IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 31, 2019, accepted April 15, 2019, date of publication April 22, 2019, date of current version May 1, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2912172

Optimal Representation for Web and Social Network Graphs Based on K^2-Tree

FENGYING LI, QI ZHANG, TIANLONG GU, AND RONGSHENG DONG
Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China

Corresponding author: Fengying Li (lfy@guet.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61762024 and Grant U1501252, and in part by the Natural Science Foundation of Guangxi Province under Grant 2017GXNSFDA198050 and Grant 2016GXNSFAA380054.

ABSTRACT With the rapid growth of the Internet, the scale of graphs has increased dramatically, which poses special challenges in representing both web graphs and social network graphs. In the adjacency matrix of web and social network graphs, only a very small proportion of the elements is "1" s. Furthermore, we find that using the aggregation of scattered 1 s to form a high density of adjacency matrices is beneficial to the compression of storage space. Based on these findings, we propose the DGC-K^2-tree compression approach based on K^2-tree, which can greatly increase the density of 1 s among the existing algorithms and adequately compress the blank area in the adjacency matrix. Then, we design a query algorithm for this mechanism to support the operation on the graph. The experimental results show that compared with the state-of-the-art algorithms, including the K^2-tree based on a diagonal clustering mechanism (K^2-BDC), the K^2-tree, Re-Pair, and LZ78, our approach achieves better compression ratio and shorter time consumption. In terms of storage efficiency, our approach reduces the space by an average of 34.07% compared to the best performing algorithm K^2-BDC. In terms of query efficiency, our approach reduces the time by an average of 80.63% compared to the best performing algorithm LZ78.

INDEX TERMS Web graph, social network graph, compression representation, K^2-tree.

I. INTRODUCTION

Graphs are a natural way of modeling connections in the World Wide Web and social networks [1]. In the World Wide Web, each web page corresponds to a graph node, and each link corresponds to a graph edge. Such a directed graph is called a web graph. In social networks, the population's behavior and attributes are typically represented by social network graphs. Analyzing the structure and data of the graph enables the in-depth mining of network characteristics. The operations over the graph include forward querying, reverse querying, checking the presence of a link, etc. Among them, forward querying is one of the most widely used graph operations. It can be used to determine the connection between two pages or two people, filter out all the pages linked by a specific page, determine a person's communication range, etc.

The most common representations of graph are adjacency matrix and list. For small scale graph data, these two approaches can provide efficient querying. However, with the rapid development of the Internet and the extreme growth of the World Wide Web's scale, graphs are generating at an unprecedented pace and are accumulating a large amount of data. How to analyze and use these data has become a key opportunity and an extreme challenge for many fields. To satisfy the efficient operation of some basic algorithms and operations on large-scale graph data, in recent years, many scholars have designed many data structures with good performance for the compression storage of graphs and proposed algorithms to extend operations on these graphs.

Re-Pair [2], [3] and LZ78 [2], [4] are two classic phrase-based compression algorithms. Both of them compress the redundant information stored in the adjacency list by utilizing the similarity of neighbor sets in graph data. The difference is that LZ78 outputs the dictionary as part of the result during compression. The K^2-tree [5] proposed by Brisaboa et al. is a representative compact data structure for compressing adjacency matrix. It was originally designed to compress web graphs, and further research papers prove that it performs well in other fields.

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

The above approaches mainly compress the graph by using structure characteristics of the adjacency matrix and list. The graph itself also contains significant information. In the web graph, most of the links are intra-domain, and the neighbor sets of the adjacent pages in the lexicographic order are similar. Based on the intrinsic characteristics of the web graph, the algorithm BV achieves a good balance between storage and querying [6], [7]. The algorithm VNM [8] introduces virtual nodes for pretreatment, and further optimizes the BV algorithm. In the social network graph, the friendships between people also show obvious characteristics. There are more friendships in the same community but fewer in different communities. In addition, two people who are friends with each other have mostly the same people in their circle of friends. FRS [9] and BFS [10] propose to reorder the nodes of social network graph, making people in the same community as close as possible. Then compress the redundant information of adjacency matrix or list of the graph. Other literatures including literature [11], [12], have also compressed social network graph effectively according to its characteristics.

Considering the structure characteristics of the adjacency matrix and the intrinsic characteristics of the web graph, Chang et al. put forward an algorithm based on K^2-tree which can efficiently compress graph data [13]. Compared with existing approaches, this mechanism achieves the best space/time tradeoff. Delta-K^2-tree [14] is a good data structure that makes up for the shortcomings of the K^2-tree in representing web graphs. The quadtree is a classic data structure that can be used to represent two-dimensional images, and has been applied to the representation of the graph. Chatterjee et al. introduced a new technology and proposed a general model by considering the topology of the graph [15]. For the streaming graph, the four point tree can also achieve effective compression and query [16]. In addition to the quadtree, Nelson et al. propose a data structure for the streaming graph. This data structure is a good representation for the increase and decrease of edges [17].

Extracting dense subgraphs and representing them in a compact data structure is also an efficient way to compress web graphs and social network graphs. Based on this concept, literature [18], [19] both propose compression structures that uses clustering and mining of dense subgraphs to avoid redundancy in nodes and edge representations of dense subgraphs.

In addition to compressed representation, query processing and optimization of graph data are also the focus of graph data research. As the key to query processing and optimization of graph data, the graph query language has evolved since the GraphLog [20] proposed in 1990. In the early stage, query languages included StruQL [21], UnQL [22] and so on. G-CORE [23] is a closed query language and supports multi-graph query and connection, multi-value attribute processing and other functions. RDBMS [24] proposes four relational algebra operations and supports graph processing at the SQL level.

In the field of graph compression, the compression effect depends on many factors. Literature [25] conducted a comprehensive investigation and classification of lossless graph compression for the first time. This comprehensive analysis can serve as a guide for choosing the best lossless compression scheme in a given setting.

In this paper, the authors will continue efforts to combine the structure characteristics of the adjacency matrix with the intrinsic characteristics of web graphs and social network graphs, and further optimize the K^2-tree. The authors find that when the number of 1s is constant, the storage space of the K^2-tree is smaller under the condition of an adjacency matrix with a smaller scale (in other words, a higher density of 1s). In addition, the clustering mechanism in literature [13] still has room for improvement, which the authors will address and analyze in section II-*C*. As a result, the authors propose a new approach to compactly represent graph data. The experimental results demonstrate that compared with existing approaches, our approach achieves better compression ratio and shorter time consumption.

II. RELATED WORKS

A. GRAPH AND RELATED CONCEPTS

The graph is composed of a finite non-empty set of vertices and a set of edges between vertices. It is usually expressed as *G (V, E)*, where *G* represents a graph, *V* is a set of vertices in graph *G*, *E* is a set of edges in graph *G*. According to the direction of edges, graphs are divided into undirected graphs and directed graphs. In this paper, graphs refer to directed graphs.

In an undirected graph, if there is an edge between two vertices, the two vertices are adjacent to each other. In a directed graph, if there is an arc between two vertices *v* and *v'* (the directed edge is called arc), and *v* points to *v'*, then *v'* is a adjacent node of *v*.

The adjacency matrix is a storage structure of the graph. To completely store the graph, a one-dimensional array represents the information of each vertex, and a two-dimensional array (adjacency matrix) represents the information of the relationship between the vertices. Fig. 1 is a directed graph and the corresponding adjacency matrix.

0	1	1	0	0
1	0	0	1	0
0	0	0	1	0
1	1	0	0	0
0	0	1	1	0

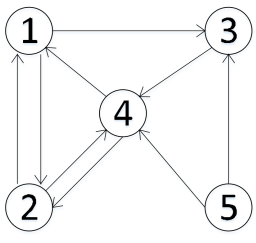

FIGURE 1. A directed graph and the corresponding adjacency matrix.

B. THE K^2-TREE

The K^2-tree is a compact data structure for representing binary relations that takes advantage of large empty areas of the adjacency matrix of web graphs. In the K^2-tree, the main

idea is to represent the adjacency matrix of a web graph by a K^2-ary tree of height $h = \lceil log_K n \rceil$, where n is the number of web pages.

More precisely, in the phase of K^2-tree construction, the adjacency matrix is divided into K^2 equal submatrices. Corresponding to each submatrix, the approach will generate a subnode for the root node. If the elements of the submatrix are all '0's, the corresponding subnode of the K^2-tree is 0; if the submatrix contains at least one 1, the subnode is 1. For those submatrices containing 1s, the approach will divide them recursively into K^2 equal parts until the size of the submatrix is 1×1. During the division, all the subnodes are sorted from top to bottom and from left to right.

In a web graph or a social network graph, regarding to its corresponding adjacency matrix, if the order is not an integral power of K, the approach will add rows and columns 0s to the bottom and right sides of the adjacency matrix until the order of the matrix is an integer power of K.

Fig. 2 shows an example of the adjacency matrix of a web graph with 11 web pages in the left part. The authors set the value of K to 2. The 0s in the gray area are used to complement the adjacency matrix into a 16-order matrix. The 2^2-tree of the adjacency matrix is shown in the right part. In this 2^2-tree, the 1s (except the nodes in the lowest level) indicate that the corresponding submatrix contains 1s and the 0s (except the nodes in the lowest level) indicate that all elements in the corresponding submatrix are 0s. The 0s and 1s in the lowest level represent the values of the corresponding elements in the adjacency matrix. After completing the construction of the tree, we can represent the adjacency matrix via two bit vectors, T and L. T stores nodes except those in the lowest level by traversing the 2^2-tree level following a top-down and a left-right order. L stores nodes in the lowest level from left to right.

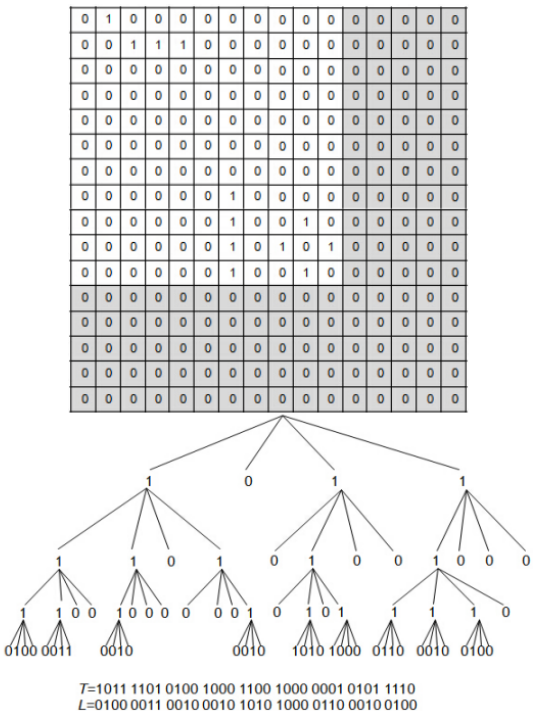

FIGURE 2. Adjacency matrix with 11 nodes and the corresponding K^2-tree with $K = 2$.

In addition to efficient storage rates, the K^2-tree also supports multiple operations on graphs [26]. Specifically, by performing a rank operation on the T and L vectors, we can retrieve direct or reverse neighbors over an adjacency matrix, where the operation rank *(T, x)* is performed to calculate the number of 1s in the former x bits in vector T. Furthermore, we can also realize range searches and check individual links.

However, the K^2-tree still has some limitations in representing web graphs: (1) the tight coupling between nodes might be destroyed, which will increase the time consumption of the operations on the graph; (2) a large area filled with 0s in the matrix might be divided into different submatrices so that it cannot be completely compressed; (3) the height of the K^2-tree will gravely affect the time consumption of operations on the graph [13]. These limitations of K^2-tree provide ideas for follow-up research.

C. THE K^2-TREE BASED ON DIAGONAL CLUSTERING MECHANISM

The K^2-BDC is a further improvement of the K^2-tree. Aiming at the limitations of K^2-tree mentioned in section II-*B*, K^2-BDC has made great progress in maintaining the coupling between nodes and reducing the height of K^2-tree. The main ideas of K^2-BDC are as follows: (1) Find areas along the principal diagonal in the adjacency matrix that contain a relatively large number of 1s (name these areas as clusters), then represent each cluster with a K^2-tree; (2) Fill the areas corresponding to all the clusters in the original adjacency matrix with 0s and represent the resulting adjacency matrix (name this matrix the zero-matrix) with a K^2-tree; (3) The compressed representation of this graph is the integration of all K^2-trees generated in the previous steps.

The K^2-BDC is a good approach to solve some problems of K^2-tree and the performance in storage and query has been significantly improved. However, this approach still has room for improvement in the following areas.

(1) Only clusters on the principal diagonal are clustered. In the adjacency matrix, the area deviating from the principal diagonal also contains a relatively large number of 1s that have not been extracted via the clustering mechanism.

(2) The extracted clusters are still sparse and do not achieve the desired compression efficiency. The basic idea of this approach is to extract the relatively dense areas on the principal diagonal for a separate operation. However, in real-world graphs, 1s in the clusters are still very sparse. In the literature, the outlier rate of the experiment is 0.00003, which indicates the sparse degree of clusters.

(3) Due to the introduction of DACs coding technology, the operation burden might increase. Because vector L is encoded with DACs, the operations on the K^2-tree require

a decoding operation of vector L, which increases the time consumption.

(4) The height of the K^2-tree has not been effectively reduced, and this result has little effect on reducing the time consumption of operations. The clusters extracted by this approach are located on the main diagonal. Therefore, when querying all neighbors of a node, we need to query one cluster and the zero-matrix. With regard to the K^2-tree representing the zero-matrix, its height is not decreased, so the query efficiency has not been prominently improved.

D. THE QUADTREE

The quadtree is originally designed to represent the black and white images. This data structure is typically used to represent two dimensional images by recursively dividing the space into four quadrants.

(1) Convert the image to a binary matrix. The black areas in the images are filled with 1s and the white areas are filled with 0s.

(2) Divide the binary matrix. The matrix is divided into 4 equal quadrants. For any quadrant, if all the elements in the quadrant are the same (all 0s or all 1s), the division is not continued; otherwise, the division continues until all the elements in the quadrant are the same or only one element is contained in the quadrant.

(3) Represent the divided matrix into a quadtree. If the value of a node in the tree is 1, the quadrant corresponding to this node needs to be divided into 4 small quadrants. If this quadrant does not need to be divided, the corresponding node will contains two numbers. The first number is 0, indicating that this quadrant does not need to be divided; the second number is 0 or 1, indicating that all elements in this quadrant are 0s or 1s.

Fig. 3 shows the process of converting a two dimensional image into a quadtree. Since the adjacency matrix is similar to the representation of the images, the quadtree can also be used as a compressed representation of the graph data. There are various approaches for applying the quadtree to graph data compression.

The idea of compressing graph data using a quadtree is similar to the K^2-tree when $K = 2$. However, dividing the original adjacency matrix of the graph data only into four parts is likely to fail to achieve efficient compression of multiple graph data [27]. Therefore, compared with the quadtree, the K^2-tree has better flexibility and adaptability.

E. THE CHARACTERISTICS OF WEB GRAPHS AND SOCIAL NETWORK GRAPHS

The web graph is a mapping of web page relationships in the World Wide Web, and the social network graph is a mapping of person relationships in a community. There are several similarities between them:

(1)The sparsity: The World Wide Web and the social networks contain a huge number of web pages and people. However, the average number of links per page or friends per person is very small.

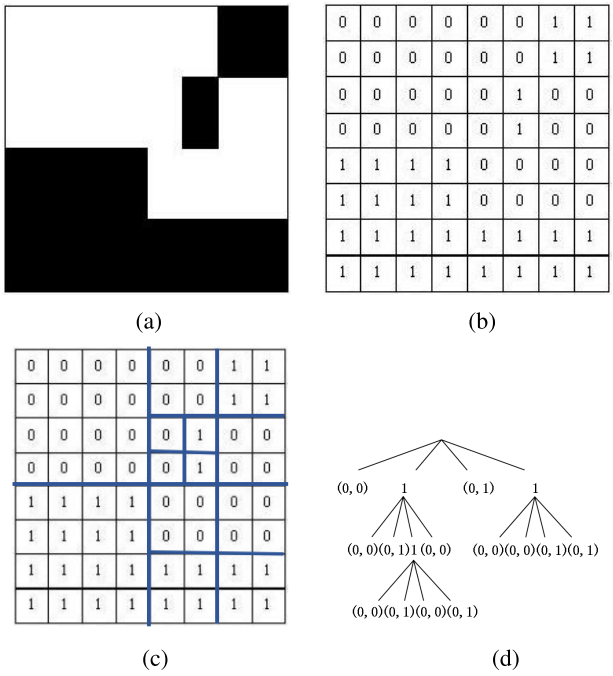

FIGURE 3. The process of converting a two dimensional image into a quadtree. (a) A two dimensional images. (b) The binary matrix of (a). (c) The divided matrix of (b). (d) The quadtree of (c).

(2)The similarity of neighbor sets: For web graphs, the neighbor sets of adjacent pages are similar. For social network graphs, there are more friendships in the same community and fewer friendships in different communities.

Based on the above findings, we have got the following ideas to make up for these limitations. Divide the adjacency matrix of the graph into submatrices, then aggregate the 1s of these submatrices to form small matrices. Next, represent these small matrices as K^2-trees. These K^2-trees are a representation of the graph. However, this idea destroys the tight coupling between the nodes of the graph and might increase the time consumption of the operations on the graph. To solve this problem, we design an algorithm to establish the connection between the nodes and these small matrices, and further shorten the time consumption.

III. OPTIMAL REPRESENTATION AND QUERYING OF THE GRAPH

A. DIVISION AND REPRESENTATION

In the field of graph data representation, when the number of 1s is constant, the storage space of the K^2-tree is smaller under the condition of an adjacency matrix with a smaller scale (in other words, a higher density of 1s). For example, the storage space required to represent the K^2-tree of a second-order adjacency matrix containing four 1s is 4bits, and the storage space required to represent the K^2-tree of a eight-order adjacency matrix containing four 1s is at least 12bits. This conclusion provides a good research direction for the compression representation of the graph.

Algorithm 1 Algorithm for Division and Representation

Require: an adjacency matrix M; an integer *Size* that specifies the size of large submatrices; an integer K that specifies the parameter of the K^2-tree.

Ensure: a queue *list* of recombination matrices; two sequences of bit vectors; a sequence of queues.

```
list := empty queue;
list1 := empty queue;
list2 := empty queue;
small_Size := K; /*the size of small submatrices*/
large_Size := K; /*the size of large submatrices*/
use parameter small_Size to divide M;
number1 := the number of small submatrices;
for i = 0 to number1 − 1 do
    if the i-th small submatrix contains 1 then
        the i-th small submatrix → list1; /*map all 1s into
        the corresponding small submatrices*/
    end if
end for
use parameter large_Size to divide M;
number2 := the number of large submatrices;
for i = 0 to number2 − 1 do
    if the i-th large submatrix contains 1 then
        the i-th large submatrix → list2; /*map all 1s into
        the corresponding small submatrices*/
        for j = 0 to number2 − 1 do
            if list1[j] ∈ list2[i] then
                list1[j] → S_i; /*S_i stores all the small submatrices contained in the i-th large submatrix*/
            end if
        end for
        M_recombine := the matrix that is recombined by
        the matrices in S_i;/*M_recombine is the recombination matrix corresponding to the i-th large matrix*/
        M_recombine → list;
        represent the M_recombine by K^2-tree and store it
        into bit vectors T_i and L_i;
    end if
end for
return list, two sequences of bit vectors T and L,
the sequence of queues S.
```

1) ALGORITHM

Taking adequate advantage of the sparseness of the web graphs, the authors propose a mechanism based on K^2-tree to recombine all 1s into multiple matrices with higher density of 1s. Finally, these recombination matrices are represented as K^2-trees for compression storage. We call the modified K^2-tree DGC-K^2-tree. The authors explain the overall process of the mechanism in the **Algorithm for division and representation**. Assume for simplicity and clarity that the authors illustrate this algorithm by taking a 16×16 adjacency matrix as shown in Fig. 4 and specify $K = 2$.

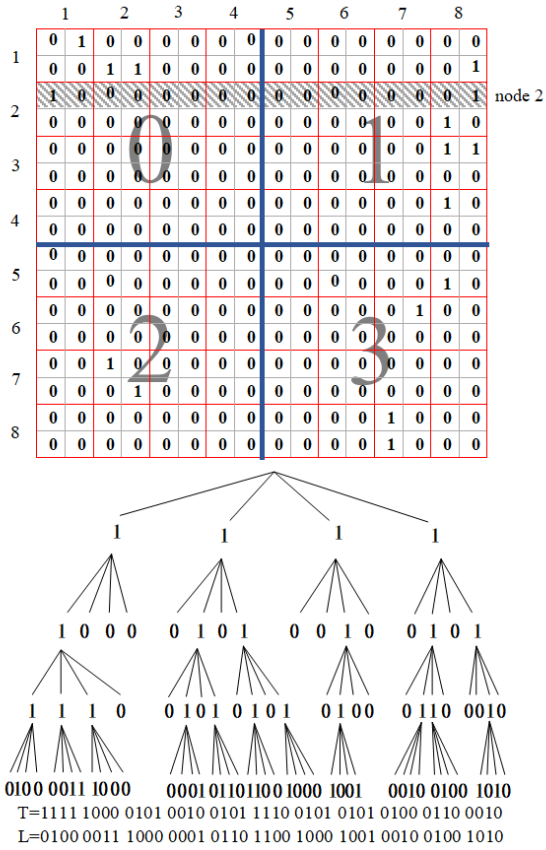

FIGURE 4. Adjacency matrix with 16 nodes and the corresponding K^2-tree with $K = 2$.

STEP1 Divide the adjacency matrix M into eight submatrices of 2×2. Then, label the submatrices that contain at least one 1 by encoding the position of these submatrices in the original matrix and storing the matrix codes into *list*1, where *list*1 = {(1, 1), (1, 2), (1, 8), (2, 1), (2, 8), (3, 8), (4, 8), (5, 8), (6, 7), (7, 2), (8, 7)}.

STEP2 Similar to the first step, divide the adjacency matrix again into four submatrices of 8×8 and then label the submatrices that contain at least one 1 by encoding the position of these submatrices in the original matrix and storing the matrix codes into *list*2, where *list*2 = {(1, 1), (1, 2), (2, 1), (2, 2)}.

STEP3 Given any matrix R in *list*2, extract the submatrices that belong to *list*1 and recombine them into a new matrix *Ms*. The size of *Ms* is $2n \times 2n$ ($n = \lceil \sqrt{x} \rceil$), where x is the number of submatrices belonging to *list*1 within the boundaries of R). Then, fill the incomplete part of R with 0s.

It is worth noting that during the second division process, the submatrix size needs to be appropriately selected to ensure that a submatrix generated after this division contains more submatrices of *list*1, and ensure that the recombination matrix will not be too large because the height of its corresponding K^2-tree is expected to be reduced.

After division, all the 1s in the adjacency matrix have been mapped to the corresponding positions of the four recom-

bination matrices. Next, the authors' work is to represent the four recombination matrices as K^2-trees. Since the K^2-tree construction process has been described in section II-*B*, the authors omit the detailed process here. Finally, the authors obtain the corresponding bit vectors *T* and *L* and the queues *S* of the small submatrices contained in each recombination matrix.

Fig. 5 shows the K^2-trees with $K = 2$ and *T*/*Ł* vectors of the recombination matrices. In the actual experiments, the storage space of *list1* and *list2* is much smaller than the storage space required to represent the K^2-tree, so the authors selectively ignore the storage space of *list1* and *list2* when calculating the storage space. The storage space required to represent the K^2-tree of the original adjacency matrix is 88 bits, and the storage space required to represent the DGC-K^2-tree is 56 bits. Evidently, DGC-K^2-tree can compress the storage space drastically by eliminating the blank area in the adjacency matrix to increase the density of 1s. Utilizing this characteristic, DGC-K^2-tree can achieve very efficient compression efficiency.

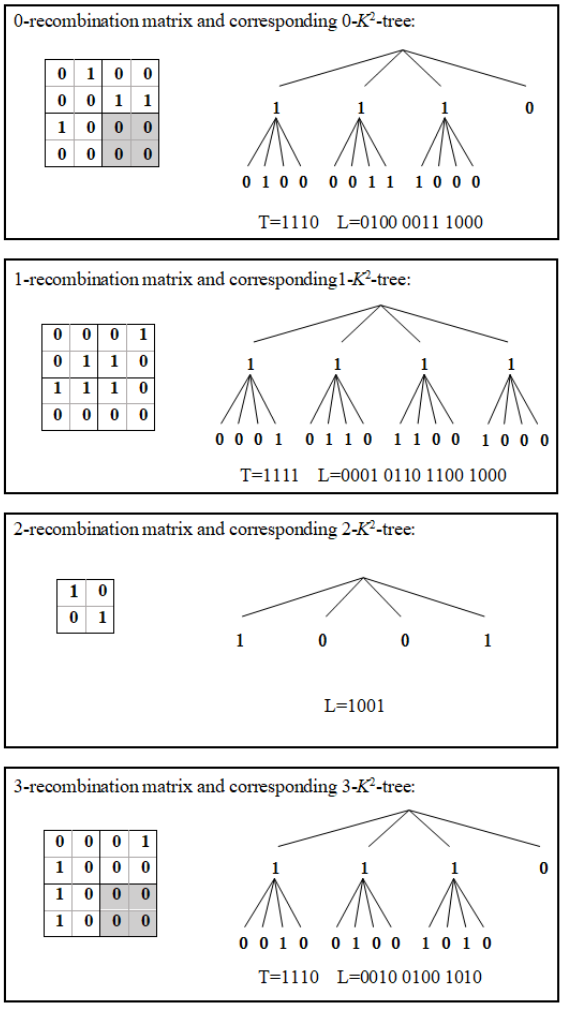

FIGURE 5. 4 recombination matrices and their corresponding K^2-trees with $K = 2$.

2) SPACE COMPLEXITY ANALYSIS

Assume the graph has *n* pages and *m* links. Since the detailed analysis of space complexity in the worst case can be found in the literature [5], the authors do not repeat the analysis here. In the original K^2-tree, the total space in the worst case is $k^2 m(log_{k^2}(n^2/m) + O(1))$ bits.

In the worst case, DGC-K^2-tree cannot compress the space, and it takes up the same space as the original K^2-tree algorithm. Assume the size of large submatrices is *s*. The number of graphs obtained after division is $num = (n/s)^2$. Since the *m* links are not uniformly distributed, in the worst case, the total space is $\sum_{i=1}^{num} k^2 m_i(log_{k^2}(n^2/m_i) + O(1))$ bits, where m_i is the number of links included in the *i*-th graph obtained after division.

On the real data set, there are large blank areas in the graph, so DGC-K^2-tree can achieve better compression effect than the worst case.

B. QUERY OPERATION

1) ALGORITHM

Aggregating 1s of the adjacency matrix into the recombination matrices can reduce the height of K^2-tree. However, the density of 1s of the recombination matrices is thousands times of the density of 1s of the original adjacency matrix. This results in an increase in the number of 1s contained in the K^2-trees of the recombination matrices. Accordingly, the number of recursions during queries will increase and the query will be more time-consuming. The authors design a query algorithm to solve this problem.

Since the recombination matrix is composed of multiple scattered submatrices of the original matrix, given a node *N* during the query process, we need some additional data to assist the query. These data have been obtained during the division and presentation process. The authors explain the overall process of the mechanism in the **Algorithm for querying neighbors of a node**. To facilitate understanding, the authors use the figures in Fig. 4, Fig. 5 and Fig. 6 as examples to illustrate the query process.

STEP1 Determine which recombination matrices need to be visited and obtain the new node numbers of node *N* and the new column numbers needed in the process. First, specify a node $N = 2$. Node 2 is included in the 0,1 submatrices, so we need to visit 0,1-recombination matrices. Then, when visiting the 0-recombination matrix, select the matrices containing node 2 in S0 and store them in *list'*, where $list' = \{(2, 1)\}$. The *n* of the 0-recombination matrix is 2, and (2,1) is the third matrix in S_0, so the new node number of node 2 is 2, and its corresponding new column number to be queried in the 0-recombination matrix is 0,1. the column offset of the 0-recombination matrix is 0 and the column offset of matrix (2,1) is 0, so the column offset of the new column number is 0.

STEP2 Visit the K^2-tree on the basis of the new node number and the new column number, and then the

Algorithm 2 Algorithm for Querying Neighbors of a Node

Require: the node number N of a given node; the queue *list* of the recombination matrices; the sequence of bit vectors T and L, the sequence of queues S.

Ensure: a queue *list_neighbor* of node N's direct neighbors.

```
1: list' := empty queue;
2: list_neighbor := empty queue;
3: list_column := empty queue;
4: i := 0;
5: while N ∈ list[i] do
6:   for j = 0 to S_i.length − 1 do
7:     if N ∈ S_i[j] then
8:       S_i[j] → list'; /*list' stores the recombination
         matrices containing node N*/
9:       node := the new node number to be queried in
         list[i] corresponding to the old node number in
         S_i[j];
10:      calculate the new column numbers N' needed in
         the process of querying N's neighbor;
11:      N' → list_column;
12:      column_offset:= the column offset of S_i[j];
13:      for m = 0 to K − 1 do
           /*simply query the value of the correspond-
           ing elements in the small submatrix containing
           node N*/
14:        result := the value of Cell(node,
           list_column[m]) in list[i];
15:        if result = 1 then
16:          neighbor := column_offset + m;
17:          neighbor → list_neighbor;
18:        end if
19:      end for
20:    end if
21:  end for
22:  i := i + 1;
23: end while
24: return list_neighbor.
```

neighbor node number with offset is obtained. For node 2, the first two elements of node 2 in the 0-K^2-tree need to be visited, and the results are 1 and 0. Therefore, in the 0 submatrix, the neighbor of node 2 is numbered 0 and it is the 0-th element in *list_column*. Add 0 to the *column_offset*, and we can obtain a real neighbor node number 0 of node 2.

STEP3 Imitate the above method to obtain the neighbor node numbers in the other submatrices. When visiting the 1-recombination matrix, $list' = \{(2, 8)\}$, $n = 2$ and (2,8) is the second matrix in S_1, so the new node number of node 2 is 0 and its corresponding new column numbers to be queried in the 1-recombination matrix are 2,3. The *column_offset* of the new column number is 14 (the sum of 8 and 6, where 8 is the offset of 1 submatrix and 6 is the column offset of (2,8) in 1 submatrix). Then visit the 1-K^2-tree to obtain the other neighbors. Here, the authors omit the specific steps and obtain the neighbor in the 1 submatrix as 15. After visiting two K^2-trees, we finally calculate the set of neighbors of node 2 to be $list\_neighbor = \{0, 15\}$.

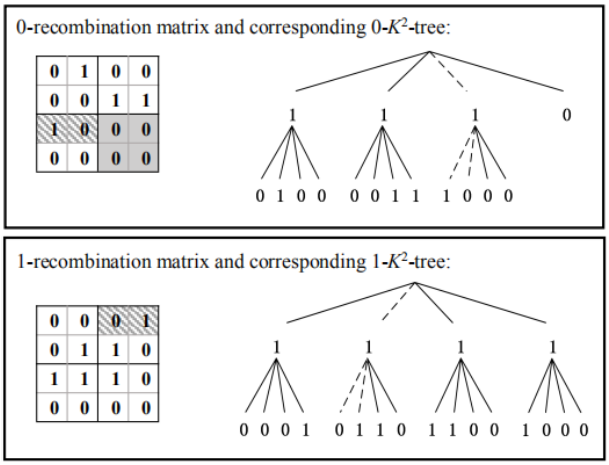

FIGURE 6. The recombination matrices and K^2-trees for querying neighbors of node 2.

Another major reason is that the DGC-K^2-tree reduces the execution frequency of recursive visits to the K^2-tree. For the sake of intuition, let us revisit the adjacency matrix in Fig. 4. To obtain the neighbors of node 2, we need to visit the K^2-tree 25 times. Processed by DGC-K^2-tree, this adjacency matrix is represented by four K^2-trees in Fig. 5, and since we have extracted the set of corresponding column numbers to be queried from node 2 during the query process, the number of visits will be reduced when traversing the K^2-trees. For example, when querying the neighbor of node 2 in the 0-recombination matrix, we first calculate the set of column numbers that need to be queried, that is, {0,1}. Therefore, in Fig. 6, we only need to visit the elements in the K^2-tree corresponding to the shaded position in the recombination matrix dispensing with recursive visits to the other elements, and these two K^2-trees need to be visited only 10 times in total.

2) TIME COMPLEXITY ANALYSIS

Again, since the detailed analysis of time complexity in the worst case can be found in the literature [5], the authors do not repeat the analysis here. Assume the m links are uniformly distributed. In the worst case, the path to each element in the K^2-tree is unique, so there is no need to divide the adjacency matrix. In this case, the time complexity is as same as the K^2-tree, which is $O(\sqrt{m})$.

As mentioned in literature [5], if the matrix is clustered, actually, the graph has the similarity of neighbor sets. As a result, DGC-K^2-tree can perform better than the worst case.

IV. ALGORITHM COMPLEXITY

The complexity of DGC-K^2-tree is also an important index to measure the algorithm. For a graph with n pages and m

links, in the worst case, the total space required by K^2-tree is $k^2m(log_{k^2}(n^2/m) + O(1))$ bits [5]. When querying, K^2-tree takes $O(\sqrt{m})$, which is the same as the time required by DGC-K^2-tree in the worst case [5]. Compared with K^2-tree, DGC-K^2-tree needs to query multiple reorganization matrices separately. However, the height of the K^2-tree corresponding to the recombination matrices is much smaller than that of the original K^2-tree. Therefore, although the number of K^2-trees has increased, reducing the height of the tree has also effectively reduced the query time.

For Re-pair, application of a periodic compaction process allows the total memory space during encoding to be restricted to $5n + 4k^2 + 4k' + \lceil\sqrt{n}\rceil$ words, where n is the number of symbols in the source message, k is the cardinality of the source alphabet; and k' is the cardinality of the final dictionary [3]. In addition, its time complexity can achieve linear time [3]. LZ78 is an improvement on Re-pair. In terms of compression efficiency, LZ78 is inferior to Re-pair. But LZ78 can query quickly in terms of query efficiency, and its time complexity is $O(\lceil r/k\rceil(h + logk))$ average, where r is the number of executions of replacements, k is the number of frequent pairs, $h = O(n)$, and $n = |V|$, where V is the set of vertices [2].

V. EXPERIMENTS

A. EXPERIMENTAL ENVIRONMENT AND TEST DATA

Our experiments are executed on Windows 7 Professional(64 bits) with Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz and 4GB RAM. All experiments use only one CPU core and the compiler is gcc version 4.9.2.

Our data sets are for real web and social network graphs which are directed. They can be viewed and downloaded on the website of the LAW laboratory [28] from the University of Milan. Table 1 shows the main characteristics of the data sets, including the name and field of data set, the number of nodes and edges, the graph density, and the size of a plain adjacency list representation of the graph (using 4-byte integers).

TABLE 1. The main characteristics of the data sets.

Data set	Field	Nodes	Edges	Graph density	Size /MB
cnr-2000	Web	325,557	3,216,152	3.034×10^{-5}	14
eu-2005	Web	862,664	19,235,140	2.585×10^{-5}	77
enron	Social network	69,244	276,143	5.759×10^{-5}	1.3
dblp-2010	Social network	326,186	1,615,400	1.518×10^{-5}	7.4

Graph density is an important index to measure the sparsity of the graph. It is worth mentioning that both the web graphs and the social network graphs are directed graphs, so in this paper, the graph density is defined as:

$$D = \frac{|E|}{|V|(|V|-1)} \tag{1}$$

where E is the number of edges and V is the number of nodes in the graph. The graph density of the data set is given in Table 1, which provides a reliable reference for the sparsity of the graph data.

In addition, the authors visualize the cnr-2000 and enron data sets separately. The coordinates of the x-axis and the y-axis are node numbers, and the elements with value of 1 in the matrix are mapped to corresponding points in the coordinate system. In Fig. 7, 1s are relatively dense in the small area, and most of the adjacent 1s are clustered in the same area.

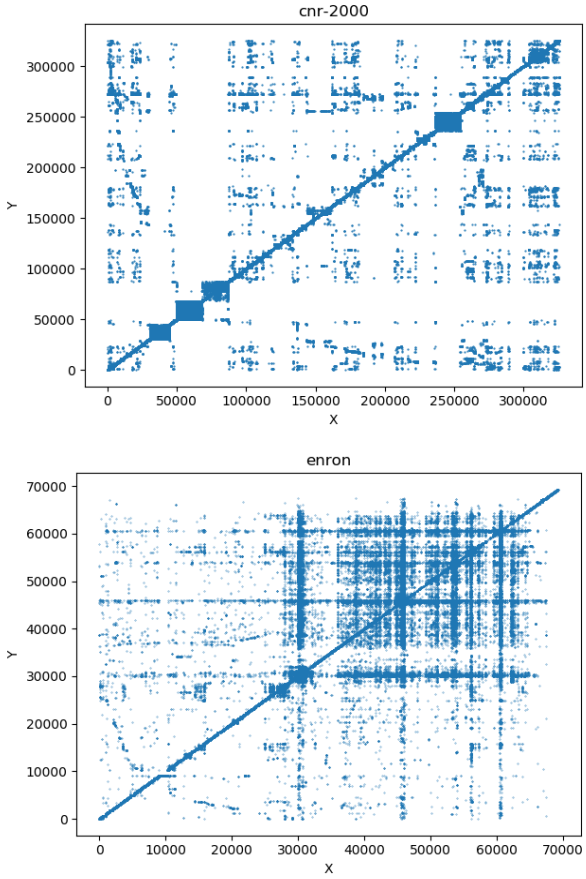

FIGURE 7. The visualization of 1s in data sets cnr-2000 and enron.

The authors use the C language to implement the algorithm. Then, the authors choose four representative and advanced algorithms, including the K^2-BDC [13], K^2-tree [5], Re-Pair [2] and LZ78 [2]. The source codes of K^2-tree, Re-Pair and LZ78 was downloaded from the website of the University of Chile's FCWGR (Fast Compact Web Graph Representations) project [29]. The source codes of K^2-BDC is obtained by the authors of literature [13]. In the same experimental environment, these algorithms are compared to DGC-K^2-tree in terms of both storage and query performances. These two performances are the most critical factors to evaluate the quality of a compression representation algorithm.

B. STORAGE SPACE COMPARISON

In the experiment, the authors use $K = 2$ to configure all K^2-trees. Space is measured as a ratio of bits per edge

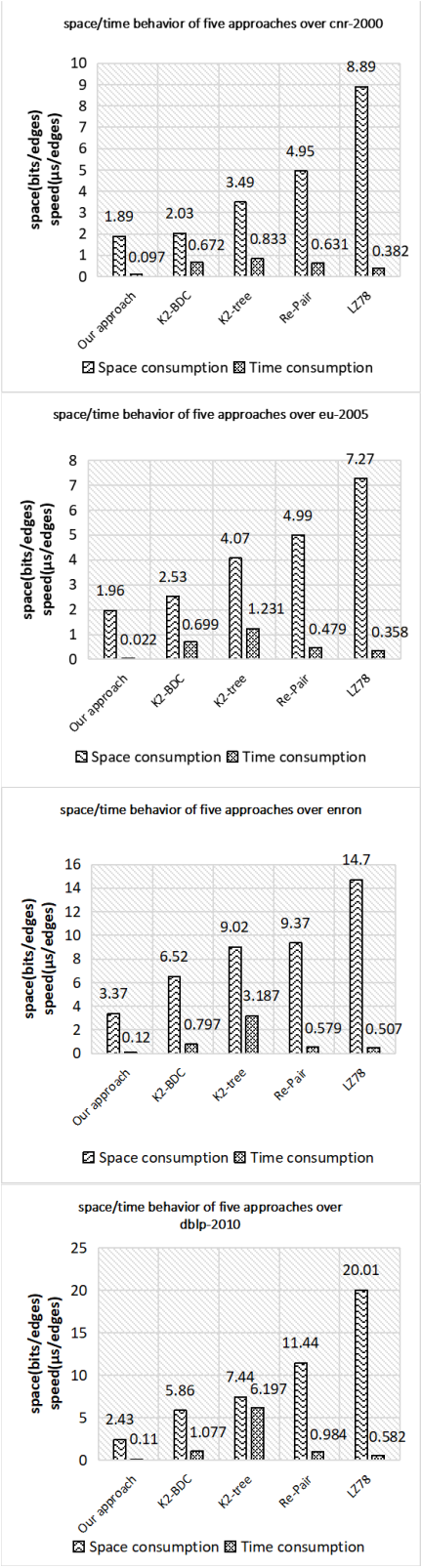

FIGURE 8. The space/time behavior of five approaches over cnr-2000, eu-2005, enron and dblp-2010.

by dividing the total space of the consumption for storing compression representation by the number of edges in the graphs.

Fig. 8 shows the storage space consumption based on the four data sets for the five approaches. Obviously, DGC-K^2-tree can significantly reduce the space consumption over all four data sets. Compared to the LZ78, the authors' approach reduces the storage space by 78.74%, 73.04%, 77.07% and 87.86% in space over cnr-2000, eu-2005, enron and dblp-2010, respectively. More conspicuously, compared to the K^2-BDC, DGC-K^2-tree shows reductions of 6.90%, 22.53%, 48.31% and 58.53%.

C. QUERY SPEED COMPARISON

For the query speed, the authors use the operation for finding all neighbors of a given node in nanoseconds per edge by dividing the total time of the consumption for querying all neighbors by the number of edges in the graph.

Fig. 8 shows the query time consumption over the four data sets. Obviously, compared with other algorithms, DGC-K^2-tree achieves the best execution speed. Let us focus on LZ78 with the best query performance. Compared to LZ78, DGC-K^2-tree achieves reductions of 74.61%, 93.85%, 76.47% and 77.59% in time consumption over cnr-2000, eu-2005, enron and dblp-2010, respectively. Compared to K^2-tree, DGC-K^2-tree achieves reductions of 88.36%, 98.21%, 96.23% and 98.22% in time consumption over cnr-2000, eu-2005, enron and dblp-2010, respectively.

The experimental results show that DGC-K^2-tree achieves the better compression ratio and the shorter time consumption in both storage and querying. In addition, since DGC-K^2-tree is extended and implemented on the basis of the K^2-tree, it supports all operations of the K^2-tree theoretically, including query links, direct neighbors and reverse neighbor queries, etc. From the perspective of the current situation and future development, DGC-K^2-tree has superiorities among the existing algorithms.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, the authors propose a mechanism combined with the K^2-tree that can adequately compress the blank space in the adjacency matrix. The main idea of DGC-K^2-tree is to aggregate the scattered 1s in the adjacency matrix into high-density regions and use the K^2-tree to represent them. Furthermore, the authors design a query algorithm for this mechanism to support the operations on the graph. The experimental results show that DGC-K^2-tree achieves better compression ratio and shorter time consumption among the existing algorithms.

In future work, the authors will further improve the approach and expand its scope of application. In the course of the experiment, the authors find that some approaches are only applicable to a class of graphs such as web graphs or social network graphs. The characteristics of such graphs are the clear demarcation of dense and sparse regions and the coupling of adjacent nodes. However, DGC-K^2-tree does not depend on this kind of coupling and regional demarcation, so it may have a broader application in theory. In future work, the authors will progress in this direction. In the next step, the authors plan to compare DGC-K^2-tree separately with

other algorithms in two directions of web graphs and social network graphs. Through a series of comparative experiments, the authors will find out which fields this approach is more suitable for. In addition, the authors will also develop the approach to support more graph operations.

REFERENCES

[1] F. Claude and S. Ladra, "Practical representations for Web and social graphs," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, New York, NY, USA, Oct. 2011, pp. 1185–1190.
[2] F. Claude and G. Navarro, "Fast and compact Web graph representations," *ACM Trans. Web*, vol. 4, no. 4, pp. 16:1–16:31, Sep. 2010.
[3] N. J. Larsson and A. Moffat, "Off-line dictionary-based compression," *Proc. IEEE*, vol. 88, no. 11, pp. 1722–1732, Nov. 2000.
[4] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, Sep. 1978.
[5] N. R. Brisaboa, S. Ladra, and G. Navarro, "k^2-trees for compact Web graph representation," in *String Processing and Information Retrieval*. Berlin, Germany: Springer, 2009, pp. 18–30.
[6] P. Boldi and S. Vigna, "The Webgraph framework I: Compression techniques," in *Proc. 13th Int. Conf. World Wide Web*, New York, NY, USA, May 2004, pp. 595–602.
[7] P. Boldi and S. Vigna, "The WebGraph framework II: Codes for the world-wide Web," in *Proc. DCC*, Snowbird, UT, USA, Mar. 2004, p. 528.
[8] G. Buehrer and K. Chellapilla, "A scalable pattern mining approach to Web graph compression with communities," in *Proc. Int. Conf. Web Search Data Mining*, New York, NY, USA, Feb. 2008, pp. 95–106.
[9] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, Jun. 2009, pp. 219–228.
[10] A. Apostolico and G. Drovandi, "Graph compression by BFS," *Algorithms*, vol. 2, no. 3, pp. 1031–1044, Sep. 2009.
[11] C. Hernández and G. Navarro, "Compression of Web and social graphs supporting neighbor and community queries," in *Proc. 5th ACM Workshop Social Netw. Mining Anal.*, NY, USA, Aug. 2010, pp. 1–10.
[12] R. Raman, V. Raman, and S. S. Rao, "Succinct indexable dictionaries with applications to encoding k-ary trees and multisets," *ACM Trans. Algorithms*, vol. 3, no. 4, p. 43, Nov. 2007.
[13] L. Chang, X. Zeng, Z. Xu, J. Qian, T. Gu, and H. Song, "Optimal representation of large-scale graph data based on K^2-tree," *Wireless Pers. Commun.*, vol. 95, no. 3, pp. 2271–2284, Aug. 2017.
[14] Y. Zhang, G. Xiong, Y. Liu, M. Liu, P. Liu, and L. Guo, "Delta-K^2-tree for compact representation of web graphs," in *Proc. Asia-Pacific Web Conf.*, Beijing, China, 2014, pp. 270–281.
[15] 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)*, Kolkata, India, Sep. 2016, pp. 192–197.
[16] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan, "On compressing massive streaming graphs with Quadtrees," in *Proc. IEEE Int. Conf. Big Data*, Santa Clara, CA, USA, Nov. 2015, pp. 2409–2417.
[17] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan, "Queryable compression on streaming social networks," in *Proc. IEEE Int. Conf. Big Data*, Boston, MA, USA, Dec. 2017, pp. 988–993.
[18] C. Hernández and G. Navarro, "Compressed representation of Web and social networks via dense subgraphs," in *Proc. Int. Symp. String Process. Inf. Retr.*, 2012, pp. 264–276.
[19] C. Hernández and G. Navarro, "Compressed representations for Web and social graphs," *Knowl. Inf. Syst.*, vol. 40, no. 2, pp. 279–313, Aug. 2014.
[20] M. P. Consens and A. O. Mendelzon, "GraphLog: A visual formalism for real life recursion," in *Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. Princ. Database Syst.*, Nashville, TN, USA, 1990, pp. 404–416.
[21] M. Fernández, D. Florescu, A. Levy, and D. Suciu, "Declarative specification of Web sites with S," *VLDB J. Int. J. Very Large Data Bases*, vol. 9, no. 1, pp. 38–55, Mar. 2000.
[22] P. Buneman, M. Fernández, and D. Suciu, "UnQL: A query language and algebra for semistructured data based on structural recursion," *VLDB J.*, vol. 9, no. 1, pp. 76–110, Mar. 2000.
[23] R. Angles *et al.*, "G-CORE: A core for future graph query languages," in *Proc. Int. Conf. Manage. Data*, Houston, TX, USA, May 2018, pp. 1421–1432.
[24] K. Zhao and J. X. Yu, "All-in-One: Graph processing in RDBMSs revisited," in *Proc. ACM Int. Conf. Manage. Data*, Chicago, IL, USA, May 2017, pp. 1165–1180.
[25] M. Besta and T. Hoefler. (2018). "Survey and taxonomy of lossless graph compression and space-efficient graph representations." [Online]. Available: https://arxiv.org/abs/1806.01799
[26] N. R. Brisaboa, S. Ladra, and G. Navarro, "Compact representation of Web graphs with extended functionality," *Inf. Syst.*, vol. 39, no. 1, pp. 152–174, Jan. 2014.
[27] Q. Shi, YH. Xiao, N. Bessis, YQ. Lu, YL. Chen, and R. Hill, "Optimizing K^2 trees: A case for validating the maturity of network of practices," *Comput. Math. Appl.*, vol. 63, no. 2, pp. 427–436, Jan. 2012.
[28] *The LAW Laboratory From the University of Milan*. Accessed: Mar. 2018. [Online]. Available: http://law.di.unimi.it/datasets.php/s
[29] *The University of Chile's FCWGR (Fast Compact Web Graph Representations) Project*. Accessed: Apr. 2018. [Online]. Available: http://webgraphs.recoded.cl/

FENGYING LI received the B.S. and M.S. degrees from the Guilin University of Electronic Technology, China, in 1994 and 2002, respectively, and the Ph.D. degree from Xidian University, in 2011. She is currently an Associate Professor with the School of Computer Science and Information Security, Guilin University of Electronic Technology. Her research interests include symbolic computing, formal methods, and Petri net theory.

QI ZHANG is currently pursuing the master's degree with the School of Computer Science and Information Security, Guilin University of Electronic Technology. Her research interests include graph data representation and formal methods.

TIANLONG GU received the M.Eng. degree from Xidian University, China, in 1987, and the Ph.D. degree from Zhejiang University, China, in 1996. From 1998 to 2002, he was a Research Fellow with the School of Electrical and Computer Engineering, Curtin University of Technology, Australia, and a Postdoctoral Fellow with the School of Engineering, Murdoch University, Australia. He is currently a Professor with the School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin, China. His research interests include formal methods, data and knowledge engineering, software engineering, and information security protocol.

RONGSHENG DONG received the B.S. degree from the China University of Geosciences, Wuhan, China, in 1989. He is currently a Professor with the School of Computer Science and Security, Guilin University of Electronic Technology. His current research interests include graph data, social computing, and protocol engineering. He is currently a member of the Advisory Commission on Teaching Computer Curriculum in Colleges and Universities under the Ministry of Education, and the Education Commission of the China Computer Federation. He has received research grants from NSFC and GXNSF.