Bulletin of Environment, Pharmacology and Life Sciences
Bull. Env.Pharmacol. Life Sci., Vol 4 [Spl issue 1] 2015: 261-268

Online ISSN 2277-1808
Journal's URL:http://www.bepls.com
CODEN: BEPLAD
Global Impact Factor 0.533
Universal Impact Factor 0.9804

FULL LENGTH ARTICLE **OPEN ACCESS**

Large Graphs Compression using a combined algorithm

Nafiseh Lavari Lashtaghani, Saman Dehghanian
International Center Asaloye Payam Noor Univ. of Iran
Nafisehlavari66@gmail.com
Islamic Azad Univ. of Dariun, Iran
saman_dehghanian@yahoo.com

ABSTRACT
Web Graph provides a framework for large-scale graph compression; it is the foundation of many real-world datasets. The increasing size of graphs presents a major obstacle in exploiting modern compression techniques. Recently some techniques have proved sufficiently efficient in allowing greater storage of Web graphs in a limited memory. These techniques mainly use repetitions that exist in the graphs for compressing purposes. In this paper I use similarity property technique based on the pages that are proximal in the lexicographic ordering which tend to have similar sets of neighbors. This technique together with an optimized Greedy algorithm can produce a new combinational algorithm that can be used to compress Web graph more efficiently.
Keywords*: Web graph, compression, similarity, Greedy algorithm.*

INTRODUCTION

One of the research fields that have received considerable scientific attention in recent years is the attempts in making compressed data structures. Some successful case-study include: text indexing [1], dictionaries, trees, and graphs [2]. These are rarity since successful compressed data structures tend to develop more slowly compared to that of uncompressed data structures. The primary motivation in the development of compressed data structure is its potential for storage of a larger amount of data in the main memory. Undoubtedly, one of the best known and meaningful compressed data structures is the Web. The development of the Web has been irregular and un-concentrated processes containing enormous resources of connected documents, while lacking any logical organisation. It is for this reason that the Web is perceived to be a data graph. Such a graph is a huge and unlabeled graph that connects the Web pages to each other. In this way, the Web pages are graph nodes and the hyperlinks are edges.
Analysis of the Web graph facilitates categorisation of the pages, recognition of Web communications and societies, as well as modelling and simulation of the Web graph making process.
It is estimated that graph for Websites[1] like Yahoo, Google, and Bring contain about 21 to 59 billion graph nodes. The magnitude of the problem can be easily indicated by supposing 50 billion nodes and 20 edges for each node, then, we are facing with one trillion edges, which is only a part of the huge Web graph. Both storing and processing of such a massive datasets necessitate the Web graph to be compressed. Research in this field has shown that the Web graph is compressible [3, 4]. Boldi and Vigna research on compressibility of Web graph has shown that web pages can be ordered lexicographically by URLs where the sources of pages have similar neighbourhoods. This technique offers beneficial properties:

- Similarity: source pages in the lexicographically ordering tend to have similar neighborhoods [3].
- Locality: most of links are intra-domain, so they point to the nearby pages in the lexicographically ordering [3].

On the basis of these properties, the BV algorithm compresses the Web graph by using a few bits for each link. Thus, the compression uses fewer bits in storing the links among the pages. In the first part of the paper, I will use lexicographically ordering with an allocated ID to each URL. In the second part, I will determine the most similar URL utilizing the property of similarity. Finally, I will merge two nodes with the most similarity by employing a kind of Greedy algorithm, and then new node is added to the list of nodes. This exercise is repeated until a desirable compression is achieved.

[1]http://www.worldwideWebsize.com, May 2010

In order to assess the efficacy of method, I will compare the result of the exercise with other algorithms. The result of the exercise will show whether the proposed techniques is more efficient or not.

The rest of the paper is organized in the following way: Section 2 provides a review of the related studies. Section 3, explains the method for computing the most similarity of URLs. Section 4, presents a kind of Greedy algorithm and suggests for an improved way for its usage. Section 5, defines the new and improved algorithm and gives a method for accessing the compressed graph without decompressing it. Section 6, presents the results of the exercises. The paper ends with a brief conclusion.

RELATED WORKS

In recent year, Web graph compression has been an intense research area, because of rapid growth of the Web. Adler and Mitzienmacher [5] have found that nodes with similar sets of neighbors, and presented a method for Web graph compression. For the first time, Randall *et al.* [6] have used lexicographically ordering as an approach for this purpose. They have shown that there are many links in the same host, and most of the related pages had similar links. Boldi and Vigna [3, 7] exploited beneficial properties of the lexicographically ordering to compression. They have identified the importance of the locality factor and presented a novel ordering method by combining host information and lexicographically ordering [8]. Based upon these findings, Raghavan and Malina [9] categorized the Web graph in a hierarchical structure. They have designated node S for approach to the locality property. Suel and Yuan [10] have used the structural categorization to identify local and global links. Also, Apostolico and Drovandi [11] have introduced a method on the basis of BFS, which additionally included gap encoding. Buecher and Chellapilla [12] have used data mining structures for simplifying the compression problems. They have shown the feasibility of complete bipartite sub-graphs by use of the technique of duplicated sub-graphs, and then replacing the edges of these sub-graphs with connection to a virtual node to whole nodes in both parts of completed sub-graph. Their method proved to be efficient by 2 bites by combining gap encoding technique with lexicographically ordering.

More recently, Chierichetti*et al.* [13] used BV algorithm and Shingle ordering [14] to compress social network graphs. All the methods described above have exploited the bit-edge coefficient to evaluate the degree of compression. Furthermore, some methods of compression use information obtained from adjacency matrix as a criteria for the purpose of comparing efficiency methods. For instance, Raghavan and Malina [9] have made some comparisons with other studies based on the method proposed by Randall *et al.* [6] using 6 different adjacency matrix. Boldi and Vigna [3,7] have also presented the Lazy iteration for the assessment of the compressed links in Web graph. In their method, the time need for accessing to each link equals few hundred nano of a second.

Similarity

In this section, I will first define the concepts which are used in the paper, and then, will explain the adjacency list and the method of computing the most similarity between its two members.

Definitions

In the present paper the Web graph is indicated as directed graph $G=(V,E)$ where V is a set of nodes and $E$is a set of edges. I also refer to V by $V(G)$ and E by $E(G)$. For an edge $e = (u, v)$, I refer to u as the source of e and v as the destination of $e.(u, v) \neq (v, u)$.Node u is neighbor of node v if there is an edge from u to v. I refer to neighbors set of node u by $N(U)$. We also refer the out-degree of edges by number of exited edges.

Graph adjacency list

One of the ways to show graph is to use adjacency matrix or list. Thus, Web graph can be presented as adjacency list by dividing Web graph into some data sets composed from URLs. Graph nodes are numbered from zero to V-1 according to lexicographically ordering. Each node is presented together with its exiting degree and a list of its neighbors. Table 1 shows such a graph presentations.

Computing the most similarity

Similarity is one of the most important Web graph properties which has been widely used for compression and is the corner stone of my proposed algorithm. However To achieve an improved compression in algorithm, it is essential to determine the amount of similarity among a selected members of adjacency matrix and their neighbours. To this end, I used the Jaccard equation [14] shown below:

$$j(A,B) = |A \cap B| / |A \cup B| \quad (1)$$

In fact, this equation gives the amount of similarity between any two datasets. If we consider out-neighbors of a node as a set, then the goal should be to find another node with maximum similarity to the out-neighbor set from the selected node.

Table 1: Graph adjacency list

Node	Out-degree	Neighbors
…	…	…
10	8	10, 11, 12, 13, 225, 340, 650, 1001
11	7	11, 12, 116, 340, 540, 1001, 3010
12	0	
13	5	10, 13, 22, 45, 225
…	…	…

Optimizing a kind of Greedy algorithm

In this section, I will briefly describe the graph and summarize the method pioneered by Shrivastava and navlakha [15]. It is clear that some nodes in graphs have similar neighbors. For example, in the Web graph, there are pages with similar URLs, or there are relationships among members of a social network. The summarizing operations are done according to these similarities between nodes. At the first stage, nodes and edges are categorized in two groups as: super nodes and super edges. Super nodes and super edges are composed of nodes and edges. Each super node includes a set of nodes. When there is an edge between two super nodes, then all components of the both super nodes are linked in the main graph by an edge. It is sometimes needed to make or remove some edges from the graph in order to add super edges and super nodes.

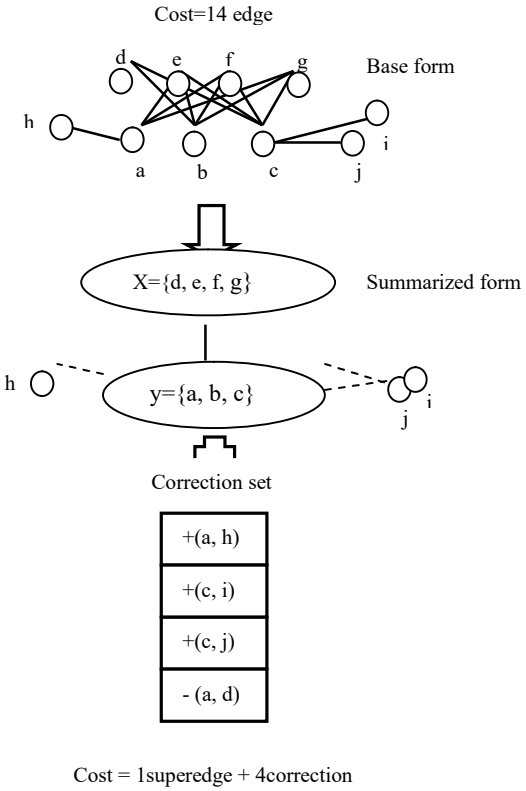

Figure 1. Graph summarization and making Correction set [15].

Such edges are stored in a set called 'correction'. Figure 1 show a graph which is converted into two super nodes and one super edge. Correction set indicates that which edges has been added or removed. Edge *da* shown as dotted line in the main graph, doesn't exist in the main graph, however, it is necessary for construction of the summarized graph structure. Such edges are shown in the correction set by the minus sign. Also, there are some edges in the main graph, which are omitted from the summarized structure of graph. The edges *ha, hi,* and *hj*, shown by dotted lines in the summarized graph of Figure 1, belong to this kind of edges. These edges are shown in the correction sets by the plus sign. Number of edges in the main graph is 14, which can be labelled as storing cost (number of edges). Thus, there are a super edge in the summarized structure and 4 edges in the correction set. Thus, the storing cost declines to 5.

Before making desirable Greedy algorithm, which merges two nodes in each step, it is necessary to identify a criterion for evaluating the capability of merging. When the cost of edge u is determined according to number of its exited edges, shown by C_u, and the cost of construction a super node is equal to total number of its super edges and edges of the related Correction set, shown as C_w, then the merging capability can be estimated as follows:

$s(u,v) = (c_u + c_v - c_w)/(c_u + c_v)$ (2)

At the first step, a node (node u)is selected randomly from the node list. At the next step, another node (node v) would be selected from the rest of node list, which has maximum value for $s(u,v)$. If amount of the calculated $s(u,v)$ in the previous step is positive, the two nodes would be merged and the new node (node m) will be added to the nodes list; otherwise, the node u will be omitted from the nodes list. Above steps will be repeated, while the nodes list would not been empty.

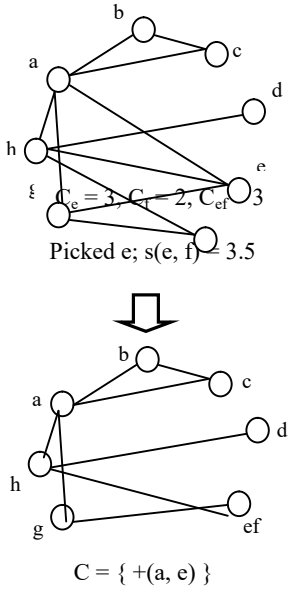

Figure 2.One step of Greedy algorithm [15].

In Figure 2, the node e is selected randomly in the first step. $s(e, v)$ is calculated (v refers to any remainder node) and the node f with maximum value of s is selected as the merging node. In the second step of Figure 2, the new graph is shown after one time running of the Greedy algorithm. However, a careful observation of the algorithm reveals some weaknesses. This algorithm is not rapid, because in each step, after a node is selected, the $s(u,v)$ should be calculated for each remainder nodes. This limitation had taken a great deal of my effort to optimize the algorithm. Indeed, my purpose was to find the maximum values of $s(u,v)$ more rapidly. In order to achieve this goal, I evaluated two conditions, listed below respectively, so as to find the node v for merging with the selected node u:

1- The node v would be selected for merging only if it has the most similar neighborhood with the node u.
2- If there is more than one node with similar neighbours for the node u, the selected node for merging should have the smallest degree.

The proposed algorithm

The graph compressor algorithm introduced in this section is composed of gap encoding and another optimized algorithm described in previous part. Indeed, using this algorithm, the Web graph will be both summarized and compressed.

The steps of proposed algorithm

This algorithm includes following steps:

- Lexicographically ordering of the URLs and putting them in the set A.
- Allocating a number as an ID to each member of the set A.
- This step can be executed in two different ways (I have already indicated that the second way is more useful and efficient in compression):
  1- Selecting node u from the A set, randomly.
  2- Selecting a node with maximum out degree as the node u.

- Exploiting the Jaccard equation in order to find node *v* which has the maximum similarity with node *u* (if there is more than one node with maximum value of similarity, the node should be chosen as node *v* that has the smallest degree).
- Merging the two nodes *u* and *v* and producing the new node *uv*.
- Removing the nodes *u* and *v* from set *A* and adding the new node *uv*(the number allocated to *uv*, is 1 greater than the maximum number that has been used).
- Inserting remainder edges of merging into the correction set.
- Above process will be repeated until the set *A* become empty.
- Doing the gap encoding operation [3] for members of the Correction set.
- Using the code ζ for encoding remainder large numbers in the Correction set [7].

Figure (3-a) shows the directed graph *G.* The set *A* is made by the nodes of this graph: A={a, b, c, d, e, f, g, h}. Then numbers 1 to 8 are allocated to its members. At the first step of running the algorithm, the node *a*with out degree 4 is selected as the first node. Calculation of similarity between this node and the other nodes indicate that the nodes *g* and *h* have the most similarity with it. Because the out degree of both nodes *g* and *h* is equal to 3, there would be no difference whether to select either one as the merging node. Hence, the node *g* is selected. Therefore, according to the algorithm, the nodes 1 and 7 are merged and removed from the *A* set, and the new node of *ag*, the node number 9, is inserted. The remainder edges of such merging will be added to the correction set. Figure (3-b) shows the resulted graph after the first operation of merging. At the second step, the node *h* is chosen, and the node *b* is selected as the merging node in order to estimate of the similarities. These two nodes are removed from the set *A* and node 10 is added to the set. This process continues until the set *A* become empty and the correction set is formed as follows:

Correction={(1, 3), (1, 8), (3, 4), (3, 7), (4, 2), (5, 1), (5, 4), (6, 8), (7, 1), (8, 2), (8, 5), (9, 2), (9, 6), (10, 3)} (3)

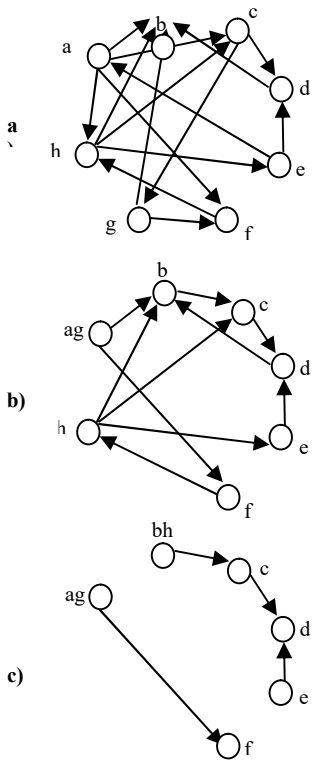

Figure 3.a)Directed graph G; b) the resulted graph after the first step of merging; c) the resulted graph after the second step of merging.

So far, the graph *G* is stored in the correction set and the compression in the merging phase is accomplished. Each member of the correction set represents an edge. Now, in order to achieve more compression, this set will be placed in a matrix with two columns. The first column includes the source nodes of the correction set, and the second one holds the neighbor set for each node. The described correction set is stored in a matrix labelled '*correction2'*, as (4):

$$Correction2 = \begin{bmatrix} 1 & 3,8 \\ 3 & 4,7 \\ 4 & 2 \\ 5 & 1,4 \\ 6 & 8 \\ 7 & 1 \\ 8 & 2,5 \\ 9 & 2,6 \\ 10 & 3 \end{bmatrix} \quad (4)$$

Afterwards, the encoding operation [3] is separately executed for sets of the second column, as well as for all members of the first column, the data is more compressed. Ultimately, the remainder large numbers after the gap encoding operation are encoded by the Boldi and Vigna's code [3]. Our study revealed that the efficacy of this code is different, depending either on the graph or ζ type.

Accessing to the compressed graph, without decompression

One of the advantages of the proposed compressing algorithm is that it can provide the opportunity for evaluating the compressed data without decompression. It can be easily accessed to the neighbors of each node by using the *correction2*, a matrix labelled *mergID* (which will be defined in the rest of the paper), and knowing the ID of the node. Indeed, the *mergID* matrix is a merged nodes matrix. Each row of this matrix includes the IDs of the two merged nodes. The first row represents the first newly produced node; the second row shows the second new one, and so on.

The *mergID* matrix is produced as below:

$$mergID = \begin{bmatrix} NodeID & NodeID \\ \vdots & \vdots \\ NodeID & NodeID \end{bmatrix} \quad (5)$$

The index of the first row of the matrix is equal to the allocated ID to the first new node. Similarly, the index of the second row is one more than the ID of the first new node (the first new node's ID is stored in a variable labelled *FirstNewID*).

Experiments

These experiments cover two main issues. *a*) Evaluation of the proposed method from the view point of reducing number of graph edges (the first step of compression); and *b*) assessing the degree of compression of the method.

Experiment data sets

The experiments have been done using five almost large graphs. These graphs were obtained from the SNAP[2] project, Slashdot social network from February 2009, Wikipedia voting network from Wikipedia, Epinion social network, Web graph of Stanford.edu and Gnutella peer to peer network from August 8 2002. Table 2 shows these graphs and their properties.

Table 2: The datasets states

Graph	*|V|*	*|E|*	*|V|/|E|*
Slashdot Zoo	82168	948464	0.08
Wikipedia Voting	7115	103689	0.06
Epinions	75879	508837	0.14
Stanford	281903	2312497	0.12
Gnutella08	6301	20777	0.3

Reduction of edges

Table 3 indicates the results of the method for reducing the edges of the graphs. The results revealed that near 15% of the graph edges were declined without any damaging to the main graph. It is noticeable that even though other methods may be more useful for reducing edges, the present method provides an optimized level of reduction for a better gap encoding and coding through the next step. Similar to some

[2]Stanford Network Analysis Package, http://snap.stanford.edu/data/

other methods, we also used the number of bits per edge as compression evaluating criterion. If selection of nodes would be on the basis of maximum out degrees in order to merging, compression may improved compared with using random selection (Table 3). The reason is that the number of neighbors increases with enhanced out degrees; therefore, the probability of more similarity among the neighbors and the given node will be increased (see Fig. 4). As Figures show, selection of nodes with maximum out degrees would intensify the reduction of edge numbers.

Table 3: Reduction of edge after step 1 of compression (random selection, max-degree selection)

Graph	*|E|*		
	Before compression	*After compression*	
		Random	*Max-degree*
Slashdot Zoo	948464	932675	920512
Wikipedia Voting	103689	90214	84824
Epinions	508837	506433	499355
Stanford	2312497	2300254	2288166
Gnutella08	20777	20177	18432

Compression assessment

In order to assessing the degree of compression, we implemented the proposed method on all graphs of table 2, and used the criterion of bit numbers per edge. Efficiency of various ζ codes has been also evaluated. Table 4 indicates the results of experiments. As can be seen in this table, codes $\zeta_4$and $\zeta_5$have better efficiencies compared with others. The best result from the method introduced by H. Maserrat and J. Pei [16] was obtained from the graph *Gnutella08* which was equal to 21.63 bits per edge. The best result of our study on the same graph was 9.61 bits per edge. It should be considered that our method only supports the out-neighborhood queries, while the mentioned study supports both the out- and in-neighborhood queries.

The number of node/the number of edge ratio (Table 2) for the graph *Wikipedia voting* equals to 0.06, which is less than other graphs. Table 4 shows that also the most efficient compression is related to this graph. Thus, the efficiency of the present method would be improved with enhancing the graph mass or reducing *the number of node/the number of edge* ratio.

Table 4: The number of bits per edge after compression

Graph	ζ_1	ζ_2	ζ_3	ζ_4	ζ_5	ζ_6
Slashdot Zoo	14.01	12.67	11.64	11.23	11.40	12.94
Wikipedia Voting	9.94	8.28	8	8.08	8.27	8.54
Epinions	17.25	16.34	16.07	15.92	15.80	18.11
Stanford	13.26	11.94	11.49	11.02	11.18	14.01
Gnutella08	12.63	10.26	9.78	9.72	9.61	10.03

CONCLUSION

In this study, a combination method for summarizing and compressing the Web graph is presented, using similarity criterion and optimizing a kind of Greedy algorithm. Comparing the results of our experiment with similar methods indicate that using the important property of similarity may have a crucial role in improving the results. Main properties of the proposed algorithm are:

1- Proper utilization of the important property of similarity in the Web graph.
2- Optimizing the Greedy algorithm and using it in the compression process.
3- Exploiting the summarization in order to optimized compression.
4- Reducing the costs of compression.

REFERENCES

1. G. Navarro and V. Makinen: *Compressed full-text indexes*. ACM Computing Surveys, 39(1) 2007,p. article 2.
2. J. I. Munro and V. Raman: *Succinct representation of balanced parentheses, static trees and planar graphs*, in IEEE Symposium on Foundations of Computer Science (FOCS), 1997, pp. 118–126.
3. P. Boldi and S. Vigna.*The webgraph framework I: Compression techniques*. In Proc. 13th WWW, pages 595–602, 2004.

4. G. Buehrer and K. Chellapilla. *A scalable pattern mining approach to web graph compression with communities*. InProc. 1st WSDM, pages 95–106, 2008.
5. M. Adler and M. Mitzenmacher.*Towards compressing web graphs*.In Data Compression Conference, 2001.
6. K. H. Randall, R. Stata, J. L. Wiener, and R. Wickremesinghe. *The link database: Fast access to graphs of the web*. In DCC, 2002.
7. P. Boldi and S. Vigna.*Thewebgraph framework II: Codes for the world-wide web*. In Data Compression Conference, 2004.
8. P. Boldi, M. Santini, and S. Vigna.*Permuting web graphs*.In Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph (WAW'09), Berlin, Heidelberg, 2009, Springer-Verlag.
9. S.Raghavan and H.Garcia-Molina*. Representing webgraphs*.In ICDE, 2003.
10. T. Suel and J. Yuan.*Compressing the graph structure of the web*.In Data Compression Conference, 2001.
11. A. Apostolico and G. Drovandi. Graph compression byBFS. Algorithms, 2(3):1031–1044, 2009.
12. G. Buehrer and K. Chellapilla.*A scalable pattern mining approach to web graph compression with communities*. In WSDM, 2008.
13. F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan.*On compressing social networks*.In KDD, 2009.
14. A. Z. Broder, M. Charikar, A. M. Frieze and M.Mitzenmacher.Min-wise independent permutations.*J.Comput. Syst. Sci*., 60(3):630–659, 2000.
15. N.Shrivastava, S.Navlakha and R.Rastogi. *Graph Summarization With Bounded Error*InSigmod, 2008*.*
16. H.Maserrat and J.Pei. *Neighbor Query Friendly Compression of Social Networks* In KDD, 2010