Partition based Graph Compression

Meera Dhabu, Dr. P. S. Deshpande, Siyaram Vishwakarma
Department of Computer Science & Engineering,
Visvesvaraya National Institute of Technology,
Nagpur – 440010 (India)

***Abstract*—Graphs are used in diverse set of disciplines ranging from computer networks to biological networks, social networks, World Wide Web etc. With the advancement in the technology and the discovery of new knowledge, size of graphs is increasing exponentially. A graph containing millions of nodes and billions of edges can be of size in TBs. At the same time, the size of graphs presents a big obstacle to understand the essential information they contain. Also with the current size of main memory it seems impossible to load the whole graph into main memory. Hence the need of graph compression techniques arises. In this paper, we present graph compression technique which partition graphs into subgraphs and then each partition can be compressed individually. For partitioning, proposed approach identifies weak links present in the graph and partition graph at those weak links. During query processing, the partitions which are required need to be decompressed, eliminating decompression of whole graph.**

Keywords— graph compression; su graph; partitioning

I. INTRODUCTION

Today, numerous large-scale systems and applications need to analyze and store massive amounts of data that involve interactions between various entities – this data is best represented as a graph; for instance, the link structure of the World Wide Web, group of friends in social networks, data exchange between IP addresses, market basket data, etc., can all be represented as massive graph structures. As witnessed in the core tasks of these applications graph patterns could help build powerful, yet intuitive models for better managing and understanding complex structure. Some of these application domains are [19]:

- World Wide Web. The Web has a natural graph structure with a node for each page and a directed edge for each hyperlink. This link structure of the Web has been exploited very successfully by search engines like Google [18] to improve search quality. Other contemporary research works mine the Web graph to find dense bipartite cliques, and through them Web communities [16] and link spam [05]. Recent estimates from search engines put the size of the Web graph at around 3 billion nodes and more than 50 billion arcs [14]. (Note that these are clearly lower bounds since the Web graph has been growing rapidly over the years as more of the Web gets discovered and indexed.) Thus, the Web graph can easily occupy many terabytes of storage.
- Social Networking. Popular social networking websites like Facebook, MySpace and LinkedIn cater to millions of users at a time, and maintain information about each user (nodes) and their friend-lists (edges). Mining the social network graph can provide valuable information on social relationships between users, the music, movies, etc. that they like, and user communities with common interests.
- IP Network Monitoring. IP routers export records containing source and destination IP addresses, number of bytes transmitted, duration, etc. for each IP communication flow. Recently, Iliofotou et. al. [12] proposed the idea of extracting Traffic Dispersion Graphs (TDGs) from network traces, where each node corresponds to an IP address and there is an edge between any two IP addresses who sent traffic to each other. Such graphs can be used to detect interesting or unusual communication patterns, security vulnerabilities, hosts that are infected by a virus or a worm, and malicious attacks against machines.
- Market Basket Data. Market basket data contains information about products bought by millions of customers. This is essentially a bipartite graph with an edge between a customer and every product that he or she purchases. Mining this graph to find groups of customers with similar buying patterns can help with customer segmentation and targeted advertising.

Several approaches have been proposed for the analysis and discovery of concepts in graphs in the context where graphs are used to model datasets. Modeling objects using graphs allows us to represent arbitrary relations among entities and capture the structural information. The utilization of richer and more elaborate data representations for improved discovery leads to larger graphs. The graphs are often so large that they cannot fit into the dynamic memory of conventional computer systems. Even if the data fits into dynamic memory, the amount of memory left for use during execution of the discovery algorithm may be insufficient, resulting in an increased number of page swaps and ultimately performance degradation. One of the main challenges for knowledge discovery and data mining systems is to scale up their data interpretation abilities to discover interesting patterns in large datasets. This paper addresses the scalability of graph-based discovery to monolithic datasets, which are prevalent in many real-world domains where vast amounts of data must be examined to find meaningful structures.

In [23], many challenges are faced by graph mining algorithms due to the huge size of graph. One issue is that a huge graph may severely restrict the application of existing

pattern mining technologies. Additionally, directly visualizing such a large graph is beyond our capability. In computer science, it is more important to understand the information embodied in abstract structures that are of our particular interests. For instance, how can we quantify the amount of information in the structure of graphs such as the Internet, social networks, and biological networks? How can we understand and utilize the "structure" of nonconventional data structures such as biological data, topographical maps, medical data, and volumetric data? Imagine a compressed graph, conserving the characteristics of the original graph. We can easily visualize it. The goal of compressing a graph is to make the high-level structure of the graph easily understood. Therefore, informative graph compression techniques are required and have wide application domains. Many graph compression techniques have been developed for compressing a web graph [7, 14, 25, 10, 4, 9]. In this paper we proposed partition based compression approach which helps in storing the compressed subgraphs on the systems that are located geographically apart. Thus it reduces the network traffic in distributed computing [6] since data will be available on local system itself. The aim of the proposed technique is to represent the data in compressed form while retaining the ability to answer the same queries as their uncompressed counterpart. We aim at representing graphs in highly compressed form, so as to manage huge instances in main memory.

The remainder of this paper is organized as follows. Section II reviews the background information as well as related work on graph compression. Section III presents the details of proposed partition based approach. Section IV presents the results of performance evaluation. Section V summarizes and concludes our paper.

II. BACKGROUND

The biggest challenge in graph compression is ever increasing demand of high compression ratio, which reduces memory requirement of a graph. A graph containing billions of nodes and trillions of edges cannot be stored in memory without compression and if we store it on disk then operations which need to be performed on this graph would require many disk I/O and disk seek operations which reduces algorithm performance drastically. Hence a graph needs to be divided to ensure that each partition is small enough to fit in main memory and thus reduces I/O operations significantly.

A. Problem definition

Given an undirected graph $G = (V, E)$, where V is set of vertices and E is set of edges in the graph G. We need to represent graph such that the compression ratio and bits per edge are maximum and minimum respectively. Compression ratio and bits per edge are given by the following formulae:

$$Percentage\ Compression\ or\ Compression\ Ratio = \frac{input\ graph\ size - output\ graph\ size}{input\ graph\ size} * 100$$

$$Bits\ per\ edge = \frac{size\ of\ output\ or\ compressed\ graph}{total\ edges\ in\ the\ graph}$$

B. Related work

In recent years many compression algorithms have been proposed. In [14] Gap encoding makes use of locality [8] property of web graph. Locality suggests that each list of successors should be represented as list of gaps. More precisely, if $S(x) = (s_1, s_2, \dots, s_k)$, then it can be represented as $(s_1 - x, s_2 - s_1 - 1, s_3 - s_2 - 1, \dots, s_k - s_{k-1} - 1)$. However; reference compression [14] technique exploits similarity property of web graphs. In this method, adjacency list $S(x)$, is represented as a *"modified"* version of some list $S(y)$, called the *reference list*. The difference $x - y$ is called the *reference number*. This results into *reference compression*, in which a sequence of bits, one bit for each successor in the reference list, tells whether the corresponding successor of y is also a successor of x. Nodes which are not covered by reference list are called *extra nodes*.

In differential compression, the differences with $S(y)$ are represented as a sequence of *copy blocks*. *Copy list* can be represented as an alternating sequence of 1 and 0-blocks, and specify the length of each block. This sequence of integers is preceded by a block count telling the number of blocks that will follow [14]. Consecutivity among extra nodes is frequent, hence to exploit this consecutivity, subsequences are isolated corresponding to integer intervals and number of integers in these intervals is called *length* [14].

In [8], an un-weighted graph $G = (V_G, E_G)$ can be represented as $R = (S, C)$ where $S = (V_s, E_s)$ is graph summary and C is set of edge corrections. Every node v in V_G belongs to a *super node* V in V_s which represents a set of nodes in G. A *super edge* $E = (V_i, V_j)$ in E_s represents the set of all edges connecting all pairs of nodes in V_i and V_j i.e. it simply collapse one bi-partite graph into two super nodes V_i and V_j and replaces all the edges by super edge between the super nodes. The edge correction C has parts $+e$ (edge to be added) and $-e$ (edge to be removed) which is considered during recreation of original graph.

In [7], Re-pair recursively finds pair of repeated symbols across all the lists and replace them by a new "non-terminal" symbol which has to be expanded later when extracting the lists. In [3], a directed bipartite clique $G = (V, E)$ can be transformed into a directed star. A directed bi-partite clique (S, T) is a pair of two disjoint set S and T such that $u \in S$ and $v \in T$ and there is a directed link from u to v in G. For a bi-clique $\langle S, T \rangle$ a new compressed graph $G' = \langle V', E' \rangle$ is formed by adding a new vertex x to the graph, removing all the edges in $\langle S, T \rangle$ and adding a new edge $ux \in E'$ for each u $\in S$ and new edge $xv \in E'$ for each $v \in T$.

In [13], an undirected graph $G = (V, E)$, where V is a set of nodes and E is set of edges, is represented using adjacency list method and thus $m + n$ space is required, where $n = |V|$ and $m = |E|$. But for simple undirected graphs G, it should be noted that the complement graph G^c of G is sufficient for representing G. For a very dense graph G, the size of the edge set of the complement graph may be much less than m. Therefore, the original graph is store in a data structure if $m \leq \frac{n(n-1)}{4}$, and the complement graph if $m > \frac{n(n-1)}{4}$. this

method requires $n + \min(m, m^c)$ space, where $m^c = \frac{n(n-1)}{2} - m$.

TABLE I. NOTATIONS

Notation	Description
w	Maximun no. of edges in bridge
neb_i	Set of neighbors of node *i*
S_1, S_2	Sets S_1 store neighbor of node *i* and set S_2 stores rest nodes
deg_i	Degree of node *i*
$G = (V, E)$	A graph G where V is set of nodes and E is set of edges

III. PROPOSED PARTITION BASED APPROACH

Compression allows more efficient storage and transfer of graph data, and may improve the performance of various algorithms by allowing computation to be performed in faster levels of computer memory hierarchies. Good compression requires using the structural properties of the graph, and hence first important step is to understand this structure. For example, in Web graphs, there appear to be natural clusters of related pages with similar connections.

In this paper we restrict our discussion to undirected graph but can be easily extended for directed graph. We use an undirected graph for modeling complicated structures which contains dense clusters and these dense clusters are connected with weak links called bridges. Proposed partition based compression algorithm exploits graph property *locality*.

Link locality has been independently observed and reported by several authors. For instance, Suel and Yuan [16] observe that on average, around three-quarters or 75% of the links from a page are to other page on the name domain/host. Given this observation, we attempt to partition graph into dense clusters and then these dense clusters are further compressed using reference compression technique.

We employ breadth first search algorithm starting from a randomly chosen node, say $x \in V$ which returns a connected component C. Now take a node say i from C and make two sets S_1 *and* S_2. Set S_1 contains all neighbors of node and set S_2 contains all the nodes in C except the nodes in S_1. A node x in S_1 with degree d is a bridge node of width w if the following conditions are satisfied:

1) $d_x - w$ *neighbors of node x are in set* S_1 *and exactly w neighbors are in set* S_2.

2) $(|neb_y| - 1)$ *neighbors of node y are in* S_2, *where* $y \in neb_x$ *and* $y \in S_2$.

If both the conditions are not satisfied then node may be shifted to set S_2 or if more than half neighbors of node x are in set S_2. We repeat this process for all the nodes in set S_1 and S_2 until we find a bridge between the two sets or no change in set S_1 and S_2. In this way we find bridge between set S_1 and S_2 which results into two subgraphs. Repeat the same procedure by choosing another random node from C until we get sufficiently small subgraphs. These subgraphs are compressed sequentially using reference compression technique [14].

Each subgraphs thus obtained after partitioning is compressed by applying reference compression algorithm. In this method, instead of representing adjacency list $S(x)$ for node *x* directly, it is represented as a *"modified"* version of some next list $S(y)$, called the *reference list*. The difference $y - x$ is called the *reference number*. Thus the reference compression results in a sequence of bits, one for each successor in the reference list, which tells whether the corresponding successor of node *y* is also a successor of node *x*. The representation of $S(x)$ with respect to $S(y)$ is made of two parts: a sequence of $|S(y)|$ bits, called the *copy list*, and the list of integers $S(x)/S(y)$, called the *list of extra nodes*. *Copy list* specifies which of the links contained in the reference list should be copied: it will contain 1 at the i^{th} position; iff the i^{th} entry of $S(y)$ also appears in $S(x)$[14].

For each node *i* in a subgraph s, we find reference node *j* (node which has maximum number of common successors with node *i*). we consider *reference_width* for finding reference node. r*eference_width* can be fixed or can be equal to size of subgraph.

For reference node, we calculate *reference_number*, *copylist* and *extra nodes*. c*opylist* is further compressed as a sequence of *copyblock* which contains the information about the number of 1's and 0's appearing in *copylist* alternatively. Further *extranodes* are compressed since there is consecutively among *extranodes*. Once all the nodes are covered in subgraph we take next subgraph for compression.

A. Algorithm

In this section pseudo-code for partitionven.ing the large graph is given. Function *check_condition_1* and *check_condition_2* will return *"1"* if *condition 1* and *2* mentioned in *section III* is true for node *k*.

Arrange all nodes of *G* in decreasing order of *degree*.

```
Procedure PartitionGraph(G, w)
begin
while (w > 0)
{
   for each node i ∈ V
   {
     S1 = i ∪ nebi;
     S2 = V − S1;
     while (no change in S1 and S2 or bridge is not found)
     {
        for each node k ∈ S1
        {
          flag_1 = check_condition_1(S1, S2, k)
          if (flag_1 == true)
          {
            flag_2 = check_condition_2(S1, S2, k)
            if (flag_2 == true)
            {
            a.  Node k is a bridge node of width w.
            b.  Remove all w edges between k and S2.
            }
```

else if (more than $\frac{|neb_k|}{2}$ *nodes are in* S_2*)*
{
$S_1 = S_1 - k;$
$S_2 = S_2 \cup k;$
}
for each node $k \in S_2$
{
flag_1 = check_condition_1(S_2, S_1, k)
if (flag_1 == true)
{
flag_2 = check_condition_2(S_2, S_1, k)
if(flag_2 == true)
{
a. Node k is a bridge node of width w.
b. Remove all w edges between k and S_1.
}
else if (more than $\frac{|neb_k|}{2}$ *nodes are in* S_1 *)*
{
$S_2 = S_2 - k;$
$S_1 = S_1 \cup k;$
}
}
}
w--;
}end while;
end-begin.

IV. Performance Evaluation

In this section, we present experimental results. We have performed experiments on 2.10 GHz Intel core i3 processors with 4GB main memory, running on 32-bits Windows 7 platform. Proposed algorithm is implemented in Java. We performed experiment on synthetic datasets generated using graph generator. Details of the graph dataset used for experiments are given in Table II.

A. Graph Partitioning

A graph having 9985 nodes and 123416 edges is partitioned into 354 subgraphs, size of each subgraph varies from 26 to 30 nodes both inclusive with bridge width equal to 3. Whereas number of bridges is 510, among these 358 bridges are of width one, 98 bridges are of width two and 54 bridges are of width three. On the other hand, a graph of 1979 nodes and 24340 edges is partitioned into 71 subgraphs, size of each subgraphs again vary from 26 to 30 both inclusive where bridge width is three. Number of bridges is 98 among these 66 bridges are of width one, 20 bridges are of width two, 10 bridges are of width three.

TABLE II. Details of Graph Dataset

Id	No. of nodes	No. of edges
G_1	1979	24340
G_2	4993	61392
G_3	9985	123416

B. Effect of different parameters on compression ratio

In Fig. 1, *y*-axis represents compression ratio and *x*-axis represents reference width *w* [9]. Different reference widths are 3, 5, 7 and a *subgraph*. Reference width equal to the subgraph means all the nodes in the subgraph will be considered in search of reference node [10]. Fig.1. shows compression ratio without copy blocks for different graph size. Compression ration increases slowly with the increase in reference width. Fig. 2 shows compression ratio with copy blocks for different graph size. Fig. 3 shows compression ratio with copy blocks and extra nodes for different graph size. From Fig. 2 and 3 can observe that the compression ratio increases rapidly with the increase in reference width. When reference width is equal to subgraph compression ratio is maximum. For all reference width, compression ratio of the graph with 9985 nodes and 123416 edges is higher among all graphs. Hence ratio increases with increase in number of nodes and edges i.e. we get better compression ratio for dense graphs.

Boldi and Vigna [14] have given the best algorithm ever which takes 2 to 3 bits per edge for a graph of size 18.5 million nodes and 300 million edges. S. Raghavan [21] has shown that super node and super edge representation takes 5.07 bits per edge for average over 25 million, 50 million, 100 million nodes. Broder [1] showed that a graph of 200M nodes and 1.5G edges requires 37.87 bits per edge.

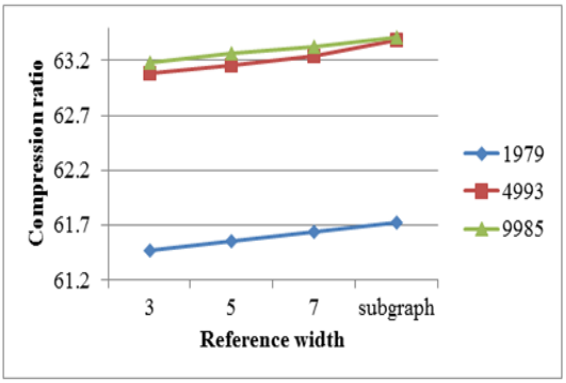

Fig. 1. Compression ratio (without copy block) v/s reference width *w*.

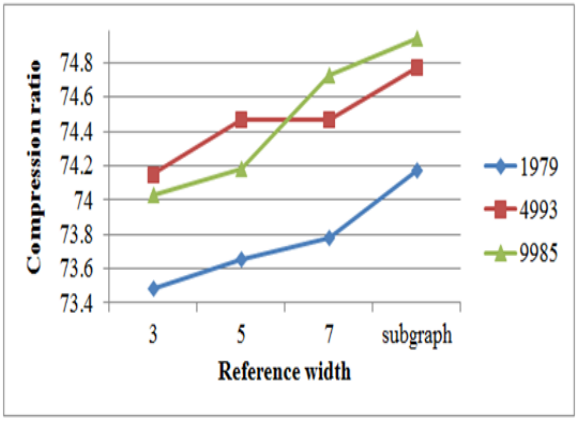

Fig. 2. Compression ratio (with copy block) v/s reference width *w*.

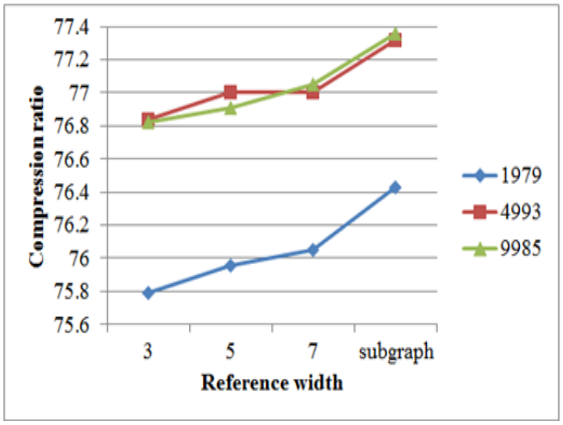

Fig. 3. Compression ratio (with copy block and extra nodes) v/s reference width *w*.

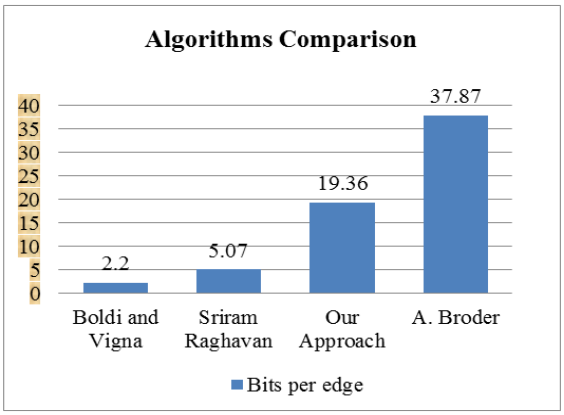

Fig. 4. Comparison of Boldi and Vigna, Sriram Raghvan, and A. Broder with Partition based reference compression approach.

V. Conclusion & Future Work

In this paper we proposed an effective solution in the form of a partitioning approach, to one of the main challenges for graph-based knowledge discovery and data mining systems, which is to scale up their data interpretation abilities to discover interesting patterns in large graph datasets. We observed that for partition based reference compression approach, compression ratio increases with increase in reference width and it is maximum when reference width is equal to size of subgraph. Moreover it helps in distributed computing by reducing network traffic and storage burden on single system.

Possible future enhancement to the proposed approach is reducing partitioning time which increases sharply with the increase in graph size. Since we run BFS algorithm for each node in the graph which gives connected component but we can ignore the nodes which are in the partition and cannot be partitioned further.

Our algorithm is sequential i.e. first graph partitioning is done and then reference compression algorithm is applied. This causes re-loading of each partition for the compression. It can be improved by compressing the partition when it cannot be partitioned further.

References

[1] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, “Graph structure in the web”, Journal of Computer Networks, pp. (1-6), 2000.

[2] A. Inokuchi,T. Washio and H. Motoda, “An apriori-based algorithm for mining frequeunt substructures from graph data”, In PKDD'00, pp. 13-23, 2000.

[3] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: A stream database for network applications" In SIGMOD, pp. 647-651, 2003.

[4] D. Blandford, G. Blelloch, I. Kash, “Compact representation of separable graphs” In Proc. of the 14th SODA, pp. 679-688, 2003.

[5] D. Gibson, R. Kumar, and A. Tomkins,"Discovering large dense subgraphs in massive graphs" In VLDB, pages 721- 732, 2005.

[6] F. Zhou, “Graph Compression”, Department of Computer Science and Helsinki Institute for Information Technology HIIT, pp. 1-12.

[7] F. Claude and G. Navarro,“Fast and compact web graph representations”, In Proceedings of 14th International Symposium on String Processing and Information Retrieval, Santiago, Chile, pp. 105-116, October 29-31, 2007.

[8] H. Khalili, A. Yahyavi and F. Oroumchian, “Web graph pre compression for similarity based algorithms”, Proceedings of the 3rd International Conference on Modeling, Simulation and Applied Optimization Sarjah, U.A.E., pp. 20-22, January 2009.

[9] J. Guillaume, M. Latapy, L. Viennot, “Efficient and simple encodings for the web graph”, In Meng, X., Su, J., Wang, Y. (eds.) WAIM 2002 LNCS, vol. 2419, pp. 328–337, Springer Heidelberg.

[10] K. Bharat, A. Broder, M. Henzinger, P. Kumar, S. Venkatasubramanian,“The connectivity server: fast access to linkage information on the web”, In Proc. of the 7th WWW, pp. 469-477, 1998.

[11] K. Randall, R.Stata, R. Wickremesinghe and J. Wiener, “The link database: fast access to graphs of the web”, Research Report 175, Compaq Systems Research Center, Palo Alto, CA, 2001.

[12] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, "Network monitoring using traffic dispersion graphs (tdgs)" In IMC, pp. 315–320, 2007.

[13] N. Larsson and A. Moffat, “Off-line dictionary-based compression”, Proceedings of the IEEE, Vol. 88, No. 11, November 2000.

[14] P. Boldi and S. Vigna, “The WebGraph framework I: compression techniques”, In Proceedings of the 13th international conference on World Wide Web, ACM, pp. 595-602, USA, 2004.

[15] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks" In KDD, 2006.

[16] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communies" In WWW, pp. 1481-1493, 1999.

[17] R. Rathi, D. J. Cook and L. B. Holder A Serial Partitioning Approach to Scaling Graph-Based Knowledge Discovery. American Association for Artificial Intelligence, pp. 188-193. 2002.

[18] S. Brin and L. Page, "The anatomy of a large search hypertextual web engine" In WWW, pp. 107–117, 1998.

[19] S. Navlakha, R. Rastogi and N. Shrivastava, Graph Summarization with Bounded Error. SIGMOD’08, June 9–12, 2008, Vancouver, BC, Canada.

[20] S. Perugini, M. Gon Calves, and E. Fox, "Recommender systems research: A connection-centric survey". J. Intell. Inf. Syst., 23(2):107–143, 2004.

[21] S. Raghvan and H. Garcia-Molina,“Representing web graphs”, In Proc. of the IEEE International Conference on Data Engineering, 2003.

[22] T. Suel and J. Yuan, “Compressing the graph structure of the web”, In Proc. of the IEEE Data Compression Conference, pp. 213-222,2001.

[23] X. Yan and J. Han,“gSpan:Graph Based Substructure Pattern Mining”, Technical Report UIUCDCS-R-2002-2296, Department of Computer Science, University of Illinois at Urbana-Champaign, 2002.

[24] Y. Asano, Y. Miyawaki and T. Nishizeki, “Efficient compression of web graphs”, In Proc. 14th Conference on Computing and Combinatorics (COCOON) , LNCS 5092, pp. 1-11, 2008.

[25] Y. Asano, T. Ito, H. Imai, M. Toyoda, M. Kitsuregawa, "Compact encoding of the web graph exploiting various power laws: statistical reason behind link database". In: Dong, G., Tang, C.-j., Wang,W.(eds.) WAIM 2003 LNCS, vol. 2762, pp. 37–46, Springer Heidelberg.

AUTHORS PROFILE

P. S. Deshpande received M.Tech in Computer Science from IIT, Bombay (India) and received Ph.D. from Nagpur University (India). He is currently working as a Associate Professor in the Department of Computer Science & Engineering at Visvesvaraya National Institute of Technology, Nagpur (India). His research interests include Data Bases, Data Mining, and Pattern Recognition.

Meera Dhabu is working as a Assistant Professor in the Department of Computer Science & Engineering at Visvesvaraya National Institute of Technology, Nagpur (India) and persuing her Ph. D. in Computer Science. Her areas of interests are graph mining and soft-computing.

Siyaram Vishvakarma received M. Tech. in Computer Science from Department of Computer Science & Engineering, Visvesvaraya National Institute of technology, Nagpur (India). His area of interests are graph mining and pattern recognition.