Web Archiving: Organizing Web Objects into Web Containers to Optimize Access

(Technical Report UMIACS-TR-2007-42)

Sangchul Song and Joseph JaJa
Department of Electrical and Computer Engineering
Institute for Advanced Computer Studies
University of Maryland, College Park

Abstract

The web is becoming the preferred medium for communicating and storing information pertaining to almost any human activity. However it is an ephemeral medium whose contents are constantly changing, resulting in a permanent loss of part of our cultural and scientific heritage on a regular basis. Archiving important web contents is a very challenging technical problem due to its tremendous scale and complex structure, extremely dynamic nature, and its rich heterogeneous and deep contents. In this paper, we consider the problem of archiving a linked set of web objects into web containers in such a way as to minimize the number of containers accessed during a typical browsing session. We develop a method that makes use of the notion of PageRank and optimized graph partitioning to enable faster browsing of archived web contents. We include simulation results that illustrate the performance of our scheme and compare it to the common scheme currently used to organize web objects into web containers.

1. Introduction

An unprecedented amount of information encompassing almost every facet of human activity across the world is currently available on the web and is growing at an extremely fast pace. In many cases, the web is the only medium where such information is recorded. However, the web is an ephemeral medium whose contents are constantly changing and new information is rapidly replacing old information, resulting in the disappearance of a large number of web pages every day and in a permanent loss of part of our cultural and scientific heritage on a regular basis. A number of efforts, currently underway, are trying to develop methodologies and tools for capturing and archiving some of the web's contents that are deemed critical. However there are major technical, social, and political challenges that are confronting these efforts. Major technical challenges include automatic tools to identify, find, and collect web contents to be archived, automatic extraction of metadata and context for such contents including linking structures that are inherent to the web, the organization and indexing of the data and the metadata, and the development of preservation and access mechanisms for current and future users, all at unprecedented scale and complexity.

Leaving aside dynamic and deep contents, web contents involve a wide variety of objects such as html pages, documents, multimedia files, scripts, etc., as well as, linking structures involving these objects. While the size of most Web pages is small, the total number of web pages on a single web site can range from one to several millions. For example, as of Oct 30, 2006, Wikipedia.org alone claims to have about 1.4 million articles [7], each making up a distinct Web

page. A critical piece of web archiving is to capture the linking structures and organize the archived pages in such a way that future generations of users will be able to access and navigate through the archived web information in the same way as in the original linked structure. Note that by that time, the archived web contents may have migrated through several generations of hardware and software upgrades, including migration through different types of media, different file systems, and different formats.

In this paper, we address the problem of how to organize the web objects so that we will be able to navigate through the linking structure of the web objects as effectively as possible. Since the majority of web pages tend to be small, they are typically aggregated into relatively large containers as the objects are accessed during the crawling process. An emerging standard for such containers is the WARC format [4], which evolved from the ARC container format developed by the Internet Archive, currently the world's largest internet archive. Moreover, many Web crawlers and access tools, such as Heritrix [29], NutchWAX [3], Wayback [2], WAXToolbar [5] and Wera [6], assume this format.

Given a set of WARC containers that hold an archived linked set of web objects, a future browsing process of the archived objects starts with a web object defined by a seed link, followed by navigation through the linked structure until the desired web object is found. Our goal is to organize the web objects into containers so as to minimize the number of containers needed to complete a typical browsing process. We develop an algorithm that assigns web objects to containers by performing an initial link analysis on the given linked structure, followed by a partitioning process that leads to an efficient solution to this problem. We show that our method enables effective navigation through the archived linked structure and compare its performance to the dominant scheme in use today.

We start in Section 2 by describing the previous work related to our problem, followed by developing and justifying our method in Section 3. We apply our method to two web site examples and examine the performance gains achieved by our method in Section 4. We conclude in Section 5.

2. Related Work

We review in this section the possible storage formats for archiving web contents and a couple of techniques in link analysis and graph partitioning which will form the core of our method.

2.1. Archival Storage

In order to organize and store Web objects in an archive, several methods have been proposed and are currently in use. A straightforward method (such as the one described in [1]) is based on using the local file system where the target Web material is copied object by object to the local file system, maintaining the relative structure among the objects. For future access, the html tag 'file' can replace the 'http' tag in the original object. We can then use the local file system for navigation through the archived Web material. For example,'http://www.example.org/index.html' can be rewritten as 'file:///archive/2007.08.01/www.example.org/index.html'. It is relatively easy to set up and run this type of web archiving and the retrieval process is carried out using local file access mechanisms. However, there are several problems in using this method for web archiving including its limited scalability to what the local file system can handle, and the difficulty to preserve the contents over time as they are tightly coupled to the specific file system. Moreover, this strategy requires modifications to the original contents, and thus the strict faithfulness to the original contents cannot be maintained in most cases [26].

The second approach extracts documents from the hypertext context and reorganizes them in a different format while setting up different access mechanisms. For example, a small set of Web pages can be converted into a single PDF document. However, this strategy makes sense mainly for specific objects that were originally created independently of the Web. Although it is possible to maintain the hypertext structure within the converted documents, for the broader range archiving, this approach loses the hypertext structure between multiple such documents [26].

The most popular method currently in use by most Web archives, including the Internet Archive, stores Web objects in WARC [4] container files. A WARC file holds a set of harvested Web files, each with its own auxiliary metadata. The size of a WARC file can vary up to hundreds of megabytes (usually 100~500MB). Typically, an external indexing server is maintained to provide the mapping between hyperlinks inside a WARC file and the location of the archived object that the hyperlinks point to. For example, if, inside a WARC file, there is a Web page archived on September 24, 2007 which has an outgoing hyper link with a tag <a href="http://www.example.org/images/welcome.jpg>, the indexing server could return in response to the tag and date something like '20070924082031-00007.warc' and '1463539' which are the WARC file name and the offset in the WARC file, respectively. In this paper, we will also assume that web files are placed in such containers such that a certain upper bound on the size of the container is assumed.

2.2. Graph Partitioning Techniques

Web material can be considered as a graph (web graph) where each constituting Web page is represented by a vertex, and each incoming/outgoing link corresponds to a directed edge. Once represented as a graph, the web graph can be partitioned into multiple subgraphs using one of existing graph partitioning techniques. The basic goal of a minimum edge-cut partitioning is to minimize some defined cost on the edges connecting the partitions. There are many ways to define the external cost of graph partitioning but the two notions most widely used are the maximum weight of the edges between vertices which lie on different partitions, and the total weight of all the edges connecting distinct partitions. Although the graph partitioning problem is known to be NP-complete, many heuristic algorithms have been developed that find very good partitions in practice [10, 11, 16-19, 21, 24, 27, 28, 32]. However, for our application, we will require additional constraints, which cannot necessarily be handled by many of the well-known graph partitioning algorithms. We review here some of the algorithms that can be used to solve our graph partitioning problem that will be defined formally in Section 3.

Perhaps the best known graph partitioning algorithm is the Kerninghan-Lin algorithm [24], where the partitioning process starts with an arbitrary partition, and then proceeds to decrease the external cost by a series of interchanges of subsets of the partitions repeatedly until no further improvement is possible. To avoid local optimality, the algorithm is applied repeatedly to obtain a number of locally optimum partitions among which the best partition is chosen. Although Fiduccia and Mattheyses [12] later improved the performance of the Kerninghan-Lin algorithm, their algorithm is considered computationally expensive especially if the graph is large, which is clearly the case for our application.

In order to cope with large graphs, researchers devised multilevel graph partitioning schemes [10, 11, 17, 21, 32] where the algorithms reduce the size of the graph (or "coarsen" the graph) by collapsing vertices and edges, partition the resulting smaller graph, and then "uncoarsen" it to construct a partition for the original graph. While the multilevel scheme was mainly developed and used to improve the partitioning performance of a large graph at the expense of worse partition quality [32], more recent multilevel algorithms, such as in [10, 11, 17, 21], further refine the partition during the uncoarsening phase, thus obtaining a partition quality that is comparable

or even better than other existing techniques [19]. The Kerninghan-Lin algorithm is often used as the refinement algorithm.

2.3. Link Analysis Technique – PageRank

PageRank [30] is a link analysis algorithm that assigns a numerical weight to each element of a hyperlinked set of documents, such as web material. Intuitively, a web page with a higher PageRank should have a higher probability of being visited. The intuition behind PageRank is that if page *u* has a link to page *v*, then page *u* is implicitly conferring some importance to page *v*. In other words, page *u* can be thought as voting for page *v*. The more votes a page receives, the more important it is considered. However, not every vote counts equally: votes cast by pages that are themselves "important" weigh more heavily and help other pages become more "important".

In the ideal model, the PageRank value, PR(u), for page *u* can be expressed as:

$PR(u) = \sum_{v \in I_u} p_{vu} PR(v)$, where I_u is the set of pages with links to page *u*, and p_{vu} is the probability that a random surfer visiting page *v* jumps to page *u*. Since it is not possible to know the exact value of p_{vu}, p_{vu} is usually set to 1/out_degree(v), that is, all outgoing links from *v* are assumed to be equally likely.

However, the ideal model has two problems. The first problem is the presence of dangling pages that shut the surfer when visited. A solution to the problem is to patch dangling pages by artificially placing outgoing links from each dangling page to all the other pages. Each artificial link can be given either equal probability of $\frac{1}{\mathrm{N}}$ (N: total number of pages), or personalized probability which records a generic surfer's preference for each page. The second problem with the ideal model is that the surfer can get trapped by a cyclic path in the Web Graph. Brin and Page [9] suggest enforcing irreducibility by adding a new set of artificial transitions that, with low probability, jump to all nodes. Mathematically, this corresponds to the following equation:

$PR(u) = \frac{1-d}{N} + d \sum_{v \in I_u} p_{vu} PR(v)$, where N is the total number of pages, and $(1 - d)$ is the probability the random surfer jumps to a random page without a link.

We note that this equation is slightly different from the original PageRank equation as proposed by Brin and Page [9]. The original equation, $PR(u) = 1 - d + d \sum_{v \in I_u} p_{vu} PR(v)$, has brought up some confusion since, unlike the inventors' claim, the sum of all PageRanks is not one, but *N*. The above scaled version, however, leads to $\sum PR(v) = 1$, and each PageRank can be thought as a probability. In the above equation, the parameter d is called the *damping factor* which can be set somewhere between 0 and 1. As suggested in [30] and [9], we use d = 0.85 in our work which will be further described in the next section.

If we let $G = (V, E)$ be a web graph, and **A** the modified adjacency matrix of G defined by:

$$A_{ij} = \begin{cases} \frac{1-d}{N} + \frac{d}{O_j}, \text{if } (j,i) \in E \\ \frac{1-d}{N}, \text{otherwise} \end{cases}$$, where O_j is the number of out-links from page j.

and let $\mathbf{P}$ be an N-dimensional column vector of PageRank values, then $\mathbf{P}$ can be expressed by the following matrix equation: $\mathbf{P} = \mathbf{AP}$

This is the characteristic equation of the eigensystem whose solution is the eigenvector corresponding to the eigenvalue of one. Furthermore, **A** can be considered as a stochastic matrix that is also irreducible and aperiodic, due to the modifications we performed earlier to avoid dangling nodes and cyclic paths. Therefore, by the Ergodic theorem of Markov chains [31], a finite Markov chain defined by the stochastic transition matrix A has a unique stationary probability distribution. This implies that, starting with any initial value of $\mathbf{P}$, we can iterate the application of the matrix **A** to **P**, and $\mathbf{P}$ will converge to a steady-state probability vector, which in turn is the eigenvector of $\mathbf{A}$ corresponding to the eigenvalue of one. In practice, a well known mathematical technique called power iteration [15] can be used to efficiently determine $\mathbf{P}$.

As will be discussed further in the next section, our link analysis technique is based on the PageRank algorithm. However, unlike the PageRank algorithm that assigns a weight to each page, we assign a weight to each link, which will then be used to partition the graph.

3. Our Method

As discussed earlier, the most popular storage method for Web archiving is to use containers where each container holds a number of Web pages. Typically, web material is archived using many containers. The primary goal of our work is to develop techniques to allocate web pages to containers such that each container has as closely related web pages as possible, thereby minimizing the chances of accessing many different containers when a user browses through the archived web material. When web contents are archived in the form of multiple containers, we can view these containers as a coarsened web graph (or container graph) where the original nodes within the same container are collapsed together to form a super node, and only edges between different containers survive with assigned weights as will be explained next.

In the container graph, $G_c=(V_c,E_c)$, we define the cost of the edge-cut, EC, as follows:

$$EC = \sum_{e \in E_c} w_e$$, where w_e is the weight of edge e.

In order to accomplish our goal, we analyze the link structure within the web material to be archived to find, for each edge, a good estimate of the probability that the edge will be taken. Using this estimate as the edge weight, we partition the web graph in such a way as to minimize EC. The following two subsections discuss our link analysis and the partitioning technique used to minimize EC.

3.1. Edge Weights

Edge weights should represent the relative likelihood of an edge being taken during a browsing session. We will follow a line of attack similar to the one used for computing PageRanks. We

start with some simple observations. If a vertex has only one outgoing edge, this edge will be more likely taken than an edge from another vertex with many out-links, and thus should be weighed more heavily. A possible simple solution is to assign edge weights depending on the number of out-links of the source vertex. For instance, if the source vertex of edge *e* has *k* outgoing edges, the weight of $\frac{1}{k}$ is given to edge *e*.

When a personalized vector is not in use, the PageRank algorithm also uses the same method in assigning edge weights. In this case, the only deciding factor to the edge weight is the number of the outgoing edges from the source vertex, and thus the edge weight only represents the local probability of the edge being taken, *once the source vertex is visited*. In other words, the edge weight is only locally meaningful, and thus it is not possible to say that an edge is more likely to be taken than the other if they belong to different vertices.

For our method, the probability of each vertex being visited is computed first using the PageRank algorithm. The PageRank value (or steady-state probability) of each vertex is then divided by the number of outgoing edges from the vertex. We call this quotient *EdgeRank (ER)* and assign the same EdgeRank value as the weight to every edge coming out from the same vertex.

$$ER(e) = \frac{PR(v)}{\text{outdegree}(v)}$$, where vertex *v* is the source vertex of edge *e*.

Note that, since $\sum PR(v) = 1$, $\sum ER(e) = 1$ too.

Now that we have an edge-weighted graph representing our web contents, the allocation of web pages to containers is performed using a graph partitioning algorithm.

3.2. Graph Partitioning

As discussed in Section 2, there are a number of existing min-cut graph partitioning heuristics that seem to work well in practice. Although their primary partitioning criterion is to minimize the cost of the edge-cut, they differ from one another in input, output, and partitioning parameters. For example, some algorithms support size-constrained partitioning while others do not. Also, not all algorithms support weighted vertices and edges. Before proceeding let's define our graph partitioning problem more formally.

Web Graph Partitioning Problem: Given a directed web graph $G : \{V, E\}$ with weighted nodes (weight of a node is the size of the corresponding page) and weighted edges, determine a partition $V = P_1 \cup P_2 \cup P_3 \cup ... \cup P_n$ such that,

1. The sum of the weights of the edges that connect any two different partitions is minimized.

2. For all *i*'s, $|P_i| \leq K$ for some fixed *K*, where $| P_i |$ is the sum of the weights of the vertices in the partition and *K* is an upper bound on the size of a container.

The first condition is shared by almost all partitioning algorithms (some require non-weighted edges), while the second condition, which is the size constraint imposed to every partition, is supported only by a few partitioning algorithms (such as [19, 21, 24]), sometimes with a slight modification.

In this work, we adopt the multilevel graph partitioning algorithm to solve our problem. The primary reason is that it supports the constraints on the partition size; moreover, the method is

fast, which is important in our case considering the typically large sizes of web graphs. In particular, we adapt a partitioning technique suggested by Karypis and Kumar [21] as follows.

Their scheme first computes a maximal matching using a randomized algorithm, and coarsens the graph by collapsing the matched vertices together. This coarsening step is repeated until a desired size of the coarsened graph is achieved. Once the graph is coarsened, the minimum edge-cut bisection is computed using some of existing algorithms such as spectral bisection [8, 33], geometric bisection [28] or combinatorial methods [13, 14, 24]. The partitioned graph is then refined and uncoarsened. The improved Kerninghan-Lin algorithm that was developed by Karypis and Kumar is applied to this uncoarsening-with-refinement phase.

In particular, we use Metis [22], a partitioning tool that implements the Karypis-Kumar scheme. Although Metis does not explicitly support the partition size constraints – our second condition, it does support vertex-weight-based size balancing among partitions, making the size of all partitions similar. Therefore, based on the sum of all the vertex weights of the web graph, we pre-compute the necessary number of partitions before running the partitioning tool, so that the resulting partitions will meet the second condition.

4. Experimental Evaluation of our Scheme

In order to examine the performance of our algorithm in terms of the number of containers accessed during a typical browsing session, we consider two datasets. The first is the web graph of the University of Maryland Institute for Advanced Computer Studies (UMIACS) web site, located at http://umiacs.umd.edu domain, which we call the UMIACS web graph. We crawled every Web page within five-hop distance (or depth) under this domain, and constructed the web graph corresponding to this crawling. The second dataset is the Stanford web graph which was generated from a crawl of the stanford.edu domain created in September 2002 by the Stanford WebBase project [20], and is widely used by the web graph analysis community. Unlike the first dataset, the Stanford web graph has neither the size information of vertices, nor the actual URLs with which we might have been able to obtain estimates of the web pages (which undoubtedly have changed since then). Consequently, we randomly assign vertex sizes using two Gaussian distributions – one for html files, the other for non-html files. Their parameters are based on the findings from a Web statistics study [25]. In particular, we assumed there are about 18% html objects by total file size, and the average html file size is 605 KB. This size modeling is not intended to mimic the actual Web object sizes in the Stanford web page. Rather, we intend to assign some reasonable sizes to run our experiments. Note that the quality of our method does not depend on the accuracy of the vertex sizes.

Table 1 describes these two datasets.

Table 1. The Two Datasets Used for Evaluating our Method

Datasets	# Vertices	# Edges	Total Vertex Weight
UMIACS Web Graph	4579	9732	2.49GB
Stanford Web Graph	281903	2312497	215.82GB

In our experiments, we allocate pages to containers (or WARC files) in three different ways.

1. CONV: Pages are allocated to containers as they are fetched during the crawling process. Once a container is full, we use a new container (Figure 1).
2. GP: The graph partitioning technique is applied so as to minimize the number of edges connecting any two partitions. All the pages belonging to a partition are allocated to a single container (Line 3 in Figure 2 is omitted).
3. ER+GP: The EdgeRank technique is used to assign weights to edges (Line 3 in Figure 2), and the graph is partitioned using a minimum-weight partitioning algorithm Again, containers are constructed based on the resulting partitions. In each case, the damping factor, d = 0.85, is used in EdgeRank.

```
     Input
     Seed URLs : {url1, url2, … }
     MAX_CONTAINER_SIZE

     Procedure
 1:  Enqueue(Q, Seed URLs)
 2:  i←1
 3:  visited[] ← FALSE
 4:  Ci ← new Container()
 5:  while (Q is non-empty)
 6:        u ← Dequeue(Q)
 7:        Fetch(u);
 8:        visited[u] ←TRUE
 9:        if (Size(Ci) + Size(u) > MAX_CONTAINER_SIZE)
10:              i = i + 1
11:              Ci = new Container()
12:        Ci = Ci  ∪ u
13:        for each v ∈ Adj[u]
14:              if (visited[u] = FALSE)
15:                    Enqueue(Q, v)
```

Figure 1. Conventional Allocation of Pages to Containers

```
     Input
     Seed URLs : {url1, url2, … }
     MAX_CONTAINER_SIZE

     Procedure
 1:  G ← BuildWebGraph(Seed URLs)  /* Using BFS algorithm */
 2:  n ← GetNumberOfContainers(G, MAX_CONTAINER_SIZE)
 3:  G ← EdgeRank(G) /* Optional */
 4:  {UL1,UL2,…,ULn} ← PartitionGraph(G, n)
 5:  for ( 1 ≤ i ≤ n)
 6:        Ci ← new Container()
 7:              for (v ∈ ULn)
 8:                    fetch(v)
 9:                    Ci = Ci ∪ v
```

Figure 2. Container Construction Based on Graph Partitioning with or without EdgeRank (Line 3)

Figure 1 shows a typical BFS algorithm where a visited node is stored in the current container as long as the size of the resulting container does not exceed the predefined value (MAX_CONTAINER_SIZE) (Line 9~12). A new container is created if necessary.

In the algorithm shown in Figure 2, a web graph is first built (Line 1) using a BFS-based crawling algorithm similar to the one in Figure 1, followed optionally by computing EdgeRank (Line 3) in order to obtain edge weights in the graph. This graph is then partitioned into the pre-calculated (Line 2) number (n) of partitions. This number depends on the total sum of vertex weights (page sizes) in the graph, as well as the predefined maximum container size (MAX_CONTAINER_SIZE). Once partitioned, the URLs in each partition are re-visited and packaged in the n containers (Line 5~9). In practice, depending on the resource availability, the Web objects downloaded from the previous crawl (Line 1) can be stored and reused in the packaging process.

In our simulation, to be discussed in Section 4.2, the UMIACS dataset was partitioned into 25 partitions and the Stanford dataset was partitioned into 2200 partitions, resulting in the size of each partition being between 100MB and 200MB.

4.1. Edge-Cut

In order to evaluate the graph partitioning performance, we measure the edge-cut obtained from the graph partitioning scheme, and compare it to the conventional breadth-first-search (BFS) partitioning. We defined the cost of an edge-cut earlier as the sum of the weights of all the external edges between partitions. However, as we performed the experiments on the two separate datasets with different numbers of nodes, edges and partitions, we scaled down the cost of the edge-cut to a web graph with the total edge weight of 100, as follows:

$$EC_{scaled} = \frac{EC \times 100}{|E|}$$, where $|E|$ is the total edge weights in the web graph.

We begin by considering the case where the web graph has no edge weight (or equal edge weight). We observe that the edge-cuts generated by the conventional method were about 70~80 for both datasets while those generated by the graph partitioning scheme are 12 and 47 for the UMIACS and Stanford datasets respectively. Using edge weights based on the PageRank technique, the graph partitioning approach similarly reduces the costs of the edge-cuts relative to the conventional approach as illustrated in Table 2.

Table 2. Edge-Cut Results

EC_{scaled}	Unweighted Edges		Weighted Edges	
	CONV	GP	ER+CONV	ER+GP
UMIACS Web Graph	73.87	12.38	62.36	36.03
Stanford Web Graph	80.50	47.33	63.56	32.20

4.2. Simulation

Although the edge-cut figures show favorable results when the partitioning technique is employed, we additionally ran simulations to further see how much the partitioning and the EdgeRank will in fact reduce the number of containers necessary for a random user to browse through the

archived Web material. In these simulations, we set a virtual user who randomly walks through links, and counted the number of containers that the user had to access.

Table 3 shows the parameter-value pairs used in our simulations.

Table 3. Simulation Parameters

Parameter	Value
Number of Hops	10
Probability of Going Back	30%
Outdegree of Starting Vertex	> 5
Policy At Dangling Vertex	Go back

Each random walk consists of ten random hops, and at each random hop, each outgoing link is given an equal probability of being taken. Also, we assume that the BACK button on a browser is pressed with 30% probability. We base this choice on a recent browser usage research [23] which shows that hyperlinks are taken 41.7% of time, followed by other navigation (23.6%) and the back button (18.9%). Since, in our simulation, we only consider hyperlinks and the back button, we assume that the back button is pressed about 30% ($\approx \frac{18.9}{41.7+18.9}$) of the time. Once the random walk reaches a vertex with no outgoing link (or a dangling), the random walk goes back to the previous vertex, if any, as if the user presses the BACK button. In order to avoid the situation where there are no more vertices left to visit soon after the start of the simulation, we insist that the randomly selected starting vertex has an outdegree of five or larger.

In the simulations, we ran the random walk 1000 times over each dataset, and monitored both the number of inter-container hops and the number of distinct containers needed for each random walk. Inter-container hops occur whenever a different container needs to be accessed. For example, if a random walk switches back and forth between two containers, A and B, ten times, the number of inter-container hops will be ten, while the total number of distinct containers is only two. In a system with no caching policy or a limited memory, the inter-container hops will serve as a more useful metric because, even if a user requests a previously retrieved container, the system will always need to retrieve it from storage. However, if a system can cache enough containers, the total number of distinct containers will make more sense in assessing the system's performance. Figures 3 and 4 show the histograms of the number of inter-container hops and distinct containers accessed for the UMIACS web graph, respectively, while Figures 5 and 6 show the corresponding histograms for the Stanford web graph.

It can be observed that when the graph partitioning scheme is used, many random walks only need a single container (thus, zero inter-container hops). Figure 7 depicts the average number of inter-container hops during the random walks over the two web graphs. From the figure, it can be seen that the GP and ER+GP schemes reduced the average number of inter-container hops from five to one for the UMIACS web graph. For the Stanford web graph, the GP scheme reduced the number from seven to five, while ER+GP further reduced the number down to four. The average number of containers needed is shown in Figure 8. Although the improvements are not as dramatic as the number of inter-container hops, compared to the CONV scheme, the GP scheme required about 28% and 11% less number of distinct containers for the UMIACS and Stanford web graph, respectively. The ER+GP scheme further reduced the numbers 9% and 17% less than those from the GP scheme.

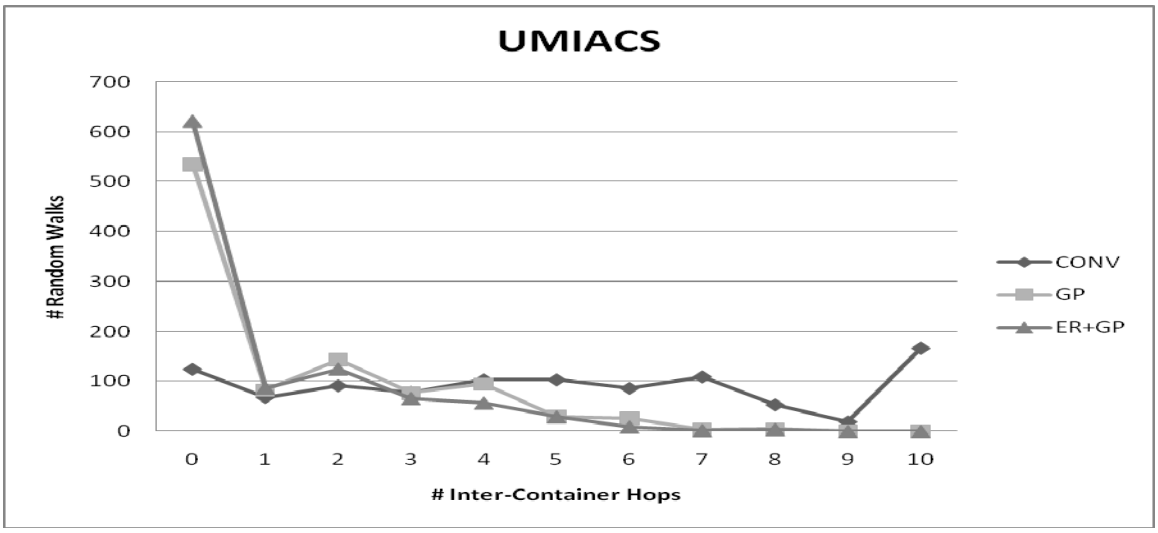

Figure 3. Histogram of Number of Inter-Container Hops for UMIACS Web Graph

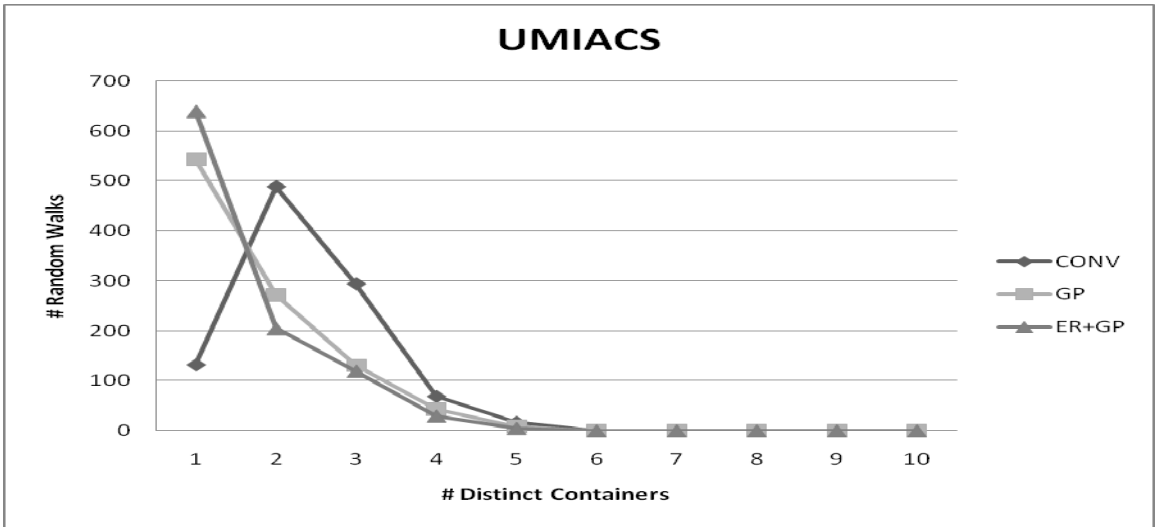

Figure 4. Histogram of Number of Distinct Containers Accessed for UMIACS Web Graph

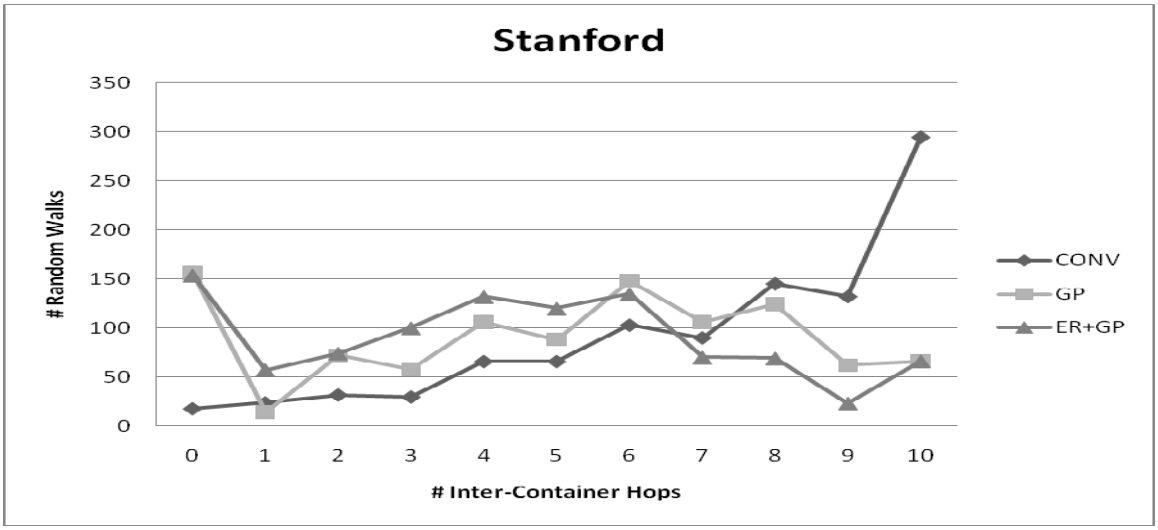

Figure 5. Histogram of Number of Inter-Container Hops for Stanford Web Graph

Figure 6. Histogram of Number of Distinct Containers Accessed for Stanford Web Graph

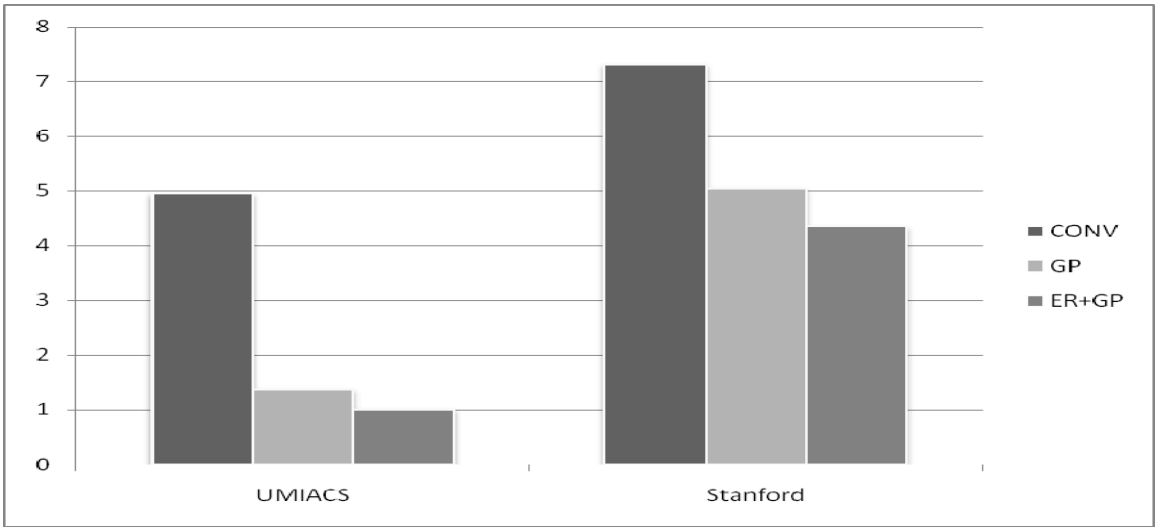

Figure 7. Average Number of Inter-Container Hops

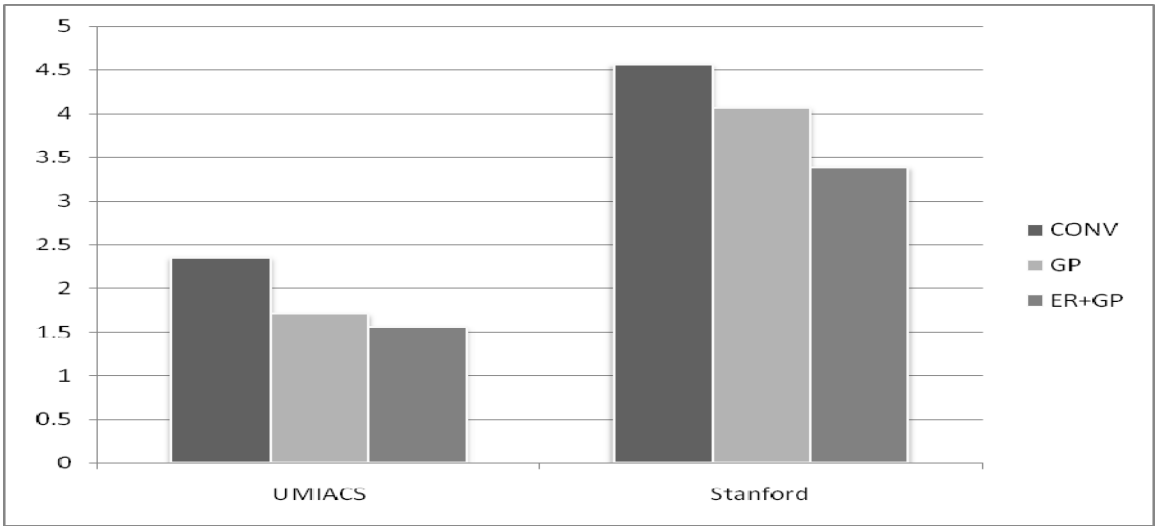

Figure 8. Average Number of Distinct Containers

5. Conclusion

In this paper, we have shown that a graph partitioning scheme for organizing archive containers significantly reduces the number of containers that need to be accessed when a user browses through the archived Web material. Also shown was a PageRank-derived technique, called EdgeRank, which can improve this number even further. The overhead required by this technique is relatively small. For instance, on our 2Ghz Intel Core 2 Duo processor, we could fully partition and compute EdgeRank of a large graph (the Stanford web graph that contains about 300,000 vertices, and 2.3 million edges) within minutes.

6. References

[1] HTTrack `http://www.webcitation.org/5SCSBqOXe`

[2] *The Internet Archive - The Wayback Machine* `http://www.webcitation.org/5SCSL2r8e`

[3] NutchWAX `http://www.webcitation.org/5SCS7U2LE`

[4] *WARC, Web ARChive file format* `http://www.webcitation.org/5RPhvw0Wa`

[5] WAXToolBar `http://www.webcitation.org/5SCSFHkK3`

[6] WERA `http://www.webcitation.org/5SCSHC1w7`

[7] *Wikipedia Statistics* `http://www.webcitation.org/5QwnKX6Gp`

[8] S. T. Barnard and H. D. Simon, *A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems*, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, Norfolk, Virginia, USA, 1993, pp. 711-718.

[9] S. Brin and L. Page, *The anatomy of a large-scale hypertextual Web search engine*, in P. H. Enslow and A. Ellis, eds., *Proceedings of the Seventh International Conference on World Wide Web 7*, Elsevier Science Publishers B. V., Brisbane, Australia, 1998, pp. 107-117.

[10] T. N. Bui and C. Jones, *A heuristic for reducing fill in sparse matrix factorization*, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, Norfolk, Virginia, USA, 1993, pp. 445–452.

[11] C.-K. Cheng and Y.-C. A. Wei, *An improved two-way partitioning algorithm with stable performance*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 10 (1991), pp. 1502-1511.

[12] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, *Proceedings of the 19th Conference on Design Automation*, IEEE Press, 1982, pp. 175-181.

[13] A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345-363.

[14] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite*, Prentice Hall Professional Technical Reference, 1981.

[15] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[16] L. Hagen and A. Kahng, *Fast spectral methods for ratid cut partitioning and clustering*, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Santa Clara, CA, USA, 1991, pp. 10-13.

[17] L. Hagen and A. B. Kahng, *A new approach to effective circuit clustering*, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Santa Clara, CA, USA, 1992, pp. 422-427.

[18] M. T. Heath and P. Raghavan, *A Cartesian parallel nested dissection algorithm*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 235-253.
[19] B. Hendrickson and R. Leland, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM Journal on Scientific Computing, 16 (1995), pp. 452-469.
[20] J. Hirai, S. Raghavan, H. Garcia-Molina and A. Paepcke, *WebBase: A repository of Web pages*, Computer Networks, 33 (2000), pp. 277-293.
[21] G. Karypis and V. Kamar, *Multilevel k-way partitioning scheme for irregular graphs*, Journal of Parallel and Distributed Computing, 48 (1998), pp. 96-129.
[22] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 5.0pre2*, Minneapolis, 2007.
[23] M. Kellar, C. Watters and M. Shepherd, *The impact of task on the usage of web browser navigation mechanisms*, *GI '06: Proceedings of Graphics Interface 2006*, Canadian Information Processing Society, Quebec, Canada, 2006, pp. 235-242.
[24] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical journal, 49 (1970), pp. 291-307.
[25] P. Lyman and H. R. Varian, *How Much Information* `http://www.webcitation.org/5SCSQh9n9`
[26] J. Masanès, *Web Archiving: Issues and Methods*, *Web Archiving*, Springer, Berlin, 2006, pp. 1-53.
[27] G. L. Miller, S.-H. Teng, W. Thurston and S. A. Vavasis, *Automatic Mesh Partitioning*, *IMA Volumes in Mathematics and its Applications:*, Springer-Verlag, New York, NY, USA, 1993, pp. 57-84.
[28] G. L. Miller, S.-H. Teng and S. A. Vavasis, *A unified geometric approach to graph separators*, *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, San Juan, Puerto Rico, 1991, pp. 538-547.
[29] G. Mohr, M. Kimpton, M. Stack and I. Ranitovic, *Introduction to Heritrix, an archival quality web crawler*, *4th International Web Archiving Workshop*, Bath, UK, 2004.
[30] L. Page, S. Brin, R. Motwani and T. Winograd, *The PageRank citation ranking: Bringing order to the Web*, 1998.
[31] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochasitic Processes*, McGraw-Hill, New York, 2002.
[32] R. Ponnusamy, N. Mansour, A. Choudhary and G. C. Fox, *Graph contraction and physical optimization methods: a quality-cost tradeoff for mapping data on parallel computers*, *International Conference of Supercomputing*, Tokyo, Japan, 1993.
[33] A. Pothen, H. D. Simon and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 430-452.