$ because of merging the similar pair of equivalence classes. (c) The *HcRPC* perform best on social networks and product co-purchasing networks since social networks and product co-purchasing networks are with higher connectivity than other graph types. The denser edges of the network, the more nodes can be merged. Compared with other networks, the compression ratio $cr(G_s)$ obtained by *HcRPC* on P2P network is about 60% higher than others.

4) EFFECTIVENESS OF INCREMENTAL REACHABILITY COMPRESSION

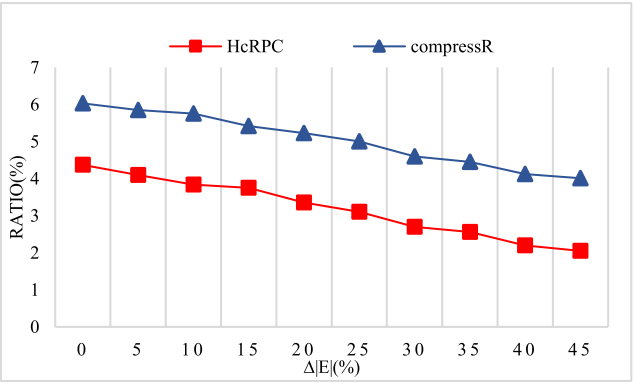
We evaluate the effectiveness of the *IncRPC*, in terms of compression ratio $cr(G_s)$. The result is shown in Figure. 10, where $\Delta|E|$ in X-axis represents the percentage of the total edge updates performed in 5% increments and Y-axis represents the compression ratio $cr(G_s)$. To facilitate comparison with the *incRCM*, we only select edge insertions and execute them for edge updates. Figure. 10(a) shows the compression effectiveness of *IncRPC* and *incRCM* on wiki-Vote dataset. We can see that the more edges inserted into

TABLE 2. Reachability preserving: compression ratio.

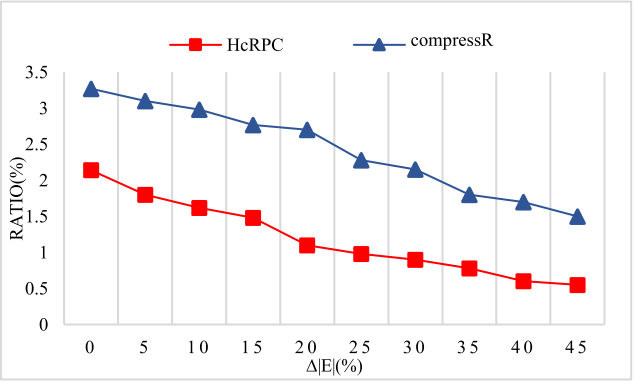
dataset	$\|G\|(\|V\|, \|E\|)$	$cr(G_s)$ by $compress_R$	$cr(G_s)$ by *HcRPC*
soc-Epinions1	585K (76K, 509K)	2.88%	1.82%
soc-Pokec	31.6M (1.6M, 30M)	2.79%	2.01%
wiki-Vote	110K (7K, 103K)	2.03%	1.27%
wiki-Talk	7.4M (2.4M, 5.0M)	3.27%	2.14%
amazon	3.8M (403K, 3.4M)	1.86%	1.29%
p2p-Gnutella	211K (63K, 148K)	6.03%	4.37%



(a). wiki-Vote dataset



(b). P2P dataset



(c). Wiki-Talk dataset

FIGURE 10. Effectiveness of incremental reachability compression.

original graph, the better the graph can be compressed for reachability queries as with the increase of edges, more nodes may belong with the same reachability equivalent classes. As can be seen from Figure. 10(a), the *IncRPC* achieves better

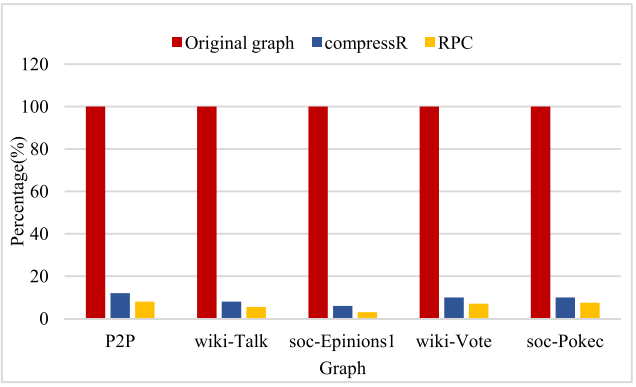


FIGURE 11. Effectiveness of query processing.

compression ratio than *incRCM* on account of merging the pair of equivalence classes, which exceeds the joint similarity threshold θ. Figure. 10(b) and Figure. 10(c) show the result of the execution of *IncRPC* and *incRCM* on P2P and wiki-Talk datasets, respectively. Similar to the result with Figure. 10(a), *IncRPC* shows better compression performance than *incRCM* as shown in Figure. 10(b) and Figure. 10(c). Meanwhile, the compression effectiveness of the two algorithms on the P2P dataset is inferior to compression effectiveness on the other two datasets.

5) EFFECTIVENESS OF QUERY PROCESSING

In this set of experiments, we evaluate the performance of *HcRPC* for reachability queries on the original and compressed graphs. Meanwhile, $compress_R$ is selected as the compared algorithm. Figure. 11 shows the experimental result run on five different datasets, where Y-axis represents the percentage of reachability query time on compressed graphs obtained by the two algorithms to reachability query time on the original graphs. From Figure. 11, we can see that the reachability query time on compressed graphs is significantly less than the reachability query time on the original graphs. Indeed, the running time of reachability query on the compressed graph is only 6% of the cost on original graphs in average, and the optimal result of the running time of reachability query on compressed graph is 2% on soc-Epinions1 dataset. In addition, the time spent for reachability queries on the compressed graph generated by *HcRPC* is 30% faster than that on the compressed graph generated by $compress_R$ in average, as *HcRPC* can generate smaller compressed graphs than $compress_R$, the smaller the size of

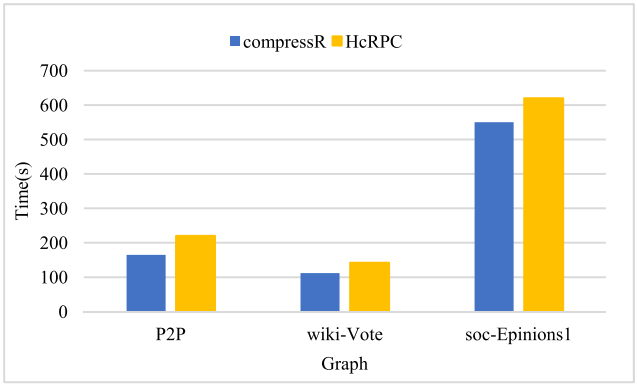


FIGURE 12. Efficiency of reachability compression.

the query graph, the shorter the reachability query time will be.

6) EFFICIENCY OF REACHABILITY COMPRESSION

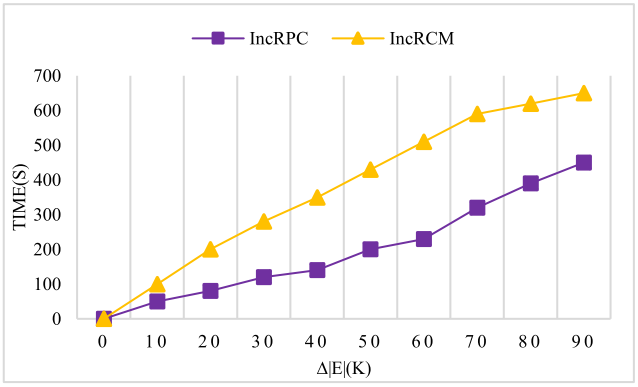
In this set of experiments, we evaluate the efficiency of *HcRPC* via compressing execution time on three datasets, P2P, wiki-Vote and soc-Epinions1. It is not surprising that the compressing execution time of *HcRPC* will be slower than $compress_R$ since *HcRPC* needs to calculate similarities based on the equivalent class obtained by $compress_R$. Fortunately, MinHash adopted in *HcRPC* is a technique for quickly estimating how similar two sets are. As is shown in Figure. 12, the execution time of *HcRPC* is comparable with than of $compress_R$. Although the execution time of *HcRPC* is slightly slower than that of $compress_R$, *HcRPC* can achieve better compression result.

7) EFFICIENCY OF INCREMENTAL REACHABILITY COMPRESSION

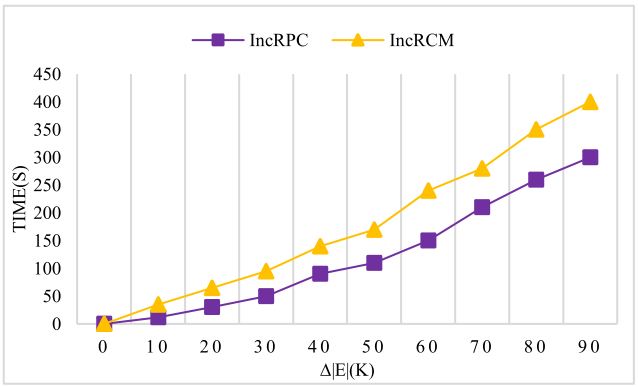
We next evaluate the efficiency of *IncRPC* by execution time of edge updates. Fixing the number of nodes in the social network soc-Epinions1, we vary the number of edges from 519K to 607K by inserting edges in 10K increments and the number of edges from 519K to 429K by deleting edges in 10K decrements. As is shown in Figure. 13(a) and Figure. 13(b), *IncRPC* outperforms *incRCM* in execution time when performing edge insertion updates and edge deletion updates. Intuitively, *IncRPC* runs 17% faster than *incRCM* in the execution time of edge updates in average. The reason is that *IncRPC* does not need to separate each node involved in edge updates to compute the topological ranking values and merge them. It is replaced by computing the joint similarity between the node affected by edge update and its equivalent classes, and the nodes whose joint similarity does not satisfied the threshold θ are separated, thus greatly saving the execution time of edge updates.

8) STORAGE COST ANALYSIS

We then evaluate the storage cost of *HcRPC* on five datasets with three graph types. As is shown in Figure. 14, the storage cost of compressed graphs generated by the two algorithms is significantly lower than that of original graphs. What is more, *HcRPC* has better storage cost on social networks and product co-purchasing networks than $compress_R$ since social networks and product co-purchasing networks are with higher connectivity than P2P and wiki-Talk. That is to say, the edges on social networks and product co-purchasing networks are denser than others. The denser edges of the network, the fewer the corrections are needed. Meanwhile, the storage cost of the compressed graph generated by *HcRPC* is almost the same as that generated by $compress_R$ on the P2P and wiki-Talk networks since few corrections are introduced.



(a). Comparison for edge insertions



(b). Comparison for edge deletions

FIGURE 13. Efficiency of incremental reachability compression.

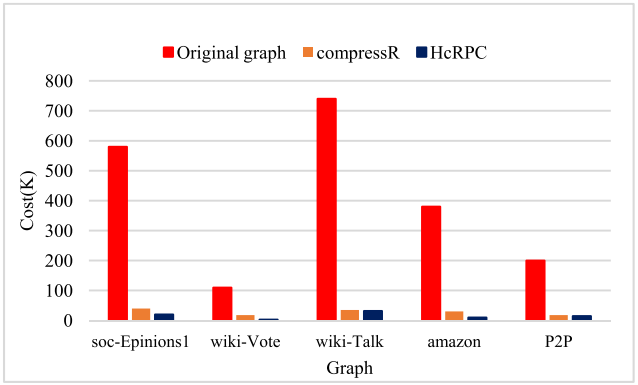


FIGURE 14. Storage cost analysis.

VII. CONCLUSION

In this paper, we introduce a highly compact reachability preserving graph compression algorithm that is able to preserve

the reachability relations between nodes in original graph. For reachability queries, the compressed graphs can be directly queried without decompression, using any available evaluation algorithms for the queries. The highly compressed representation of a given graph G consisting of a compressed graph and a set of corrections, where the set of corrections is introduced to preserve the reachability relations between nodes in graph G. In our reachability preserving compression algorithm, a highly compact compressed graph is obtained via merging a similar pair of equivalent classes when the joint similarity of the pair of equivalent classes exceeds threshold. Meanwhile, we propose an incremental reachability preserving compression algorithm for compressed graphs in order to deal with the dynamic changes of graphs without decompression. Instead of separating each node involved in edge updates, we only compute the joint similarity between the node affected by edge updates and its equivalent classes, and separate the nodes whose joint similarity does not satisfied. The experimental results on real world datasets show that our algorithms are effective and efficient.

REFERENCES

[1] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, 2018, Art. no. 62.

[2] B. Dolgorsuren, K. U. Khan, M. K. Rasel, and Y.-K. Lee, "StarZIP: Streaming graph compression technique for data archiving," *IEEE Access*, vol. 7, pp. 38020–38034, 2019.

[3] S. Anirban, J. Wang, and M. S. Islam, "Multi-level graph compression for fast reachability detection," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2019, pp. 229–246.

[4] N. Kahl, "Graph vulnerability parameters, compression, and quasi-threshold graphs," *Discrete Appl. Math.*, vol. 259, pp. 119–126, Apr. 2019.

[5] S. Maneth and F. Peternek, "Grammar-based graph compression," *Inf. Syst.*, vol. 76, pp. 19–45, Jul. 2018.

[6] D. Zhao, Y. Zhang, J. Lin, W. Song, A. Liotta, and D. Huang, "Query of marine big data based on graph compression and views," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2018, pp. 252–257.

[7] M. Nelson, S. Radhakrishnan, and C. N. Sekharan, "Queryable compression on time-evolving social networks with streaming," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 146–151.

[8] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan, "Queryable compression on streaming social networks," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 988–993.

[9] A. Chatterjee, M. Levan, C. Lanham, M. Zerrudo, M. Nelson, and S. Radhakrishnan, "Exploiting topological structures for graph compression based on quadtrees," in *Proc. 2nd Int. Conf. Res. Comput. Intell. Commun. Netw. (ICRCICN)*, Sep. 2016, pp. 192–197.

[10] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan, "On compressing massive streaming graphs with quadtrees," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct./Nov. 2015, pp. 2409–2417.

[11] M. Purohit, B. A. Prakash, C. Kang, Y. Zhang, and V. S. Subrahmanian, "Fast influence-based coarsening for large networks," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1296–1305.

[12] Y. Mehmood, N. Barbieri, F. Bonchi, and A. Ukkonen, "CSI: Community-level social influence analysis," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Berlin, Germany: Springer, 2013, pp. 48–63.

[13] L. Shi, H. Tong, J. Tang, and C. Lin, "VEGAS: Visual influEnce GrAph summarization on citation networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 12, pp. 3417–3431, Dec. 2015.

[14] Q. Qu, S. Liu, F. Zhu, and C. S. Jensen, "Efficient online summarization of large-scale dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3231–3245, Dec. 2016.

[15] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen, "Sparsification of influence networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 529–537.

[16] C. Dunne and B. Shneiderman, "Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2013, pp. 3247–3256.

[17] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "TimeCrunch: Interpretable dynamic graph summarization," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1055–1064.

[18] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "VoG: Summarizing and understanding large graphs," in *Proc. SIAM Int. Conf. Data Mining*. Philadelphia, PA, USA: SIAM, 2014, pp. 91–99.

[19] G. Buehrer and K. Chellapilla, "A scalable pattern mining approach to Web graph compression with communities," in *Proc. Int. Conf. Web Search Data Mining*, 2008, pp. 95–106.

[20] C. Chen, C. X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, and J. Han, "Mining graph patterns efficiently via randomized summaries," in *Proc. VLDB Endowment*, 2009, vol. 2, no. 1, pp. 742–753.

[21] K. Anyanwu and A. Sheth, "The ρ-operator: Enabling querying for semantic associations on the semantic Web," in *Proc. 12th WWW Conf.*, 2003.

[22] J. van Helden, A. Naim, R. Mancuso, M. Eldridge, L. Wernisch, D. Gilbert, and S. J. Wodak, "Representing and analysing molecular and cellular function using the computer," *Biol. Chem.*, vol. 381, nos. 9–10, pp. 921–935, 2000.

[23] J. X. Yu and J. Cheng, "Graph reachability queries: A survey," in *Managing and Mining Graph Data*. Boston, MA, USA: Springer, 2010, pp. 181–215.

[24] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 157–168.

[25] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proc. 20th Int. Conf. World Wide Web*, 2011, pp. 587–596.

[26] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, "Graph OLAP: Towards online analytical processing on graphs," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 103–112.

[27] N. Hassanlou, M. Shoaran, and A. Thomo, "Probabilistic graph summarization," in *Proc. Int. Conf. Web-Age Inf. Manage.* Berlin, Germany: Springer, 2013, pp. 545–556.

[28] K. LeFevre and E. Terzi, "GraSS: Graph structure summarization," in *Proc. SIAM Int. Conf. Data Mining*. Philadelphia, PA, USA: SIAM, 2010, pp. 454–465.

[29] S. Navlakha, R. Rastogi, and N. Shrivastava, "Graph summarization with bounded error," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 419–432.

[30] K. U. Khan, W. Nawaz, and Y.-K. Lee, "Set-based unified approach for summarization of a multi-attributed graph," *World Wide Web*, vol. 20, no. 3, pp. 543–570, 2017.

[31] Z. Liu, J. X. Yu, and H. Cheng, "Approximate homogeneous graph summarization," *Inf. Media Technol.*, vol. 7, no. 1, pp. 32–43, 2012.

[32] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 567–580.

[33] N. Zhang, Y. Tian, and J. M. Patel, "Discovery-driven graph summarization," in *Proc. IEEE 26th Int. Conf. Data Eng. (ICDE)*, Mar. 2010, pp. 880–891.

[34] H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka, "Compression of weighted graphs," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 965–973.

[35] H. Maserrat and J. Pei, "Neighbor query friendly compression of social networks," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 533–542.

[36] S. Raghavan and H. Garcia-Molina, "Representing Web graphs," in *Proc. 19th Int. Conf. Data Eng.*, Mar. 2003, pp. 405–416.

[37] F. Zhou, S. Malher, and H. Toivonen, "Network simplification with minimal loss of connectivity," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2010, pp. 659–668.

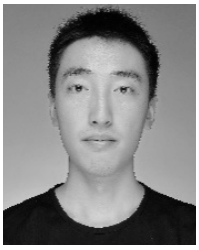
[38] N. Ruan, R. Jin, and Y. Huang, "Distance preserving graph simplification," in *Proc. IEEE 11th Int. Conf. Data Mining*, Dec. 2011, pp. 1200–1205.

[39] A. Sadri, F. D. Salim, Y. Ren, M. Zameni, J. Chan, and T. Sellis, "Shrink: Distance preserving graph compression," *Inf. Syst.*, vol. 69, pp. 180–193, Sep. 2017.

[40] D. M. Moyles and G. L. Thompson, "An algorithm for finding a minimum equivalent graph of a digraph," Manage. Sci. Res. Group, Pittsburgh, PA, USA, Tech. Rep., 1967.

[41] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 131–137, 1972.

[42] T. Feder and R. Motwani, "Clique partitions, graph compression and speeding-up algorithms," *J. Comput. Syst. Sci.*, vol. 51, no. 2, pp. 261–272, Oct. 1995.
[43] S. J. van Schaik and O. de Moor, "A memory efficient reachability data structure through bit vector compression," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 913–924.
[44] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the Web," *Comput. Netw. ISDN Syst.*, vol. 29, nos. 8–13, pp. 1157–1166, Sep. 1997.



RUI BING received the B.E. degree from Northwest Normal University, China, in 2017, where he is currently pursuing the master's degree with the College of Computer Science and Engineering. His general research interest includes graph data mining.

HUIFANG MA received the B.E. degree from Northwest Normal University, China, in 2003, the M.S. degree from Beijing Normal University, China, in 2006, and the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010. She is currently a Professor with the College of Computer Science and Engineering, Northwest Normal University. Her research interests include data mining and machine learning.

XIANGCHUN HE received the B.E., M.S., and D.Ed. degrees from Northwest Normal University, China, in 2003, 2010, and 2019, respectively, where he is currently an Associate Professor with the College of Education Technology. His research interests include data mining and learning analysis.

ZHIXIN LI received the B.S. and M.S. degrees from the Huazhong University of Science and Technology, in 1992 and 2004, respectively, and the Ph.D. degree in computer software and theory from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010. He is currently a Professor with the College of Computer Science and Information Technology, Guangxi Normal University. His research interests include image understanding, machine learning, and multimedia information retrieval. His doctoral dissertation has won the Best Doctoral Dissertation Award of the Chinese Association of Artificial Intelligence, in 2011.

LIJUN GUO received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2011. He is currently a Full Professor with Ningbo University, Ningbo, China. His current research interests include computer vision, intelligence science, and machine learning.