Research Article

Optimal Representation of Large-Scale Graph Data Based on Grid Clustering and K^2 -Tree

Fengying Li , Enyi Yang , Anqiao Ma , and Rongsheng Dong

Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China

Correspondence should be addressed to Rongsheng Dong; ccrsdong@guet.edu.cn

Received 2 September 2019; Accepted 20 December 2019; Published 22 January 2020

Academic Editor: Xiangyu Meng

Copyright © 2020 Fengying Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The application of appropriate graph data compression technology to store and manipulate graph data with tens of thousands of nodes and edges is a prerequisite for analyzing large-scale graph data. The traditional K^2 -tree representation scheme mechanically partitions the adjacency matrix, which causes the dense interval to be split, resulting in additional storage overhead. As the size of the graph data increases, the query time of K^2 -tree continues to increase. In view of the above problems, we propose a compact representation scheme for graph data based on grid clustering and K^2 -tree. Firstly, we divide the adjacency matrix into several grids of the same size. Then, we continuously filter and merge these grids until grid density satisfies the given density threshold. Finally, for each large grid that meets the density, K^2 -tree compact representation is performed. On this basis, we further give the relevant node neighbor query algorithm. The experimental results show that compared with the current best K^2 -BDC algorithm, our scheme can achieve better time/space tradeoff.

1. Introduction

As a basic structure representing the relationship between data, graphs are widely used in web network analysis [1], biometric analysis [2], social group analysis [3], and other fields. With the continuous generation and accumulation of graph data, the traditional graph data representation method can no longer support the storage and operation of tens of thousands of nodes and edges [4]. According to Global Web statistics [5], the number of Facebook users exceeded 2.5 billion in 2019, and the average number of friends per person exceeds 300. If the adjacency list is used for storage, close to 900 TB is needed. According to CNNIC statistics [6], the number of Chinese web pages reached 2816 billion in 2019 and the number of hyperlinks is estimated to exceed 10^{16} . If adjacency table is used for storage, 10^6 TB space is required. To support fast querying, the entire adjacency table is usually loaded into memory. However, in actual situations, this strategy requires an excessive amount of storage space. Furthermore, with the rapid growth of users, storage problems will only become more and more severe. In recent years, many scholars have designed numerous data

structures for the compressed storage of graphs and have proposed algorithms to extend operations on these graphs.

There are several existing methods that are noteworthy for their good performance. Adler and Mitzenmacher [7] proposed a web graph compression scheme by finding nodes with similar sets of neighbors. Randall et al. [8] first proposed using a dictionary ordering of web page URLs to compress web graphs. Their method exploits the fact that many web pages on a common host have many similar neighbors. Boldi and Vigna [9] continued exploiting properties of web graphs in lexicographical ordering to propose gap coding and differential compression. In 2009, Chierichetti et al. modified Boldi and Vigna's compression method to compress social networks. Their approach used the principle of locality and similarity of web pages and the existence of a large number of interactive edges in social networks and involved backlink compression [10]. In 2010, Maserrat and Jian [11] proposed a compression method for social networks that can query neighbors in sublinear time. They achieve this by using an Euler data structure and multiple position linearization. Considering the similarity of neighbor nodes in the web page graph, LZ78 [12] and Repair

[13] achieve compression by replacing frequent pairs of characters in the adjacency list. Exploiting the sparsity and clustering of web pages, Brisaboa et al. proposed K^2 -tree [14], which uses a bit string to store information about the adjacency matrix of the original web graph. Since most of the elements in the adjacency matrix are zero, this method effectively saves storage space. Although K^2 -tree can achieve satisfactory time-space tradeoff, many isomorphic subtrees remain. To address this problem, Gu et al. applied an MDD (multivalued decision diagram) [15] to K^2 -tree representation, and K^2 -MDD [16] compression scheme was proposed. This method can compress web graph efficiently and compactly, but the query time is relative long. Delta- K^2 -tree [17] is an improved K^2 -tree algorithm that overcomes the shortcomings of the original K^2 -tree representation. Claude et al. proposed K^2 -partitioning [18] by using the unique rules of the domain in the web graph. Exploiting the distribution law of nodes in the web graph, Chang et al. [19] proposed an improved K^2 -tree algorithm K^2 -BDC, which can effectively represent the web graph and achieve the best time/space tradeoff. The algorithm is based on dividing the adjacency matrix into different squares along the main diagonal. Each square contains graph data of edges satisfying a certain density threshold. Each square is represented by K^2 -tree representation. However, it still has room for improvement in the following areas: (1) the technique considers that the adjacency matrix is divided along the main diagonal, the dense region away from the main diagonal cannot be well captured, and the dense structure may be destroyed. (2) K^2 -BDC uses the DAC coding technique [20] to further compress T vector and L vector, which may increase node neighbor query time. (3) K^2 -BDC cannot easily compress other types of the graph data. The method depends on the structure of graph data, and real clustering is not realized.

The quadtree [21] can effectively represent the graph data, and the construction thought is similar to the K^2 -tree representation by recursively dividing adjacency matrix. However, the division rule of the quadtree depends heavily on the distribution of the submatrix in the adjacency matrix. In the real network graph, the number of submatrices whose distribution of submatrices of the corresponding adjacency matrix satisfying the division rule is relatively small. In the case of dealing with large-scale graph data, this approach undoubtedly increases the required storage space overhead.

In this paper, the authors continue efforts to exploit the distribution characteristics of the adjacency matrix of the web graph and further optimize the K^2 -BDC and K^2 -tree. We find that if the dense structure in the adjacency matrix can be accurately obtained, on the one hand, the results can not only avoid the problem of the dense region segmentation caused by the mechanically partitioned adjacency matrix in the K^2 -tree scheme but also avoid the dense region away from the main diagonal in the K^2 -BDC scheme which cannot achieve good clustering. On the other hand, the new scheme reduces the height of the tree in the query operation. The main contributions of this paper are as follows: (1) a new grid clustering algorithm is proposed that can fully exploit any dense areas in the web graph making up for the shortcomings of the original K^2 -BDC. (2) The node neighbor

query algorithm after the compression structure is given. The results of the experiment are compared to those of the existing methods, and our method is found to achieve superior time/space tradeoff.

2. Related Work

In this section, we mainly introduce the related concepts of graphs and the construction principles of K^2 -tree and we analyze the edge distribution of the large web to provide theoretical support for subsequent clustering and K^2 -tree representation.

2.1. Graph and Related Concepts. Consider a graph $G = (V, E)$, where V represents the set of nodes, E represents the set of edges, n ($n = |V|$) represents the number of nodes, and m ($m = |E|$) represents the number of edges. The adjacency matrix and the adjacency list are usually used as the storage structure of the graph. Figure 1(a) is an undirected graph topology, Figure 1(b) is the adjacency matrix corresponding to the graph, and Figure 1(c) is the adjacency list of the graph. The adjacency list allows one to easily and quickly obtain the neighbors of any node in the graph and add new nodes to it. However, the list is not suitable for detecting connectivity between nodes. With the adjacency matrix, one can quickly increase or delete the edges of a node and can quickly detect the connectivity between nodes, but the storage space of the adjacency matrix is only related to the number of graph vertices and wastes a certain amount of storage space when storing sparse graphs. The addition of a new node requires space reallocation. Table 1 shows the spatial complexity required for the adjacency matrix and the adjacency list for the directed and undirected graphs, respectively. As can be seen from Table 1, for the adjacency matrix and the adjacency list, when a network graph of millions of nodes and edges is stored, the problem of excessive storage space becomes increasingly severe.

2.2. K^2 -Tree. Brisaboa et al. proposed K^2 -tree [14] using the sparseness and clustering of the web graph and achieved a satisfactory time/space tradeoff. The structural idea of K^2 -tree is divided mainly into the following two steps:

- (i) For an $n \times n$ adjacency matrix, evaluate whether n is a power of k (k is usually equal to 2). If the condition is met, go to (ii) to divide. If n is not equal to the power of k , increase the row and column in the adjacency matrix such that $n = k^s$ (s is a positive integer), where the elements of the added row and column are padded with "0," and then go to (ii) for partitioning.
- (ii) According to the MXQuntree rule [22], the matrix is divided into k^2 submatrices of the same size. If at least one of the elements in the submatrix is "1," then mark the matrix as 1 otherwise mark 0, top to bottom, and arrange these values from left to right to serve as the four children of the root node. The first layer of the K^2 -tree node is constructed. Then, the matrix labeled 1 is recursively processed and their

FIGURE 1: Undirected graph and its corresponding adjacency matrix and adjacency list.

TABLE 1: Comparison of spatial complexity when using an adjacency matrix and adjacency table for a directed graph and undirected graph.

Graph type	Directed graph	Undirected graph
Adjacency Matrix	$O(n^2)$	$O(n(n-1)/2)$
Adjacency List	$O(n+m)$	$O(n+m)$

values are used as the second layer nodes of K^2 -tree, and then it is repeated until the partitioned matrix elements are all 0 or the matrix has been divided into an element in the original adjacency matrix. As shown in Figure 2, the adjacency matrix and K^2 -tree correspond to a web graph with 16 vertices.

After the adjacency matrix is compressed, the structure information in the web graph is represented by a T vector and an L vector. The T vector stores all 0 values and 1 values of the K^2 -Tree except the last layer node, and the L vector stores the 0 value and the 1 value of the last layer node in the K^2 -tree. On the T vector and the L vector, the author of K^2 -tree representation proposed a rank operation to indirectly obtain the direct neighbors and reverse neighbors of any node. However, due to the K^2 -tree mechanical division of the adjacency matrix, the original dense structure in the adjacency matrix is broken, and the storage cost is increased. As the number of graph nodes increases, the height of the K^2 -tree increases, which necessitates more query time.

2.3. The Structural Characteristics of the Web Graph. Web graphs are often used for modeling large networks, where each web page in the web is viewed as a node in the graph and the links between web pages are treated as one edge in the graph. The study by Broder et al., in 2008, showed that most of the web graph feature functions are subject to the power law distribution [23] and that the corresponding adjacency matrix has certain sparsity and clustering [24]. To reflect this law more intuitively, we visualize the data sets CNR-2000 and EU-2005, as shown in Figures 3 and 4, respectively. The x -axis and the y -axis are the node numbers in the adjacency matrix, and we map for each edge in the adjacency matrix to a point on the two-dimensional space. We can conclude that, from a partial perspective, the distribution of edges is relatively concentrated. Overall, the entire two-dimensional plane is relatively sparse. Based on the above analysis, if we can quickly capture the dense areas in the web graph and then use K^2 -tree representation for compact representation of each dense area, we can save storage space and reduce the query time.

3. Large-Scale Web Graph Storage and Operation Scheme Based on Grid Clustering and K^2 -Tree

The representation scheme proposed for graph data based on grid clustering and K^2 -tree is mainly divided into the following three steps. First, introduce the grid clustering algorithm and how to find the dense regions in the web graph. Second, introduce a compression algorithm to compress dense areas. Finally, introduce how to query a node neighbor for a given node.

3.1. Grid Clustering

- (i) For a given graph $G = (V, E)$ with N nodes and M edges, we divide its corresponding adjacency matrix into length d (here, we take $d = 2$) to produce N^2/d^2 grid. For each edge in G , we map the edge to a different grid and count the density of each grid. Then, each grid is traversed in turn, and a grid with a grid density greater than the grid density threshold is included in the *List*.
- (ii) Traverse each grid in the *List* and calculate the Euclidean distance between the current grid and other grids according to a given distance threshold. If the distance is less than or equal to the distance threshold, combine these grids that meet the distance threshold and mark these grids and the current grids as accessed, counting the density of the merged large grid. If the density is greater than or equal to the density threshold, the large grid is included in the *cluster_list*. Repeat (ii) until all the grids have been accessed.
- (iii) Repeat (ii) according to the results of the partition in *cluster_list* until *cluster_list* is no longer changed. At this point, *cluster_list* records the location of each dense area and the clustering algorithm terminates. This pseudocode is shown in Algorithm 1.

3.2. Dense Area Compression. *cluster_list* records different dense regions in the adjacency matrix which can also be called a cluster after grid clustering. In addition, *boundary_list* has recorded the starting row, the ending row, the starting column, and the ending column of each dense area. For each dense area, compression is performed with K^2 -tree representation. This pseudocode is shown in Algorithm 2.

FIGURE 2: K^2 -tree ($k=2$) corresponding to the 16×16 adjacency matrix.

FIGURE 3: CNR-2000 visualization.

FIGURE 4: EU-2005 visualization.

3.3. Node Neighbor Query. For a given node, we first find its corresponding $cluster_list[i]$ through boundary list, and then iterate through the T and L vectors corresponding to $cluster_list[i]$ to find the neighbor of the node. This pseudocode is shown in Algorithm 3.

To describe the clustering and node neighbor query process more intuitively, we present a graph G_1 with 16 nodes and 17 edges. We use the adjacency matrix to describe the structure information of the graph, as shown in Figure 5.

In the example, the grid density threshold is set to 0.25, the distance threshold is set to 1, and the length of the grid is set to

2. The adjacency matrix is divided into 64 grids of the same size, and the position of each grid is recorded by subscripts (i, j) , where i represents the lateral offset of the grid and j represents the longitudinal offset of the grid. The filtered grids that meet the density threshold are included in the list according to the preset density threshold, $list = ((1, 2), (1, 3), (2, 2), (2, 6), (2, 7), (3, 7), (4, 4), (5, 4), (5, 5), (6, 2), (7, 4), (8, 6))$.

According to the preset distance threshold, traverse each grid in the $list$ in turn and calculate the distance between the current grid and other grids, merging and reorganizing the grids with distance less than or equal to 1 into the cluster. Then, $cluster[0] = ((1, 2), (1, 3), (2, 2))$, $cluster[1] = ((2, 6), (2, 7), (3, 7))$, $cluster[2] = ((4, 4), (5, 4), (5, 5))$, $cluster[3] = ((6, 2), (7, 4))$, and $cluster[4] = ((8, 6))$.

Each grid in the cluster is traversed in turn, and the distance between each $cluster[i]$ and $cluster[j]$ is calculated. If the distance is less than or equal to 1, the cluster is updated until the cluster is not changed. Then, $cluster[0] = (1, 2), (1, 3), (2, 2))$, $cluster[1] = ((2, 6), (2, 7), (3, 7))$, $cluster[2] = ((4, 4), (5, 4), (5, 5))$, $cluster[3] = ((6, 2), (7, 4))$, and $cluster[4] = ((8, 6))$. Using $boundary_list$ to record the start-line, end-line, start-column, and end-column of each $cluster[i]$, $boundary_list = ((1, 2, 2, 3), (2, 3, 6, 7), (4, 5, 4, 5), (6, 7, 2, 3), (8, 8, 6, 6))$. For each $cluster[i]$, a compact representation is made with K^2 -tree representation, as shown in Figures 6(a)–6(e).

For G_1 in Figure 5, the traditional K^2 -tree representation is used. As shown in Figure 7, the sum of the T and L vectors is 112 bits; however, our method requires only 64 bits. The storage space occupied by the boundary list can be neglected when processing the web graph of millions of nodes and edges. Our method not only saves 43% of the storage space but also reduces the height of the K^2 -tree, so the query time is also reduced. In summary, our approach can achieve relatively strong time and space tradeoff.

4. Experiment

To verify that our method can achieve better time/space tradeoff, we compared it with K^2 -tree, LZ78, Repair, and K^2 -BDC. These algorithms are used to compactly represent large-scale web graph and offer satisfactory time/space tradeoff, with K^2 -BDC achieving the best time/space balance. Our experimental environment is configured with an Intel(R) Core(T) i5-4590 CPU@3.30 GHz, 4 GB of running

Input: an adjacency matrix M , a *density threshold* that satisfies the minimum density, and a *distance threshold* that plots the maximum distance between the grids.

Output: a *boundary_list* contains the cluster boundary information and an adjacency matrix M_0 for which the cluster is removed. A *cluster_list* contains the position of the small grid in each cluster.

```

(1) Divide the adjacency matrix  $M$  into  $N^2/d^2$  grids, denoted as  $n$ ;
(2)  $n$ : = Number of grids to be filtered; List: empty queue, Boundary: empty queue;
(3) for ( $i = 1$  to  $n$ )
(4)   if ( $g_i \geq \text{density threshold}$ ) then
(5)     add  $g_i$  to the List;
(6)   end if
(7) end for
(8) Flag := 1;
(9) while (Flag == 1)
(10)   $m := \text{List.size}()$ ;
(11)  for ( $i = 1$  to  $m$ )
(12)    for ( $j = 2$  to  $m$ )
(13)      if ( $\text{Distance}(g_i, g_j) \leq \text{distance threshold} \&\& \text{Isaccessed}(g_j) == \text{false}$ ) then
(14)        Classify  $g_i$  and  $g_j$  into one class and mark  $g_j$  as accessed;
(15)      end if
(16)    end for
(17)    Mark  $g_i$  as accessed, count the density of all meshes belonging to the same class as  $g_i$ , and merge them into a large grid.
    The grid name is recorded as  $g_i^1$ ;
(18)    if ( $g_i^1 \geq \text{density threshold}$ ) then
(19)      add  $g_i^1$  to the cluster_list;
(20)    end if
(21)  end for
(22)  if (List != cluster_list) then // If the results of the two clusters are inconsistent, iterate again;
(23)    Flag := 1; // Iteration end flag;
(24)    List := cluster_list;
(25)    cluster_list := [];
(26)  else
(27)    Flag := 0; // Iteration end flag;
(28) end while
(29) Record the boundary value of each grid in the cluster_list, store this value in the boundary_list, and extract each cluster in the list
    from the original adjacency matrix, and the extracted adjacency matrix is termed  $M_0$ ;
(30) return  $M_0$ , boundary_list, cluster_list;

```

ALGORITHM 1: Grid clustering.

Input: *boundary_list* to record the boundary of the cluster boundary information and remove the adjacency matrix M_0 of the cluster.

Output: $T_1, T_2, T_3, \dots, T_n, T_{M0}, L_1, L_2, L_3, \dots, L_n, L_{M0}$;

```

(1)  $n := \text{boundary\_list.size}()/4$ ;
(2) for ( $i = 1$  to  $n$ )
(3)    $K^2$ -tree representation for each cluster  $i$ , stored with  $T$  and  $L$  vectors, denoted  $T_i, L_i$ ;
(4) end for
(5)  $K^2$ -tree representation of  $M_0$ , stored by  $T$  and  $L$  vectors, denoted as  $T_{M0}, L_{M0}$ ;
(6) return  $T_1, T_2, T_3, \dots, T_n, T_{M0}, L_1, L_2, L_3, \dots, L_n, L_{M0}$ ;

```

ALGORITHM 2: Cluster compression.

memory, operating system is Windows 8 (64 bits), and all experiments use only one core. The programming language is C++, and the compilation environment is gcc.

The data sets of our experiments are Enron, CNR-2000, and EU-2005. They can be obtained from the University of Milan's law web library [25]. The specific parameters of the data set can be obtained from Table 2, including the number of nodes, the number of edges, the average number of edges

of one node, the density of the graph, and the adjacency list are used to indicate the size required for the graph.

We use two indicators to evaluate the algorithm. The first is the bits needed for an edge average. The total storage space can be needed by calculating the compression divided by the number of edges of the data set. The second is the time required to query the neighbors of a node. For each node, we calculate the time to obtain all the neighbors of the node, and

Input: n is the number of the vertex of the graph, $boundary_list$ is the boundary value of the cluster, and $cluster_list$ is the actual position of each cluster in the adjacency matrix.

Output: direct neighbor set $List$ for node n ;

(1) $m := boundary_list.size()/2$;

(2) $List :=$ empty set;

(3) **for** ($i = 1$ to m)

(4) **if** ($boundary_list[2 * i] \leq m \&\& boundary_list[2 * i + 1] \geq m$) **then**

 Find the T vector and L vector corresponding to the cluster satisfying the boundary condition, and add the queried neighbors to the $List$;

(5) **end if**

(6) **end for**

(7) Find the T vector and the L vector of the M_0 . If there is a neighbor of the node, add the queried neighbor to the $List$.

(8) **return** $List$;

ALGORITHM 3: Node direct neighbor query.

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

FIGURE 5: G_1 adjacency matrix representation.

FIGURE 6: K^2 -tree representation of each cluster $[i]$.

FIGURE 7: The traditional K^2 -tree representation.

then divide these time sums by the total number of edges of the data set. The unit of time we use is μs .

As shown in Figure 8, in the data set Enron, compared with the result of LZ78 and Repair, our storage space is reduced by 57.1% and 32.6%. Compared with the results of

traditional K^2 -tree, our storage space is reduced by 30.1%, and the corresponding node neighbor query consumption is also reduced by 15.6%. Relative to the result of the current best algorithm K^2 -BDC, our storage space is also reduced by 3.2%, and the corresponding node neighbor query consumption is also reduced by 6%.

As shown in Figure 9, in the data set CNR-2000, compared with the result of LZ78 and Repair, our storage space is reduced by 78% and 61%, respectively. Our node neighbor query consumption is also reduced by 16.9% compared to the Repair. Compared with the results of traditional K^2 -tree, our storage space is reduced by 44.6%, and the corresponding node neighbor query consumption is also reduced by 37.1%. Relative to the result of the current best algorithm K^2 -BDC, our storage space is also reduced by

TABLE 2: Specific parameters of the web graph.

Name	Nodes	Edges	Edges/nodes	Graph density	Size (MB)
Enron	69244	276143	3.98	5.759×10^{-5}	2.4
CNR-2000	325557	3216152	9.88	3.034×10^{-5}	14
EU-2005	862664	19235140	22.30	3.585×10^{-5}	77

FIGURE 8: Space/time tradeoff over Enron.

FIGURE 10: Space/time tradeoff over EU-2005.

FIGURE 9: Space/time tradeoff over CNR-2000.

5% and the corresponding node neighbor query consumption is also reduced by 23%.

As shown in Figure 10, in the data set EU-2005, compared with the LZ78 and Repair, our storage space was reduced by 71% and 57.3%, respectively. Compared with the result of traditional K_2 -tree, our storage space is reduced by 46.4%, and the corresponding node neighbor query consumption is reduced by 49.3%. Compared with results of the current best algorithm K_2 -BDC, our storage space is reduced by 15.8% and the corresponding node neighbor query consumption is reduced by 10.8%. The experimental results show that our method can achieve better time and space tradeoff.

5. Conclusion

This paper proposes a large-scale graph data representation method based on clustering and K_2 -tree, which adopts a grid clustering algorithm to fully exploit the dense regions in the adjacency matrix so that a large number of “1” values are included in the dense region. Compared with the original adjacency matrix, the edge length of each dense region is

greatly reduced, which reduces the number of recursions required from the top layer to the leaf node in the K_2 -tree query operation and increases the storage space utilization. This method can efficiently and compactly represent the graph data of millions of nodes and edges and can also support node neighbor query operations.

Compared with the current best K_2 -BDC, our method can achieve better time and space tradeoff. In future research, we plan to the multivalued decision diagram to further improve the isomorphic subtree problem caused by the K_2 -tree representation and support more graph data operations on the compressed structure. Another component of planned future is to use this algorithm to compactly represent additional large-scale graph data with various distribution characteristics.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Natural Science Foundation of China (No. 61762024) and Natural Science Foundation of Guangxi Province (Nos. 2016GXNSFAA380054 and 2017GXNSFDA198050).

References

- [1] Z. Zhi, Q. Peng, Z. Li, X. Guan, and O. Muhammad, “Fast pagerank computation based on network decomposition and DAG structure,” *IEEE Access*, vol. 6, no. 1, pp. 41760–41770, 2018.

- [2] Y. Zhang, H. Lin, Z. Yang, J. Wang, and Y. Liu, "An uncertain model-based approach for identifying dynamic protein complexes in uncertain protein-protein interaction networks," *BMC Genomics*, vol. 18, no. 7, pp. 743–754, 2017.
- [3] M. Mohammadreza, S. M. Ebrahim, and R. A. Masoud, "Identifying fake accounts on social networks based on graph analysis and classification algorithms," *Security and Communication Networks*, vol. 13, no. 2, pp. 15–24, 2018.
- [4] Y. Zhang, Y. Liu, and G. Xiong, "Survey on succinct representation of graph data," *Software Journal*, vol. 25, no. 9, pp. 1937–1952, 2014, in Chinese.
- [5] "Facebook user statistics, Global web index," London, England, UK, 2019, <http://www.globalwebindex.com/reports/page/8>.
- [6] China Internet Development Report, China Internet Network Information Center, Beijing, China, 2019, <http://www.cnnic.cn/hlwfzyj/hlwxbzg/hlwztjbg/201902/P020190318523029756345.pdf>.
- [7] M. Adler and M. Mitzenmacher, "Towards compressing web graphs," in *Proceedings of the DCC*, pp. 203–212, Snowbird, Utah, USA, March 2001.
- [8] K. H. Randall, R. Stata, R. G. Wickremesighe, and J. L. Winener, "The link database: fast access to graphs of the Web," in *Proceedings of the DCC*, pp. 122–131, Snowbird, Utah, USA, 2002.
- [9] P. Boldi and S. Vigna, "The webgraph framework I: compression techniques," in *Proceedings of the 13th International Conference*, pp. 595–602, World Wide Web, New York, NY, USA, May 2004.
- [10] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *Proceedings of the ACM-SIGKDD*, pp. 219–227, Paris, France, June 2009.
- [11] H. Maserrat and P. Jian, "Neighbor query friendly compression of social networks," in *Proceedings of the ACM-SIGKDD*, pp. 303–325, Paris, France, July 2010.
- [12] N. J. Larsson and A. Moffat, "Off-Line dictionary-based compression," in *Proceedings of the DCC*, pp. 1722–1732, Snowbird, Utah, USA, March 2000.
- [13] P. Bille, I. L. Gørtz, and A. Prezza, "Space-efficient re-pair compression," in *Proceedings of the DCC*, pp. 171–180, Snowbird, Utah, USA, April 2017.
- [14] N. R. Brisaboa, S. Ladra, and G. Navarro, " K^2 -trees for compact web graph representation," in *String Processing and Information Retrieval*, Springer, Berlin, Germany, 2009.
- [15] A. Srinivasan, T. Ham, and S. Malik, "Algorithms for discrete function manipulation," in *Proceedings of the ICCAD*, pp. 92–95, New York, NY, USA, November 1990.
- [16] R. Dong, X. Zhang, and T. Gu, "Research on K^2 -MDD representation method and operation of large-scale graph data," *Journal of Computer Research and Development*, vol. 52, no. 12, pp. 2783–2792, 2016.
- [17] Y. Zhang, G. Xiong, Y. Liu, M. Liu, P. Liu, and L. Guo, "Delta- K^2 -tree for compact representation of web graphs," in *Proceedings of the Asia-Pacific Web Conference*, pp. 270–281, Beijing, China, September 2014.
- [18] F. Claude and S. Ladra, "Practical representations for web and socialgraphs," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 1185–1190, New York, NY, USA, October 2011.
- [19] 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 Personal Communications*, vol. 95, no. 3, pp. 2271–2284, 2017.
- [20] N. R. Brisaboa, S. Ladra, and G. Navarro, "DACs: bringing direct access to variable-length codes," *Information Processing & Management*, vol. 49, no. 1, pp. 392–404, 2013.
- [21] M. Nelson, S. Radhakrishnan, A. Chatterjee, and C. N. Sekharan, "On compressing massive streaming graphs with Quadrees," in *Proceedings of the IEEE International Conference on Big Data*, pp. 2409–2417, Santa Clara, CA, USA, November 2015.
- [22] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, San Francisco, CA, USA, 2006.
- [23] A. Broder and R. Kumar, "Graph structure in the web," *Computer Networks*, vol. 33, no. 1–6, pp. 309–320, 2000.
- [24] S. Raghavan and H. G. Molina, "Representing web graphs," in *Proceedings of the ICDE*, pp. 405–416, Bangalore, India, March 2003.
- [25] "The LAW laboratory from the University of Milan," 2018, <http://law.di.Unimi.it/datasets.php/s>.

Submit your manuscripts at
www.hindawi.com