Wireless Pers Commun (2017) 95:2271–2284
DOI 10.1007/s11277-017-4087-5

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

Liang Chang[1] · Xiangxuan Zeng[1] · Zhoubo Xu[1] · Junyan Qian[1] · Tianlong Gu[1] · Houbing Song[2]

Published online: 15 March 2017

Abstract Graph is widely used to model data in various applications. With the rapid growth of many emerging applications such as Internet of Things, it is urgent to require the processing capability on large scale graphs with billions of vertices. Web graph is a typical case of graph data that is widely used for analyzing the structure, behavior and evolution of the World Wide Web. In this paper, we focus on optimal representation of large-scale Web graphs. Our work is motivated by the need of fit large-scale graphs into the main memory and carry out analyze on them. By analyzing the adjacency matrix of Web graphs, we find two characteristics on the distribution of 1s in the matrix. Firstly, only a very small proportion of elements in the matrix are 1s. Secondly, majority of 1s gather around the principal diagonal and form a few number of clusters in the matrix. Based on these characteristics, we first develop a clustering mechanism to locate the clusters of 1s in the adjacency matrix. Then, we combine this clustering mechanism with a structure named K^2-tree and propose an approach for representing large-scale Web graphs compactly. Basic

✉ Tianlong Gu
cctlgu@guet.edu.cn

✉ Houbing Song
h.song@ieee.org

Liang Chang
changl@guet.edu.cn

Xiangxuan Zeng
zxx18229844034@163.com

Zhoubo Xu
xzbli_11@guet.edu.cn

Junyan Qian
qjy@guet.edu.cn

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

[2] Department of Electrical and Computer Engineering, West Virginia University, Montgomery, WV 25136, USA

idea of the approach is trying to compress a large number of zeros as a single zero. Experimental results show that, our approach not only reduces the space for representing a Web graph, but also reduces the time consumption for operations such as retrieving neighbors of any nodes on the graph; compared with existing approaches, our approach achieves the best space/time tradeoff.

Keywords Graph data · Web graph · Optimal representation · Clustering mechanism · K^2-tree

1 Introduction

Graph is widely used to model data in various applications [1–7]. With the rapid growth of emerging applications like social network analysis, semantic Web analysis, bioinformatics network analysis, and Internet of Things, it is urgent to require the processing capability on large scale graphs with billions of vertices [8–14]. In order to analyze the structure, behavior and evolution of the World Wide Web, researchers often model the Web as a directed graph by treating Web pages as nodes and treating the links between pages as directed edges. Such a directed graph is also called Web graph [14]. Web graph is a typical case of graph data.

With the rapid increase in the size of the World Wide Web, it becomes a great challenge to represent a large-scale Web graph on computers. According to the report of China Internet Network Information Center, by the end of 2015, the total numbers of Web pages and hyperlinks in China were more than 212 billion and 10^{18} respectively. If the corresponding Web graph is represented as adjacent matrix, then the space for storing it is more than 40 TB. Such a Web graph is too big to be loaded in the main memory of computer for analyzing. Obviously, the explosive growth of the scale of Web graphs has made it difficult for using traditional storage structure to represent and manipulate the graphs.

At present, there are three approaches to deal with large-scale Web graphs. (1) Make use of external memory to store Web graph [15]. Shortcoming of this approach is a large amount of time consumption for operations on graphs; since the access speed of external memory is 4–6 orders of magnitude slower than that of main memory, and the graph has to be loaded from external memory to main memory frequently. Therefore, when taking this approach, the main concern of researchers is how to reduce the number of disk I/O operations by optimizing the spatial locality of data access. (2) Divide a big Web graph into many parts and store them in main memories of different computers in a distributed system [16]. Shortcoming of this approach is a big communication cost between computers caused by the coupling among all the parts of the Web graph. Therefore, when taking this approach, the main concern of researchers is how to reduce the communication cost and time consumption by technically designing a good segmentation algorithm. (3) Convert the Web graph into a compressed form so that it could be stored in the main memory of a computer and could be handled effectively [17]. In this paper, we follow the third approach.

By analyzing the adjacency matrix of a Web graph, we find two characteristics on the distribution of the values of ones in the matrix. First, only a very small proportion of elements in the matrix are ones. Secondly, majority of 1s gathers around the principal diagonal and form a few number of clusters in the matrix. Based on these characteristics,

we develop a clustering mechanism to locate the clusters in the matrix, and incorporate this mechanism into a structure named K^2-tree. As a result, we propose an algorithm for converting large-scale Web graphs into compressed forms, and we demonstrate that this algorithm achieves the best space/time tradeoff in all existing algorithms.

2 The K^2-Tree

K^2-tree is a structure proposed by Brisaboa et al. [18] for representing graphs in the case that the number of edges is far less than the square of the number of nodes. The basic idea is to transform the adjacency matrix of a graph into a complete K^2-fork tree, where K is a positive integer. In this K^2-complete tree, each node labeled with 1 just stands for an element of 1 in the matrix, whereas each node labeled with 0 will stands for an area which is full of 0s in the matrix.

More precisely, given an adjacency matrix, the K^2-tree approach will first generate a root node and divides the matrix into K^2 submatrixes of the same size. Be corresponding to each submatrix, the approach will generate a child for the root node, and labels this child node with 0 if all elements in the submatrix is 0; otherwise the child node will be labeled with 1. During this process, all the K^2 submatrixes are sorted by the order that from top to bottom and then left to right. Such a sequence is reflected in the ordering of the corresponding children nodes that from left to right. Then, for each submatrix which contains 1, the approach will divide it again into K^2 submatrixes; and correspondingly K^2 children nodes will be generated and be labeled with 0 or 1 in the tree. Such a process is repeated, until the submatrix containing 1 is a 1×1 matrix.

In the case that the order of an adjacency matrix is not an integral power of the integer K, the K^2-tree approach need to do some preprocessing. It will introduce a number of rows and columns of 0s into the bottom and the right-hand side of the adjacency matrix so that the order of the matrix is an integral power of K. Because a square matrix of 0s will be compressed as a single node labeled with 0 in the K^2-tree, 0s introduced in the preprocessing stage don't bring many effects on the performance of the K^2-tree approach.

As an example, the white part of the matrix in the left part of Fig. 1 is an adjacency matrix for a graph with 11 nodes and 12 edges. We will represent it as a K^2-tree with $K = 2$. First, 0s in the grey part of the matrix are introduced to transform the adjacency matrix into a matrix with an order 16. Then, the complete 2^2-fork tree shown in the right

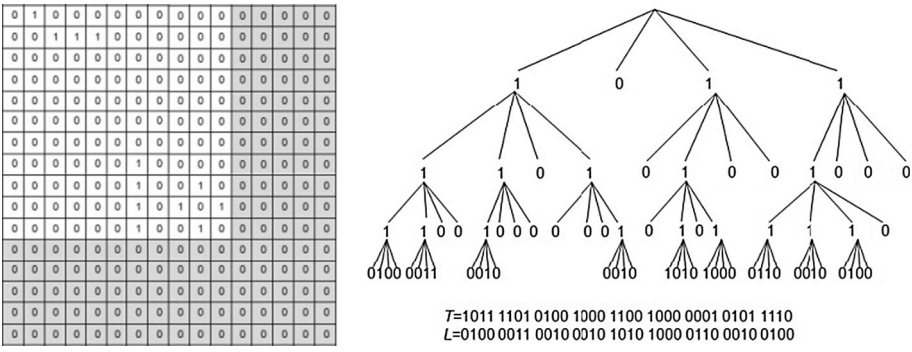

Fig. 1 Adjacency matrix and the corresponding K^2-tree with $K = 2$

part of this figure will be constructed by the K^2-tree approach. Now, let L be a bit-vector composed of the labels of the nodes in the lowest level of the tree, with the order that from left to right; and let T be a bit-vector composed of the labels of all the other nodes in the tree, with the order that from top to bottom and then left to right. Then, we just need to store these two bit-vectors in computer since they contain all the information of original graph. In this example, in the case that the graph is represented as adjacency matrix, the space consumption is 121 bits. By making use of the K^2-tree approach, we only need 72 bits to store the vectors T and L. Obviously, the K^2-tree approach can reduce storage space greatly.

The K^2-tree approach also supports efficient operations on graphs. Based on a top-down traverse of the tree, we can realize forward and backward navigation on graphs. Furthermore, the operations of checking individual links and retrieving links between given ranges of nodes can also be realized [19].

3 Limitations of K^2-Tree in Representing Web Graphs

K^2-tree provides a good approach for representing graphs compactly. However, since it divides a matrix into K rows and K columns mechanically during the compression process, it has as least three shortcomes.

First, the close coupling between nodes might be destroyed and it will increase the time consumption for query operations on the Web graph. In a World Wide Web there are always some groups of Web pages which are related tightly. In an extreme case, there exist links between every pair of pages in a group. By the K^2-tree approach, nodes in such a group might be dispersed to different submatrixes and therefore make a close connection between submatrixes. As a result, for operations such as forward or backward navigation on the graph, the visit of nodes in a K^2-tree has to be switched frequently between brother nodes. The switch of visit between brother nodes is time consuming, since it has to go through the father node.

Second, a large area of matrix full of 0s might be divided into many submatrixes and therefore can't be compressed completely. The spirit behind the K^2-tree approach is trying to compress a large number of zeros as a single zero. If there is a large area of matrix full of 0s, the ideal result is to represent this area by a single node labeled with 0. However, once it is divided into many submatrixes, more 0s and consequently more spaces will be used for the representation.

Thirdly, the height of the K^2-tree has a major impact on the time consumption of operations on the graph. Suppose the number of nodes in a Web graph is n, then the height of the corresponding K^2-tree will be $h = \lceil \log_k n \rceil$. For operations such as forward or backward navigation on the Web graph, there will be lots of top-down traversals accompanied with backtracking processes. For example, on the data set CNR-2000 [14] of Web graph which has 325,557 nodes and 3,216,152 edges, we made an experiment to visit all the neighbors of a given node in it. We found that the K^2-tree was recursively visited 450 times in average, before all the neighbors of a given node were found. Therefore, in order to reduce the time consumption of query operations, we need to reduce the height of the K^2-tree.

Now let us have a closer look of Web graph and analyze the adjacency matrix of it. CNR-2000 is a public data set of Web graph that often used as benchmark in literatures. As we mentioned before, the graph has 325,557 nodes and 3,216,152 edges. In order to

investigate the distribution of 1s in the adjacency matrix, we map the line number and column number of nodes into the X-coordinate and Y-coordinate respectively, and map each element in the matrix with value 1 into a corresponding point in the coordinate system. As a result, we get Fig. 2. It is obvious that majority of 1s are gather around the principal diagonal and form a few number of clusters.

Next, we make a quantitative analysis of this phenomenon. Let us horizontally divide the adjacency matrix of CNR-2000 into many parts. Without loss of generality, we let the height of each part be 65,536, and in this case, the matrix is in fact divided into 5 parts. We count the number of 1s contained in each part and denote it as *total_num*, as shown by the second column in Table 1. Furthermore, for each part, we pick out the square matrix whose principal diagonal is a part of the principal diagonal of the adjacency matrix; we call this square matrix as *main submatrix*. Now, for each part, we count the number of 1s in the main submatrix and denote it as *in_num*, as shown by the third column in Table 1; we also count the number of 1s not in the main submatrix and denote it as *out_num*, shown by the fourth column in Table 1. Finally, we calculate the ratio of *out_num* to *total_num* and call the result as *outlier rate*. With the index of *outlier rate*, it is shown again that majority of ones in the adjacency matrix gather around the principal diagonal and form a few number of clusters.

Based on the above observation, we get the following idea to improve the K^2-tree approach. Given the adjacency matrix of a Web graph, instead of dividing it mechanically, we will carry out the following operations. (1) Find out such submatrixes of the adjacency matrix that majority elements in the submatrix are 1s; compress each submatrix by the K^2-tree approach. (2) Remove all of these submatrixes from the adjacency matrix and fill their places in the adjacency matrix with 0s; compress the resulted adjacency matrix by the K^2-tree approach. (3) Record all of those K^2-trees generated in the previous steps by some mechanism, and get a compressed representation of the graph.

Figure 3 illustrates the above idea. In the rest of the paper, we will call the submatrixes found in the first step as *clusters* of 1s.

Intuitively, our method can overcome or alleviate the defects of the K^2-tree approach discussed in the beginning of this section. First, the close coupling between nodes is protected by the first step. Second, areas of matrix full of 0s in the adjacency matrix are expanded in the second step, and therefore each single node labeled with 0 in the K^2-tree can represent more elements of the adjacency matrix. Thirdly, the height of the resulted

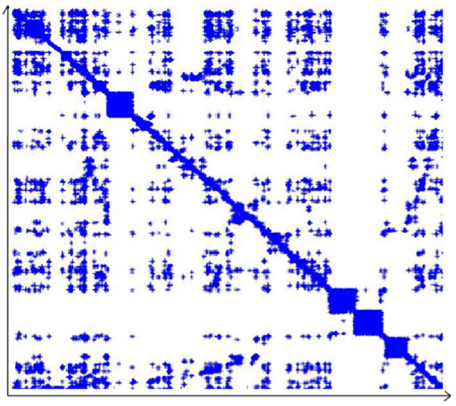

Fig. 2 Distribution of 1s in the adjacency matrix of CNR-2000

Table 1 Node distribution of the data set CNR-2000

Line numbers	Number of 1s in this part (*total_num*)	Number of 1s in the main submatrix (*in_num*)	Number of 1s not in the main submatrix (*out_num*)	Outlier rate (%)
0–65,536	776,538	768,697	7885	1
65,537–131,072	457,733	424,237	33,496	7.3
131,073–196,608	349,665	343,191	6474	1.8
196,609–262,144	960,510	954,170	6340	0.66
262,145–325,557	671,662	662,415	9247	1.38

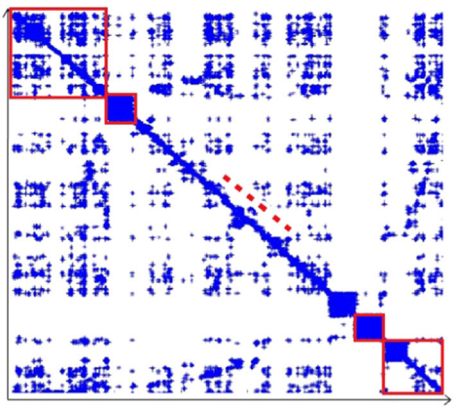

Fig. 3 Partition on the adjacency matrix according to the clustering results

K^2-tree is decreased; since we generate many K^2-trees where each K^2-tree only represents a submatrix or represents a very sparse matrix that has been backfilled by 0s.

4 Incorporating Clustering Mechanism into K^2-Tree

In this section, we give a detailed description to our method. Firstly we propose an algorithm *Clustering*(*M*, *MinSize*, ϕ) to find out the clusters of 1s in the adjacency matrix *M* along the principal diagonal. In this algorithm, the parameter *M* is an adjacency matrix of the Web graph that we will compress; parameters *MinSize* and ϕ respectively sets the minimum size and the minimum outlier rate for the clusters that we are looking for. After the execution of the algorithm, we will get a queue of matrixes that will be compressed into K^2-trees one by one in Algorithm 2.

Given a Web graph, it is possible that some nodes that have the same neighbors are not encoded in adjacent positions. Such an order of nodes will damage the locality characteristics of the adjacency matrix. Therefore, in the beginning of Algorithm 1, we firstly make use of the BFS technique [21] to change the order of nodes in the adjacency matrix so as to get a better locality characteristic.

In Algorithm 1, we look up clusters of 1s along the principal diagonal of the matrix *M*. Starting from the first element in *M*, the algorithm will selects a submatrix with the size of *width* × *width*, and then calculate the outlier rate of it. If this submatrix satisfies the two

conditions on clusters, then it will be recorded in the queue *list*, and its area in the matrix *M* will be filled with 0s. The purpose of setting the parameter *MinSize* is to avoid the cases that too many clusters will increase the time consumption of operations on the compressed representation of graphs.

Algorithm.1 *Clustering*(*M*, *MinSize* ,φ)
Input: an adjacency matrix *M*, an integer *MinSize* that specifies the minimum size of cluster, and a real number φ that specifies the minimum outlier rate of cluster.
Output: a queue *list* of matrixes
(1) use the BFS [15] technique to reorder nodes in *M*;
(2) *list* := empty queue;
(3) *n* := number of rows in *M*;
(4) *p_starting* := 0;
(5) **WHILE** (*p_starting* < *n*) **DO**
(6) *width* := 1;
(7) *flag* :=0;
(8) **WHILE** (*width* < *n* - *p_starting*) AND (*flag* = 0) **DO**
(9) *M_cluster* := the submatrix of *M* that starting from the line *p_starting* to the line (*p_starting* +*width*) and from the column *p_starting* to the column (*p_starting* +*width*);
(10) *in_num* := the number of 1s contained in *M_cluster*;
(11) *total_num* := the number of 1s contained in the area that from the line *p_starting* to the line (*p_starting* +*width*) in the matrix M;
(12) *out_rate* := (*total_num* - *in_num*) / *total_num*;
(13) **IF** (*out_rate* ≤ φ) AND (*width* ≥ *MinSize*) **THEN**
(14) put the matrix *M_cluster* into the queue *list*;
(15) replace all the 1s in the area *M_cluster* of *M* with 0s;
(16) *flag* :=1;
(17) **ENDIF**
(18) *width* := *width* +1;
(19) **ENDWHILE**
(20) *start_point* := *start_point* + *width* + 1;
(21)**ENDWHILE**
(22) put *M* into the queue *list*;
(23) **RETURN** *list*

Now we can present our compression algorithm *Cluster-K²-tree*(*M, K*). The first four steps of Algorithm 2 are designed to calculate the parameters *MinSize* and ϕ. Then the algorithm *Clustering*(*M*, *MinSize*, ϕ) is called and a queue *list* of matrixes is generated. For each matrix contained in *list*, we make a compression to it by the K^2-tree approach, and get the corresponding *T* vectors and *L* vectors. The storage mode used here is a little different from the storage mode of the K^2-tree approach. In Algorithm 2, the *T* vectors of compression results by the K^2-tree approach will be stored independent; but all the *L* vectors, except the *L* vector corresponding to the whole adjacency matrix, will be composed to form the vector *global_L*, which will then be transformed into an array and a bitmap by the DACs [22] encoding approach. The DACs coding technique can compress the *L* vectors with a high efficiency and support fast random access [13]. Due to the space limitation, here we omit the detailed process of DACs encoding.

Algorithm.2 *Cluster-K^2-tree(M,K)*
Input: an adjacency matrix M and an integer K.
Output: a sequence of bit vectors.
(1) n := number of rows in M;
(2) m := number of the element 1 contained in M;
(3) $\varphi := m / n^2$;
(4) $MinSize := n \times K^2 \times \varphi$;
(5) $list := Clustering(M, MinSize, \varphi)$;
(6) for each i ($1 \le i \le list.length$), make a compression to the matrix $list[i]$ by the K^2-tree approach, denote the resulted T bit-vector by T_i and denote the resulted L bit-vector by L_i;
(7) let L_{global} be a bit-vector generated by combining the bit-vectors L_1, …, $L_{list.length-1}$ sequentially;
(8) make a DACs[16] encoding to L_{global} and let $L_{clusters}$ be the result;
(9) **RETURN** $T_1, T_2, \ldots, T_{list.length}, L_{clusters}, L_{list.length}$.

As an example, let us consider again the adjacency matrix in the left part of Fig. 1. In the case that $MinSize = 4$ and $\phi = 0.1$, we will get two clusters shown in Fig. 4. For each of these clusters, we make a compression to it by the K2-tree approach with K = 2, and get the results shown in Fig. 5. Now we only need to record the bit-vectors $T1$, $T2$, $L1$ and $L2$, and the corresponding space consumption is only 64bit. Recall that, in Section II, it needs 121 bits to store the adjacency matrix of this graph, and needs 72 bits by the original K2-tree approach.

The advantage of our method not only reflects in the ratio of compression, but also reflects in the time consumption of query operations on Web graphs. The reason is that our method reduces the height of the generated K^2-trees and, at the same time, preserves the close coupling between nodes. For example, the height of the K^2-tree in the left part of Fig. 1 is 5, and it is now reduced to 4 in Fig. 5. Furthermore, if we need to visit all the neighbors of the tenth node in the graph, then the K^2-tree in Fig. 1 will be visited recursively 31 times; however, the K^2-trees in Fig. 5 will be visited only 15 times.

The above advantage is more obvious in large-scale Web graphs. For example, let us consider again the data set CNR-2000 that we discussed in Section III. In the case that $MinSize = 40$ and $\phi = 0.00003$, we will get 221 clusters. For the query operation that visiting all the neighbors of a given node, by our method the K^2-trees will be visited recursively 361 times in average. However, recall that, in the discussion of Section III, the

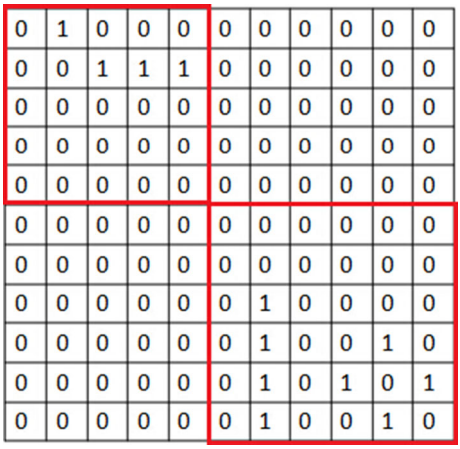

Fig. 4 Partition of an adjacency matrix

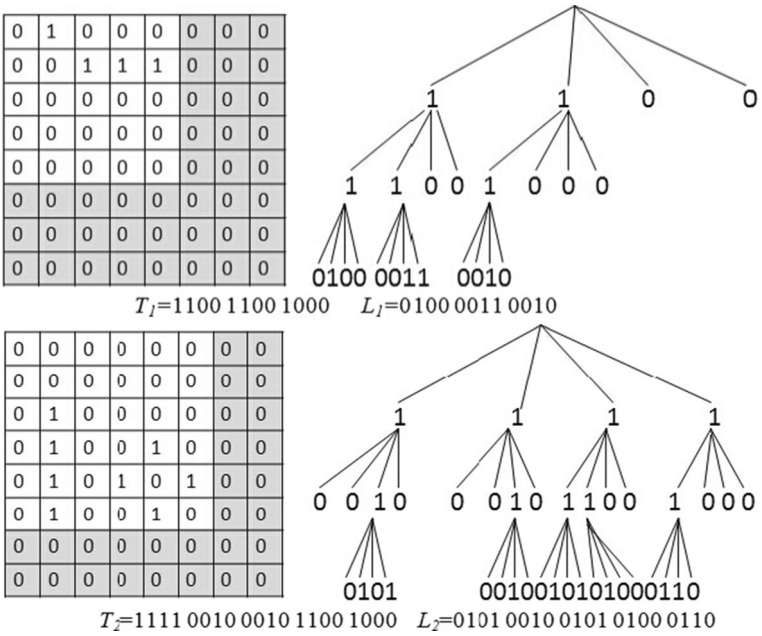

Fig. 5 Submatrixes and corresponding K^2-trees according to the result of clustering

K^2-tree generated by the original K^2-tree approach was visited recursively 450 times in average. Obviously, we get a reduction of almost 20%.

5 Experiments

In this section, we compare our method with existing approaches for representing large-scale Web graphs.

At present three are three typical algorithms in the literatures, named K^2-tree [18], Re-Pair [23] and LZ78 [23], that have excellent space/time tradeoff. Source codes of these algorithms are available on the website of the FCWGR (Fast Compact Web Graph Representations) project [24] from the University of Chile. We downloaded the source codes from the Website and ran them in our testing environment. Our testing environment is based on a computer with Intel(R) Core(T) i5-4590 CPU@3.30 GHz and 4 GB RAM. Operation system is Ubuntu GNU/Linux in version 12.04 (64 bits). All tests only use one CPU core. Programming language is the C language and the compiler is gcc in version 4.4.7.

We use the data sets CNR-2000 and EU-2005 as benchmarks. CNR-2000 and EU-2005 are data sets for real Web graphs and are available on the website of the LAW laboratory [20] from the University of Milan. Some information on these data sets are listed in Table 2, including the number of nodes, the number of edges, ratio of the number of edges to the number of nodes, and the size for a plain adjacency list representation of the graph (using 4-byte integers).

In the experiment, we take into account both space and time. Space consumption is measured as a ratio of bits per edge. More precisely, we calculate the total space for storing

Table 2 Description of the web graphs used in the experiment

	Nodes	Edges	Edges/nodes	Size (MB)
CNR-2000	325,557	3,216,152	9.88	14
EU-2005	862,664	19,235,140	22.30	77

the results of compression, and divide it by the number of edges in the Web graph. For the time consumption, we focus on the operation that finding all neighbors of a given node in the graph. For every node in the graph, we respectively calculate the time consumption of finding all the neighbors of it; then we add all of these time consumptions together and divide the sum by the number of edges in the Web graph. The unit of time in the experiment is microsecond.

Figure 6 shows the space and time consumption on the data set CNR-2000. During the execution of our algorithm, there are 221 clusters been generated. Figure 7 shows the results on the data set EU-2005, and on this data set there are 180 clusters been generated during the execution of our algorithm.

Compared with the classical K^2-tree approach, our method significantly reduces both the space consumption and the time consumption. More precisely, for the space consumption, our method reduces 42 and 38% respectively in the data sets CNR-2000 and EU-2005. For the time consumption, our method reduces 20 and 43% respectively.

Compared with the Re-Pair and LZ78 algorithms, our method significantly reduces the space consumption. More precisely, on the data set CNR-2000, our method reduces 59% to Re-Pair and 78% to LZ78; on the data set EU-2005, our method reduces 50% to Re-Pair and 65% to LZ78. The disadvantage of our method is that it needs more time on the operation of finding all neighbors of a given node. More precisely, on the data set CNR-2000, our method increases 6% to Re-Pair and 75% to LZ78; on the data set EU-2005, our method increases 45% to Re-Pair and 95% to LZ78.

Although our method needs more time compared with Re-Pair and LZ78, it achieves a better space/time tradeoff. Furthermore, for operations on Web graphs, both Re-Pair and LZ78 only support forward navigation. Since our method is an improvement of the K^2-tree approach, in theory it supports all the operations supported by K^2-tree approach, such as forward navigation, backward navigation, deciding the existence of a link, and retrieving the links between certain nodes [19].

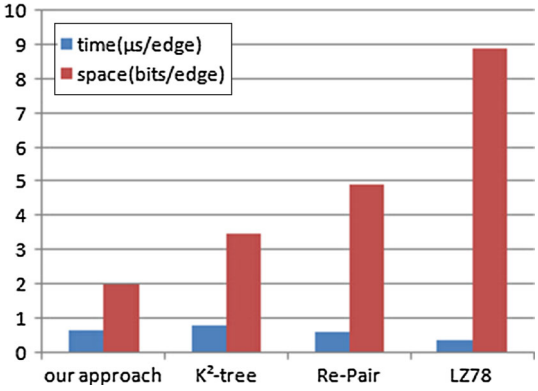

Fig. 6 Space/time tradeoff of querying for neighbors over CNR-2000

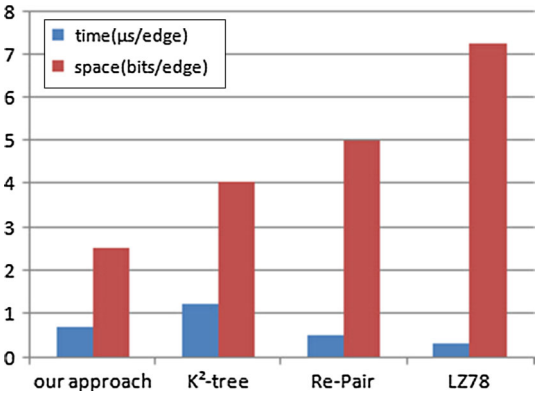

Fig. 7 Space/time tradeoff of querying for neighbors over EU-2005

6 Conclusions and Future Work

With the rapid development of World Wide Web and the rapid growth of many emerging applications on the Web, it is urgent to require the processing capability on large scale graphs with billions of vertices. How to represent such large scale graphs is a problem full of challenge.

In this paper, by combining a clustering mechanism with the K^2-tree structure, we propose a method for representing large-scale Web graphs compactly. Our method firstly finds out clusters of 1s in the adjacency matrix, and represents each cluster compactly by making use of the K^2-tree approach. Then, for the original adjacency matrix of the Web graph, our method replaces all the 1s occurring in the clusters by 0s, and gets a sparse adjacency matrix. This sparse adjacency matrix is represented compactly again by making use of the K^2-tree approach. Finally, the representations of clusters and sparse adjacency matrix are well-organized and we get a compressed representation of the whole Web graph. Compared with existing approaches in the literature, such as the classical K^2-tree approach, the Re-Pair approach and the LZ78 approach, our method not only reduces the space for representing Web graph but also reduces the time consumption for the query operation on the graph. Furthermore, our method has the best space/time tradeoff.

In the future work we will extend our method to support more operations on graphs such as the forward navigation, backward navigation, existence checking of certain links, and retrieving of links between nodes. Another work is to extend the method to deal with dynamic graph data.

Acknowledgements This work is supported by the Natural Science Foundation of China (Nos. 61572146, 61363030, 61572231, U1501252); the Natural Science Foundation of Guangxi Province (Nos. 2015GXNSFAA139285, 2016GXNSFDA380006); and the High Level of Innovation Team of Colleges and Universities in Guangxi and Outstanding Scholars Program.

References

1. Sun, Y. C., Yu, X. P., Bie, R. F., & Song, H. B. (2016). Discovering time-dependent shortest path on traffic graph for drivers towards green driving. *Journal of Network and Computer Applications, 83*(1), 204–212.

2. Lv, Z., Song, H., Basanta-Val, P., Steed, A., & Jo, M. (2017). Next-generation big data analytics: State of the art, challenges, and future research topics. *IEEE Transactions on Industrial Informatics*. doi:10.1109/TII.2017.2650204.
3. Song, H., Srinivasan, R., Sookoor, T., Jeschke, S., & Cities, S. (2017). *Foundations, principles and applications* (pp. 1–906). Hoboken: Wiley.
4. Tawalbeh, L. A., Bakheder W., & Song H. (2016) A mobile cloud computing model using the cloudlet scheme for big data applications. In *IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies* (*CHASE*) (pp. 73–77). Washington, DC.
5. Shojafar, M., Javanmardi, S., Abolfazli, S., & Cordeschi, N. (2015). FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. *Cluster Computing, 18*(2), 829–844.
6. Shojafar, M., Cordeschi, N., & Baccarelli, E. (2016). Energy-efficient adaptive resource management for real-time vehicular cloud services. *IEEE Transactions on Cloud Computing*. doi:10.1109/TCC.2016.2551747.
7. Yu, J. G., Deng, X., Yu, D. X., Wang, G. H., & Gu, X. (2013). CWSC: Connected k-coverage working sets construction algorithm in wireless sensor networks. *International Journal of Electronics and Communications, 67*(11), 937–946.
8. Sun, Y. C., Lu, C., Bie, R. F., & Zhang, J. S. (2016). Semantic relation computing theory and its application. *Journal of Network and Computer Applications, 59,* 219–229.
9. Lin, C., Song, Z., Song, H., Zhou, Y., Wang, Y., & Guowei, W. (2016). Differential privacy preserving in big data analytics for connected health. *Journal of Medical Systems, 40*(4), 1–9.
10. Tawalbeh, L. A., Mehmood, R., Benkhlifa, E., & Song, H. (2016) Mobile cloud computing model and big data analysis for healthcare applications. In *IEEE Access* (Vol. 4, pp. 6171–6180).
11. Lin, C., Wang, P., Song, H., Zhou, Y., Liu, Q., & Guowei, W. (2016). A differential privacy protection scheme for sensitive big data in body sensor networks. *Annals of Telecommunications, 71*(9–10), 465–475.
12. Cordeschi, N., Shojafar, M., & Baccarelli, E. (2013). Energy-saving self-configuring networked data centers. *Computer Networks, 57*(17), 3479–3491.
13. Yu, J. G., Chen, Y., Ma, L. R., Huang, B. G., & Cheng, X. Z. (2016). On connected target k-coverage in heterogeneous wireless sensor networks. *Sensors, 16*(1), 104.
14. Meusel, R. (2015). The graph structure in the web—analyzed on different aggregation levels. *The Journal of Web Science, 1*(1), 33–47.
15. Vitter, J. S. (2009). External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys, 33*(2), 209–271.
16. Khan, L. R., Thuraisingham, B., McGlothlin, J., Masud, M. M., & Husain, M. F. (2011). Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Transactions on Knowledge and Data Engineering, 23*(9), 1312–1327.
17. Boldi, P., & Vigna, S. (2004). The webgraph framework I: Compression techniques. In: *Proccedings of the 13th International Conference on World Wide Web* (pp. 595–602).
18. Brisaboa, N. R., Ladra, S., & Navarro, G. (2009). K^2-trees for compact web graph representation. In: *Proceedings of the 16th International Symposium on String Processing and Information Retrieval* (pp. 18–30).
19. Brisaboa, N. R., Ladra, S., & Navarro, G. (2014). Compact representation of Web graphs with extended functionality. *Information Systems, 39*(1), 152–174.
20. http://law.di.unimi.it/datasets.php.
21. Apostolico, A., & Drovandi, G. (2009). Graph compression by BFS. *Algorithms, 2*(3), 1031–1044.
22. Brisaboa, N. R., Ladra, S., & Navarro, G. (2013). DACs: Bringing direct access to variable-length codes. *Information Processing and Management, 49*(1), 392–404.
23. Claude, F., & Navarro, G. (2010). Fast and compact web graph representations. *ACM Transactions on the Web, 4*(4), 523–530.
24. http://webgraphs.recoded.cl/.

Liang Chang received his Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 2008. He is currently a professor in the School of Computer Science and Information Security, Guilin University of Electronic Technology, China. His research interests include data and knowledge engineering, formal methods, and intelligent planning.

Xiangxuan Zeng received his Bachelor's degree in computer science from the Institute of Changsha University, in 2014. He is currently a Graduate student in the School of Computer Science and Information Security, Guilin University of Electronic Technology, China. His research interests include graph data and formal methods.

Zhoubo Xu received her Ph.D. degree in computer science from Xi'dian University, China in 2011. She is currently an associate professor in the School of Computer Science and Information Security, Guilin University of Electronic Technology, China. Her research interests include data and knowledge engineering, formal methods, and intelligent algorithms.

Junyan Qian received his Ph.D. degree from the Southeast University of China in 2008. He is currently a professor in the School of Computer Science and Information Security, Guilin University of Electronic Technology, China. His research interests include formal verification and optimization algorithm.

Tianlong Gu received his Ph.D. degree from Zhejiang University, China in 1996. From 1998 to 2002, he was a Research Fellow at the School of Electrical and Computer Engineering, Curtin University of Technology, Australia and Postdoctoral Fellow at the School of Engineering, Murdoch University, Australia. He is currently a professor in the School of Computer Science and Information Security, Guilin University of Electronic Technology, China. His research interests include formal methods, data and knowledge engineering.

Houbing Song received the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, in August 2012. In August 2012, he joined the Department of Electrical and Computer Engineering, West Virginia University, Montgomery, WV, where he is currently an Assistant Professor and the Founding Director of both the Security and Optimization for Networked Globe Laboratory (SONG Lab, www.SONGLab.us), and West Virginia Center of Excellence for Cyber-Physical Systems sponsored by West Virginia Higher Education Policy Commission. In 2007 he was an Engineering Research Associate with the Texas A&M Transportation Institute. He is the editor of four books, including Smart Cities: Foundations, Principles and Applications, Hoboken, NJ: Wiley, 2017, Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications, Chichester, UK: Wiley, 2017, Cyber-Physical Systems: Foundations, Principles and Applications, Waltham, MA: Elsevier, 2016, and Industrial Internet of Things: Cybermanufacturing Systems, Cham, Switzerland: Springer, 2016. He is the author of more than 100 articles. His research interests include cyber-physical systems, internet of things, cloud computing, big data analytics, connected vehicle, wireless communications and networking, and optical communications and networking. Song is a senior member of IEEE and a member of ACM. Song was the very first recipient of the Golden Bear Scholar Award, the highest faculty research award at West Virginia University Institute of Technology (WVU Tech), in 2016.