Knowledge and Information Systems (2002) 4: 151–167

Knowledge and Information Systems

Correlation-Based Web Document Clustering for Adaptive Web Interface Design[1]

Zhong Su[1], Qiang Yang[2], Hongjiang Zhang[3], Xiaowei Xu[4], Yu-Hen Hu[5] and Shaoping Ma[1]
[1]State Key Lab of Intelligent Tech. and Systems, Tsinghua University, Beijing, China
[2]School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
[3]Microsoft Research China, Beijing, China
[4]Siemens AG, Information and Communications Corporate Technology, Munich, Germany
[5]Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, Wisconsin, USA

Abstract. A great challenge for web site designers is how to ensure users' easy access to important web pages efficiently. In this paper we present a clustering-based approach to address this problem. Our approach to this challenge is to perform efficient and effective correlation analysis based on web logs and construct clusters of web pages to reflect the co-visit behavior of web site users. We present a novel approach for adapting previous clustering algorithms that are designed for databases in the problem domain of web page clustering, and show that our new methods can generate high-quality clusters for very large web logs when previous methods fail. Based on the high-quality clustering results, we then apply the data-mined clustering knowledge to the problem of adapting web interfaces to improve users' performance. We develop an automatic method for web interface adaptation: by introducing index pages that minimize overall user browsing costs. The index pages are aimed at providing short cuts for users to ensure that users get to their objective web pages fast, and we solve a previously open problem of how to determine an optimal number of index pages. We empirically show that our approach performs better than many of the previous algorithms based on experiments on several realistic web log files.

Keywords: Adaptive web user interfaces; Clustering; Data mining; Web log mining

[1] This work was performed in Microsoft Research China.
Received 25 Nov 2000
Revised 15 Mar 2001
Accepted 14 May 2001

1. Introduction

A serious problem on the World Wide Web is the increasing inability for a user to find the right information efficiently. With the explosive growth of information content on the web, users visiting a web site often found it necessary to go through many levels of web pages to get to the right destination. A statically created web site is likely to have many levels of indirections, creating layers of barriers that increase users' efforts in browsing and searching. Ideally, a good web site should cater to the behavior of different visitors with distinct goals. Moreover, these goals may be a function of time themselves.

Our approach to the problem is to analyze web server logs to discover correlation knowledge among web pages. Such knowledge will give clues to clusters of web pages that are co-visited often by millions of users. An important motivation of using web logs to discover correlation knowledge is that web sites often only encode web designers' subjective knowledge. Users' behavior and preferences are often ignored. To address the problem, we adopt an *Analyze-and-Adapt* methodology, whereby we first analyze very large web server logs to uncover correlation knowledge among web pages. We then use the knowledge to guide adaptation of the web pages to improve the ease-of-use property of a web site.

More specifically, we first apply clustering techniques to web server logs to obtain highly correlated web pages. In doing so, we have found that the density-based clustering method (Ester et al., 1996), a well-known approach to clustering in the database field, tends to merge many clusters together. To solve this problem, we introduce a recursive density-based clustering algorithm that repeatedly abstracts the important web pages at successive higher levels of abstraction. The clusters built at higher levels are then used to guide the clustering process at lower levels. We have found that this novel approach maintains the efficiency of density-based clustering while producing higher quality results.

With the high-quality clusters generated, we then adapt the web pages to provide users with short cuts to their desired pages. Previous work on this includes the well-known PageGather algorithm by Etzioni and Perkowitz, which generates index pages that contain table-of-content information for web sites. A problem with their approach is that human designers generate the final index pages in an ad hoc way. No indication was given as to how many index pages is an optimal number. Therefore, users may be required to read a large number of index pages before getting to the pages that they want. This can create more difficulty than before. In response, we introduce a cost model whereby we can quantify the users' efforts in reading index pages. More importantly, we can use the cost model to find the optimal number of index pages, as well as the number of hyperlinks per index page, so as to minimize web site users' overall browsing effort. With the high-quality clusters and the cost models, we show empirically that users' overall effort can be guaranteed to decrease.

The paper is organized as follows. In Section 2, we review previous work in clustering and adaptive web interfaces. In Section 3, we focus on our new algorithm RDBC for clustering web pages based on statistical analysis of web logs. In Section 4, we define a cost-model-based optimization criterion for assigning index pages. In Section 5, we experimentally evaluate variants of RDBC on three realistic web server logs, and compare the performance of RDBC to previous density-based methods for clustering (Ester et al., 1996). We conclude with a discussion of future work and a summary of our contributions.

2. Background

Two particular areas are related to our research. One is the field of adaptive web interfaces, where the goal is to make a web site adapt to user behavior so as to ensure efficient access to users' objective pages. The second field is clustering, where there has been a great deal of research done in the database and data-mining field. In this section, we review both of these areas.

2.1. Previous Work on Adaptive Web Interfaces

The problem we consider in this paper falls in the class of server side customization and transformation – operations that convert a web site, on the server side, into one that is more convenient for users to visit and find their objectives. Several researchers have studied the problem of creating web interfaces that can adapt to user behaviors based on web logs. Examples include *path prediction* algorithms that guess where the user wants to go next in a browsing session so that the server can either pre-send documents or short-cut browsing paths. For instance, WebWatcher (Armstrong et al., 1995) learns to predict what links users will follow on a particular page as a function of their specified interests. A link that WebWatcher believes a particular user is likely to follow will be highlighted graphically and duplicated at the top of the page when it is presented. Another example is the AVANTI Project, which attempts to predict the user's eventual goals. AVANTI (Fink et al., 1996) will prominently present links leading directly to pages it thinks a user will want to see. Su et al. (2000) also present a web prediction algorithm. Our algorithm is an *n*-gram-based model, which utilizes path profiles of users from very large web logs to predict the users' future requests. By extending the previous work on point-based predictions, it shows that it is possible to gain a great deal in prediction precision by sacrificing the model coverage.

Yet another branch of work is that of *collaborative filtering*, in which users rate objects (e.g., web pages or movies) based on how much they like them. Users that tend to give similar ratings to similar objects are presumed to have similar tastes. When a user seeks recommendations of new objects, the site suggests those objects that were highly rated by other users with similar tastes. The site recommends objects based solely on other users' ratings or accesses, ignoring the content of the objects themselves.

The PageGather system (Perkowitz and Etzioni, 1999) provides users with short cuts, which takes as input a web server access log, where the log records the pages visited by a user at the site. Based on the visiting statistics, the system provides links on each page to visitors' eventual goals, skipping the in-between pages. An *index page* is a page consisting of links to a set of pages that cover a particular topic (e.g., electric guitars). Given this terminology Perkowitz and Etzioni define the *index page synthesis problem*: given a web site and a visitor access log, create new index pages containing collections of links to related but currently unlinked pages. An access log is a document containing one entry for each page requested of the web server. Each request lists at least the origin (IP address) of the request, the URL requested, and the time of the request. *Related but unlinked* pages are pages that share a common topic but are not currently linked at the site; two pages are considered linked if there exists a link from one to the other or if there exists a page that links to both of them. In

their PageGather algorithm, Perkowitz and Etzioni presented a clustering-based method for generating the *contents* of the new web page from server access logs.

Given an access log, an important task is to find collections of pages that tend to co-occur in visits. Clustering (Voorhees, 1986; Willet, 1988; Rasmussen, 1992) is a natural technique to consider for this task. In clustering, documents are represented in an n-dimensional space (for example, as word vectors). Roughly, a cluster is a collection of documents close to each other and relatively distant from other clusters. Standard clustering algorithms partition the documents into a set of mutually exclusive clusters.

The *PageGather algorithm* (Perkowitz and Etzioni, 1999)uses cluster mining to find collections of related pages at a web site. In essence, PageGather takes a web server access log as input and maps it into a form ready for clustering; it then applies cluster mining to the data and produces candidate index page contents as output. The algorithm has six basic steps:

1. Process the access log into visits.
2. Compute the co-occurrence frequencies between pages and create a similarity matrix.
3. Create the graph corresponding to the matrix, and find maximal cliques (or connected components) in the graph.
4. Rank the clusters found, and choose which to output.
5. For each cluster, create a web page consisting of links to the documents in the cluster.
6. Present clusters to the Webmaster for evaluation.

Let N be the number of web pages at the site. This algorithm is thus $O(N^2)$ time, quadratic in the original number of web pages. We note that because of this high complexity the algorithm is not suitable for processing large data sets that are typical of today's web access patterns. For example, every day MSN collects millions of visits. Data sets of this scale must be processed with very efficient disk-based algorithms. Thus, one of our intentions is to explore more efficient clustering algorithms for synthesizing index pages.

Another drawback of the PageGather algorithm is that it relies on the human webmasters to determine the appropriateness of the generated index pages in a final check. This will likely create a bottleneck for the workflow, especially for sites that have many web pages to be indexed. A particularly important problem is the question of *how many index pages* to create, and *how many links* to include in each index page. We answer this question in our paper.

2.2. Previous Work on Clustering

PageGather addresses the problem of finding sets of items based on their access patterns. It falls in the category of clustering algorithms. However, the quality of the index pages is mostly decided by the effectiveness and efficiency of the clustering algorithm. In addressing the efficiency issue, we observe that there is a great deal of work done previously in clustering. Typical of the clustering work are the K-means clustering (Rocchio, 1966) and hierarchical agglomerative clustering (HAC) (Voorhees, 1986). K-means requires that the user provide the number K of clusters as initial input, and in our situation we cannot be sure of the number of index pages to create beforehand. An even more serious

drawback of K-means algorithm is that it is able to form clusters that have a spherical shape, and has difficulties building clusters of arbitrary shapes (Xu et al, 1998); therefore, K-means might be too restrictive for this purpose. On the other hand, HAC algorithms are not directly suitable for large data sets. A modification is the frequent set algorithms that are designed to find sets of similar items in large collections (Agrawal et al., 1993; Savasere et al., 1995; Agrawal et al., 1996; Toivonen, 1996). Perkowitz and Etzioni found that the PageGather algorithm is even more efficient than the Apriori algorithms.

In looking for an efficient clustering algorithm with clustering output for arbitrary shaped clusters, we decided to extend the density-based clustering algorithm. The density-based method is very efficient to execute and does not require the user to pre-specify the number of clusters. Density-based methods are based on the idea that it is likely that in a space of objects dense objects should be grouped together into one cluster. Thus, a cluster is a region that has a higher density of points than its surrounding region. For any points in a space, where a point corresponds to a web page, the more web pages that co-occur with it, the higher its density is. The density-based method originates from a method called DBSCAN (Ester et al., 1998; Xu et al, 1998) for data mining. The main feature of the algorithm is that it relies on the density of data, so that it can discover clusters of shapes that are unions of spheres in order to group close objects together.

More specifically, DBSCAN accepts a radius value ϵ-based on a user-defined distance measure, and a value *MinPts* for the number of *minimal points* that should occur around a dense object; the latter is used to determine, out of many points in a space, which region is considered dense. DBSCAN then iteratively computes the density of points in an n-dimensional space, and groups the points into clusters. Next, we provide more precise definitions for the definitions of clusters.

First we define the ϵ-neighborhood of a point as the set of points that are within ϵ-distance from the point.

Definition 2.1. The *ϵ-neighborhood* of a point p, denoted by $N_\epsilon(p)$, is defined by $N_\epsilon(p) = \{q \in D | dist(p,q) \leqslant \epsilon\}$. Given a value for minimal points MinPts, a point q within the ϵ-neighborhood of a point p is said to be directly density-reachable from q (also refer to Ester et al. (1996)).

Definition 2.2. A point p is *directly density-reachable* from a point q with respect to ϵ and *MinPts* if (1) $p \in N_\epsilon(q)$ and (2) $|N_\epsilon(q)| \geqslant MinPts$(core-point condition). Armed with the notion of directly density-reachable, we can define density-reachable by transitivity (also refer to Ester et al. (1996)).

Definition 2.3. A point p is *density-reachable* from a point q with respect to ϵ and *MinPts* if there is a chain of points $p_1, \ldots, p_n, p_1 = q, p_n = p$, such that p_{i+1} is directly density-reachable from p_i (also refer to Ester et al. (1996)).

Definition 2.4. A point p is *density-connected* to a point q with respect to ϵ and *MinPts* if there is a point o such that both p and q are density-reachable from o with respect to ϵ and *MinPts* (also refer to Ester et al. (1996)).

Definition 2.5. Let D be a database of points with a distance definition upon it. A *cluster* C with respect to ϵ and *MinPts* is a non-empty subset of D satisfying the following conditions: (1) $\vee p, q$: if $p \in C$ and q is density-reachable from p with respect to ϵ and *MinPts*, then $q \in C$(Maximality); (2) $\vee p, q \in C$: p is density-connected to q with respect to ϵ and *MinPts*(Connectivity) (also refer to Ester et al. (1996)).

Based on the above definitions, Ester et al. (1996) gave a density-based cluster definition. Given fixed ϵ and *MinPts* values, the DBSCAN algorithm looks for a core point to start. It recursively expands a cluster by definition 2.5. To support disk-based processing, all the points that belong to a cluster are labeled as such and not considered again in future computation. Therefore this algorithm incurs $N*log(N)$ in time complexity, where N is number of points (here a point corresponds to a unique web page). However this algorithm with fixed ϵ and *MinPts* as inputs, when applied to web page clustering, can give rather skewed results because fixed ϵ and *MinPts* values may not be the best choice at all regions in the space. As we will show later, it often puts all the web pages in one giant cluster. Our RDBC algorithm is aimed at solving this problem by varying ϵ and *MinPts* where it is necessary. We will demonstrate that it is in fact more reasonable to induce a collection of clusters where each cluster can have different ϵ and *MinPts* values.

Algorithm DBSCAN($DB, \epsilon, MinPts$**)**
  for each $o \in DB$ **do**
    if o is not yet assigned to a cluster **then**
      if o is a core-object **then**
        collect all objects density-reachable from o according to ϵ and *MinPts*;
        assign them to a new cluster;

3. The RDBC Algorithm

A problem with DBSCAN is its tendency to merge many slightly connected clusters together. To address this problem, our key idea is to find important core points and build clusters on these core points; the rest of the core points are then absorbed by the clusters depending on their distance to the clusters.

In this section, we introduce the RDBC algorithm for clustering. RDBC is an improvement of DBSCAN for the web page clustering applications. In RDBC, a call is made to DBSCAN with different distance thresholds ϵ and density threshold *MinPts*. It recursively selects core points to build the next level of abstraction, based on the density measures. The resultant core points for clusters are returned when the number of clusters becomes appropriate, a notion we define below. The key difference between RDBC and DBSCAN is that in RDBC the identification of core points is performed separately from that of clustering. We call this an it abstraction because these core points can be regarded as clustering centers that are representative of the data points. For this purpose, different values of ϵ and *MinPts* are used in RDBC to identify this core point set, which we call *Cset*. Only after an appropriate *Cset* has been determined are the core points clustered and the remaining data points assigned to these clusters according to their proximity to a particular cluster.

The algorithm is summarized below. The RDBC algorithm is called with initial values for ϵ and *MinPts* that are user given, but is insensitive to such initial values. It returns a set of clusters as output.

Algorithm RDBC($\epsilon, MinPts, WebPageSet$)
  Use ϵ and *MinPts* to get the core-point set *Cset*;
  if $size(CSet) > |WebPageSet|/2$ **then**
    { // Stopping criterion is met
      $DBSCAN(WebPageSet, e, MinPts)$;
      Return clusters found by DBSCAN;
    } //end of then

else
{ // Continue to abstract core points
$\epsilon = \epsilon/2; MinPts = MinPts/4;$
$Clusters = RDBC(\epsilon, MinPts, CSet);$
Collect all other points in (WebPageSet - CSet) into Clusters, using new distance threshold ϵ^2;
Return Clusters;
} //end of else

Intuitively, the algorithm goes into a cycle in which the core points are taken as the points in a space, and clustering is done on that core with smaller radius around a core point. The stopping rule of the recursive process could be changed according to the specific data distribution. In our work, we set the value to $1/2$ according to a large number of experiments on web logs. It means that the process stops when nearly half the remaining points are core points. The algorithm will then begin a gathering process to gather the rest of the points around the core points found into clusters. This is done with a larger radius value ϵ^2. Intuitively, this process can avoid connecting too many clusters via 'bridges'.

The time complexity of DBSCAN is $O(N * \log N)$, where N is the number of distinct web pages. In our RDBC algorithm, the DBSCAN algorithm is run only once. The rest of the time is spent on collecting the surrounding points into a closest cluster. Therefore, the time complexity of our algorithm is $O(N * \log N)$ for N, the number of web pages.

Compared to traditional clustering algorithms such as K-means and the Scatter/Gather algorithm, our proposed RDBC algorithm has several potential advantages: (a) RBDC does not require a pre-specified number of clusters. The clustering distance threshold ϵ can be initialized to be an arbitrarily large number and eventually it will be adjusted by the algorithm. Similarly, the *MinPts* parameters are computed during the execution of the algorithm. (b) Because the algorithm uses density-based connectivity criteria, it may discover clusters of arbitrary shape, similar to the DBSCAN algorithm. (c) In addition, it has a log-linear time in complexity and hence is very efficient in processing large-scale real-world data.

4. Preprocessing for Web Log Clustering

We now consider the practical steps that are needed to preprocess the web logs in order to execute the RDBC algorithm for clustering. There are two broad steps, as described below. An example log from a NASA web site is given below, where each entry corresponds to a single request to the server and includes originating machine, time, and URL requested, and size of the web page requested.

uplherc.upl.com - - [01/Aug/1995:00:08:52 -0400] "GET /shuttle/resources/orbiters/endeavour-logo.gif HTTP/1.0" 200 5052

pm9.j51.com - - [01/Aug/1995:00:08:52 -0400] "GET /images/WORLD-logosmall.gif HTTP/1.0" 200 669

139.230.35.135 - - [01/Aug/1995:00:08:52 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786

uplherc.upl.com - - [01/Aug/1995:00:08:52 -0400] "GET /shuttle/resources/orbiters/endeavour-logo.html HTTP/1.0" 200 5052

pm9.j51.com - - [01/Aug/1995:00:08:52 -0400] "GET /images/WORLD-logosmall.html HTTP/1.0" 200 669
139.230.35.135 - - [01/Aug/1995:00:08:52 -0400] "GET /images/NASA-logosmall.html HTTP/1.0" 200 786

Step 1: Pre-process access log into sessions.

We remove requests made to access image files (.gif, .jpg) in the log. Since most of them are accompanying figures to a specific web page, the users do not request these image files explicitly. We then extract user sessions from the log data. A natural boundary for sessions is when users make unusually long pauses between browsing activities. These can be detected by observing the density of activities as a function of time. We have found that it is often the case that for a given web log one can obtain a threshold value on the time interval between two adjacent page visits. If the time interval between the visits is greater than a time threshold T, then these visits are considered to belong to two different sessions. For example, we have observed that it is safe to set T at 2 hours for NASA data that we present later, and 24 hours for MSN data.

Step 2: Compute the co-occurrence frequencies between pages within a window size W (W is given as input), and create a distance matrix.

In this step we determine the size of a moving window within which URL requests will be regarded as co-occurrent. Note that here we implicitly define a temporal locality between successive web page accesses. Since we are not using the content of each web page as feature vectors for clustering web pages, temporal proximity is used instead to indicate two web pages are close in the data space. While this distance measure is not always satisfactory, it is the best information we can extract from the web server log alone. Any pair of URLs (P_i, P_j) outside the window is considered irrelevant and thus has a co-occurrence frequency of zero.

We then calculate the co-occurrence times $N_{i,j}$ of each pair of URLs (P_i, P_j) based on the window size W. We also calculate the request occurrence N_i,N_j for this pair of URLs.

$$P(P_i|P_j) = N_{i,j}/N_j \tag{1}$$

where $N_{i,j}$ is the number of times P_i and P_j occur together in window W, and N_j is the number of times P_j occurs.

We can select any of the following three distance functions for our applications; the first distance definition is the same as that by Perkowitz and Etzioni (1999).

$$Dis1(A, B) = Max(1/P(A|B), 1/P(B|A)) \tag{2}$$

$$Dis2(A, B) = 0.5(1/P(A|B) + 1/P(B|A)) \tag{3}$$

$$Dis3(A, B) = \sqrt{(1/P(A|B))(1/P(B|A))} \tag{4}$$

In all, we spend $W * L$ in distance calculation in worse case time for a web log file of size L. Because W is a constant, the time complexity for this step is $O(L)$.

In our experiments, we found that the first distance definition $Dis1(A, B)$ to be too restrictive because in our application it often yields infinite distances between many URLs. This gives very skewed results where many web pages are put into single separate clusters. The second definition is the arithmetic mean, whereas the

Table 1. Some clustering results using RDBC. If we use DBSCAN all these pages belong to the same cluster

Cluster 1	/shuttle/missions/41-c/news/ /shuttle/missions/61-b/ /shuttle/missions/sts-34/ /shuttle/missions/41-c/images/ ...
Cluster 2	/history/apollo/sa-2/news/ /history/apollo/sa-2/images/ /history/apollo/sa-1/sounds/ /history/apollo/sa-9/sa-9-info.html ...
Cluster 3	/software/winvn/userguide/3-3-2.htm /software/winvn/userguide/3-3-3.htm /software/winvn/userguide/3-8-1.htm /software/winvn/userguide/3-8-2.htm ...
...	...

third is the geometric mean. We have found that the third definition gives the best result in all three domains where we tested our algorithm.

5. Experiments on Web Page Clustering

In this section, we present our experimental results that test the performance of our RDBC algorithm. We test the clustering algorithm on three realistic data sets. We compare our algorithm's performance with that of DBSCAN.

We first analyze the data sets under consideration. Our experiments draw on data collected from three web sites: Monash University of Australia, NASA and MSN. The first data set is used in Albrecht et al. (1999) on predicting users' requests. It consists of the server log data collected during a 50-day period of time. It includes 525,378 total user requests of 6727 unique URLs (clicks) by 52,455 different IPs, consisting of 268,125 sessions. The NASA data set contains two months' worth of all HTTP requests to the NASA Kennedy Space Center WWW server in Florida. The log was collected from 00:00:00 August 1, 1995 through 23:59:59 August 31, 1995. In this period there were 1,569,898 requests. Timestamps have 1-second resolution. There are a total of 18,688 unique IPs requesting pages, having a total of 171,529 sessions. A total of 15,429 unique pages are requested. The MSN log is obtained from the server log of www.msn.com, with all identity of users stripped away. It consists of data collected from January 27, 1999 to March 26, 1999, with a total of 417,783 user requests. This log contains 722 unique IPs requesting 14,048 unique pages. The MSN log is unique in that some requests are from groups of users submitted by proxies or ISPs. Therefore the lengths of some sessions are long. For example, the long sessions range from 8384 consecutive requests to 166,073 requests.

We compare the clustering quality between out algorithm RDBC and DBSCAN on these three data sets. We also measure the efficiency for index page construction using the different clustering results. The tables and figures show our experimental results.

Table 2. Comparing clusters obtained by RDBC and DBSCAN on Monash University data

	RDBC	DBSCAN
Number of web pages	6727	6727
Run time (s)	20	22
$\epsilon/Minpts$	10/20 5/5	10/20
Number of clusters	125	6

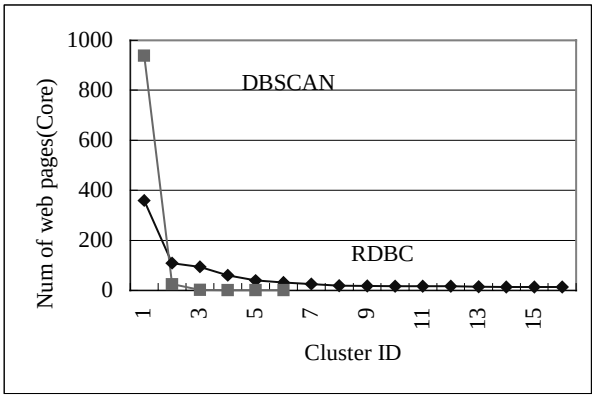

Fig. 1. Compare RDBC and DBSCAN on Monash University's log.

Table 1 gives an example of the clusters we obtained. Shown are example URLs that are extracted from the result. Table 2 and Fig. 1 show the clustering result and efficiency comparison between the DBSCAN and RDBC clustering algorithms. Using RDBC, while having about the same time complexity as DBSCAN, we can obtain more evenly distributed clusters, instead of putting everything in one cluster. The same result applies to both NASA and MSN data (see Tables 3 and 4 and Figs 2 and 3). By examining the contents of the logs, we found that the clusters we construct are more reasonable since similar topics are indeed grouped together and different topics are separated.

6. Cost Models and Optimality

Having obtained the clusters, we now turn our attention to the second contribution of the paper, in building web interfaces that provide useful short cuts for people.

Table 3. Comparing clusters obtained by RDBC and DBSCAN on NASA's data

	RDBC	DBSCAN
Number of web pages	15,429	15,429
Run time (s)	21	25
$\epsilon/Minpts$	10/20 5/5	10/20
Number of clusters	44	4

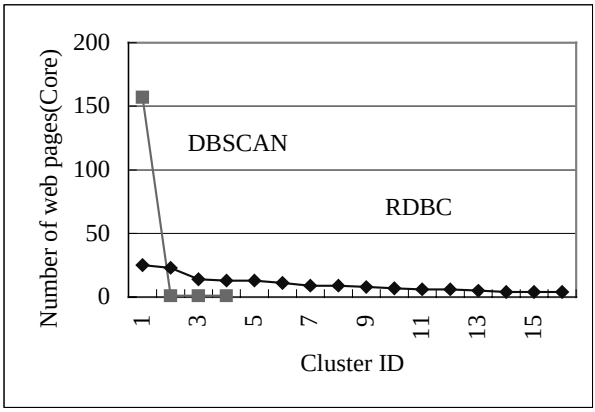

Fig. 2. Compare RDBC and DBSCAN on NASA's log.

Table 4. Comparing RDBC and DBSCAN on MSN's log

	RDBC	DBSCAN
Number of web pages	14,048	14,048
Run time (s)	21	24
$\epsilon/Minpts$	5/25 3/9	5/25
Number of clusters	125	3

We do this by providing index pages, which are table-of-content pages that we can put at the root of a web site. The idea of index pages was first proposed by Perkowitz and Etzioni (1999), but many open issues remain. In this section we address the important issue of how to extend their manually evaluated index pages by a novel technique to find an optimal construction of index pages automatically.

We first need to quantify the cost models of web browsing. The cost arising from web browsing can be summarized as a reduction of the transition costs between web pages. Various cost models can be used to describe the relation

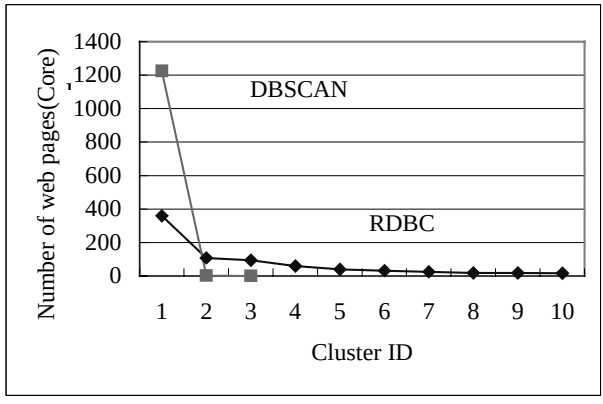

Fig. 3. Compare RDBC and DBSCAN on MSN's log.

between costs and the number of URLs traversed; in this paper we adopt a cost model such that the cost of a browsing session is directly proportional to the number of URLs on the browsing path. Short-cutting by offering index pages should not be considered to be cost free, however. Index pages themselves tax on the users' attention by requiring that users flip through extra pages in the process of finding their destinations. Therefore, when the number of index pages increases, the transition costs should decrease while, at the same time, the page cost associated with the need to flip through the index pages increases. In this section, we will explicitly model these cost functions and then strike an optimal balance between these cost functions. This optimal decision point will tell us how many index pages to create in order to have the lowest overall cost. We will then use this number to decide how to construct contents of the index pages from the output of the clustering algorithms.

Let $OverallCost$ be the overall cost of the browsing web pages and index pages. Let $PageCost$ be the cost of flipping index pages and $TransitionCost$ be the cost of switching from one web page to another. Then our cost models are as follows:

$$OverallCost = PageCost + TransitionCost \tag{5}$$

Let n be the number of index pages and N_{max} be the user defined maximum number of index pages. Then we define $PageCost$ to be a linear function of the number of index pages:

$$PageCost = \begin{cases} 0, & n < 1 \\ \frac{n}{N_{max}}, & 1 \leqslant n \end{cases} \tag{6}$$

For each index page $P_j j = 1, \ldots, n$, for each session S_i in the web log, $i = 1, \ldots, S$, where S is the total number of sessions in a log. Let $k_{i,j}$ be the number of URLs in P_j that also appears in S_i; $k_{i,j} - 1$ is the cost saved by including the index page P_j in session S_i.

$$TransitionCost = (TotalAccess - Sessions) - \sum_{i=1}^{S}(\sum_{j=1}^{n}(k_{i,j} - 1)) \tag{7}$$

We can then normalize the cost as the following:

$$OverallCost = \frac{TransitionCost}{TotalAccess - Sessions} \tag{8}$$

where $TotalAccess$ is the total number of transitions and $Sessions$ is the total number of sessions in the log. This normalization function defines a cost function within the range of zero and one.

We now describe how to compose index pages in our framework. In their algorithm, Perkowitz and Etzioni first compute clusters from the web logs and then put all clusters in index pages, so that each cluster will correspond to one index page. In our experience, we have found that often each cluster will contain a large number of index pages. When hundreds of hyperlinks are included in an index page, it is very difficult for a user to find the information he/she is looking for. In addition, we feel that there should be a limited number of index pages; if the user is required to read a huge number of index pages then it might defeat the purpose of including the index pages in the first place.

Therefore, in index page construction, we will include two parameters. Let L be

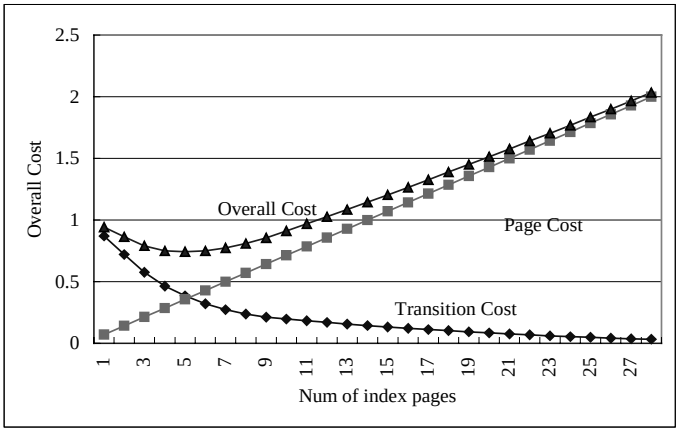

Fig. 4. Cost of page construction using the clustering results on Monash University log. We can see that the optimal value is around six.

the number of hyperlinks we would like to include in each index page, and let M be the number of index pages we wish to build. Algorithm *ConstructIndexPages* takes the parameters M and L and the clusters constructed by our RDBC algorithm, and produces M index pages as output:

Algorithm ConstructIndexPages(*Clusters*, M, L**)**
for $j = 1$ to M **do**
Sort Clusters by the frequency count of the top L web pages;
Extract the top L web pages from the first cluster and insert their hyperlinks into an index page;
if No cluster is left **or** size of each cluster $< L$ **then**
Stop;

More importantly, based on these cost functions, we can find a *minimal value M* for the *OverallCost* and its corresponding number of index pages to build. This optimal index page construction process represents another major contribution of our work. What we do is to analyze the overall cost as a function of the number of index pages M to construct, based on a fixed value of L. We can then find empirically the best value for M so as to minimize the overall cost of user browsing effort.

Figure 4 shows the overall cost function calculated from the combination of the web page switching cost and the index page cost with the Monash University data. As can be seen, for the linear functions that we choose, the cost function shows a minimal value at around five or six index pages for this domain. Likewise, cost minima can be obtained for the NASA data (Fig. 5) and MSN data (Fig. 6).

7. Conclusion and Future Work

The work reported in this paper is part of our ongoing effort in utilizing user information for building index pages of web pages. Our algorithm RDBC is based on abstracting the core points in a space of web pages and then applying

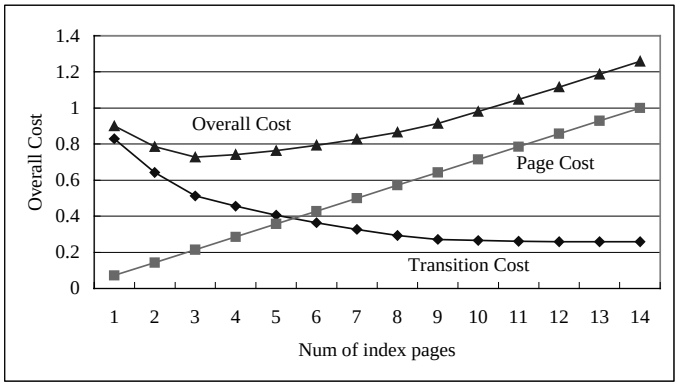

Fig. 5. Cost of page construction using the clustering results on NASA's log. We can see the optimal value is about three or four.

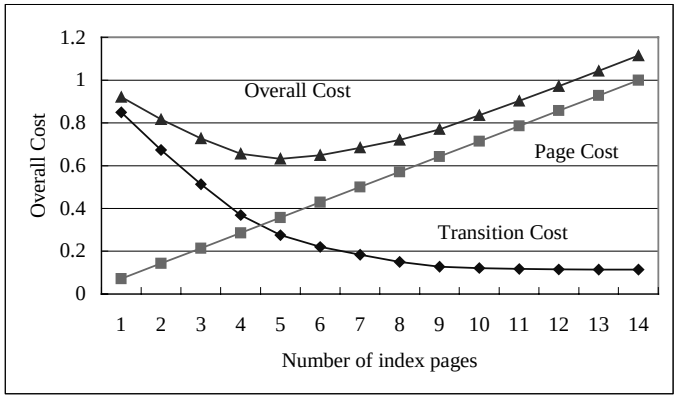

Fig. 6. Cost of page construction using the clustering results on MSN's log. We can see the optimal value is about three or four.

density-based clustering in the abstract space. This will result in an increase in the quality of the clusters. Once clusters are found, our algorithm constructs index pages and finds an optimal number of index pages so as to minimize user effort in browsing. This helps reduce the total cost of page transition and index page flipping.

In the future, we plan to try to perform experiments to learn more accurate cost models that allow us to give a more precise value for the total cost. We also need to conduct psychological studies to learn how many URLs to include in an index page to yield the best result for the user.

Acknowledgements. We thank anonymous reviewers for their very useful comments and suggestions.

References

Agrawal R, Imielinski T, Swami A (1993) Mining associations between sets of items in massive databases. In Buneman P, Jajodia S (eds) Proceedings of the ACM SIGMOD international conference on management of data, Washington DC, May 1993, pp 207–216

Agrawal R, Mannila H, Srikant R, Toivonen H Verkamo AI (1996) Fast discovery of association rules. In Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining, Ch 12. AAAI/MIT Press, Cambridge, MA, pp 307–328

Albrecht DW, Zukerman I, Nicholson AE (1999) Pre-sending documents on the WWW: a comparative study. In Dean T (ed) Proceedings of the 16th international joint conference on artificial intelligence, Sweden, July, 1999, pp 1274–1279

Armstrong R, Freitag D, Joachims T, Mitchell T (1995) Webwatcher: a learning apprentice for the world wide web. In Working notes of the AAAI spring symposium: information gathering from heterogeneous, distributed environments. AAAI Press, Menlo Park, CA, pp 6–12

Ester M, Kriegal HP, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In Simoudis E, Han J, Fayyad UM (eds) Proceedings of the second international conference on knowledge discovery and data mining. AAAI Press, Menlo Park, CA, 1996, pp 226–231

Ester M, Kriegal HP, Sander J, Wimmer M, Xu X (1998) Incremental clustering for mining in a data warehousing environment. In Gupta A, Shmueli O, Widom J (eds) Proceedings of 24rd international conference on very large data Bases, August 1998, New York. Morgan Kaufmann, San Mateo, CA, pp 323–333

Fink J, Kobsa A, Nill A (1996) User-oriented adaptivity and adaptability in the AVANTI project. In Designing for the web: empirical studies, Microsoft Usability Group, Redmond, WA

Perkowitz M, Etzioni O (1997) Adaptive web sites: an AI challenge. In Proceedings of the 15th international joint conference on artificial intelligence, Nagoya, Japan, August 1997. Morgan Kaufmann, Menlo Park, CA, pp 16–23

Perkowitz M, Etzioni O (1999) Towards adaptive web sites: conceptual framework and case study. In Proceedings of the eighth international World Wide Web conference, Toronto, Canada

Rasmussen E (1992) Clustering algorithms. In Frakes WB, Baeza-Yates R (eds) Information retrieval. Prentice-Hall, Englewood Cliffs, NJ, pp 419–442

Rocchio J (1966) Document retrieval systems: optimization and evaluation. PhD thesis, Harvard University

Savasere A, Omiecinski E, Navathe S (1995) An efficient algorithm for mining association rules in large databases. In Dayal U, Gray PMD, Nishio S (eds) Proceedings of 21st international conference on very large data bases, September 1995, Zurich, Switzerland. Morgan Kaufmann, San Mateo, CA, pp 432–444

Segal R (1996) Data mining as massive search. PhD thesis, University of Washington

Su Z, Yang Q, Zhang H (2000) A prediction system for multimedia pre-fetching in the Internet. In Proceedings of the eighth ACM multimedia conference, October 2000, Los Angeles, CA. ACM Press, New York, pp 3–11

Toivonen H (1996) Sampling large databases for association rules. In Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL (eds) Proceedings of the 22th international conference on very large data bases, September 1996, Mumbai (Bombay), India. Morgan Kaufmann, San Mateo, pp 134–145

Voorhees EM (1986) Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. Information Processing and Management 22: 465–476

Willet P (1988) Recent trends in hierarchical document clustering: a critical review. Information Processing and Management 24: 577-97

Xu X, Ester M, Kriegel HP, Sander J (1998) A distribution-based clustering algorithm for mining in large spatial databases. In Proceedings of the 14th international conference on data engineering, February 1998, Orlando, FL. IEEE Computer Society Press, Los Alamitos, CA, pp 324–331

Author Biographies

Zhong Su is a PhD student from the State Key Laboratory of Intelligent Technology and Systems, Computer Science Department, Tsinghua University. His research interests are in the area of AI, its applications in web mining and content-based image retrieval. Currently he is a visiting student of the Media Search Group in Microsoft Research China supervised by Hongjiang Zhang.

Qiang Yang is a professor at Simon Fraser University in Canada and an NSERC Industry Chair in Intelligent Software Systems. His specialty is in machine learning and planning, and their applications to web-based technologies and data mining. After graduating from the University of Maryland in 1989, he has been working at the University of Waterloo and Simon Fraser University in Canada.

Hongjiang Zhang joined Microsoft Research, China, as a Research Manager, from Hewlett-Packard Labs, Palo Alto, CA. He obtained his PhD from the Technical University of Denmark, and his BS from Zhengzhou University, China, both in Electrical Engineering. He has been actively engaged in research and development activities in the areas of video and image analysis, processing and retrieval, media compression and streaming, computer vision and their applications in consumer and enterprise markets.

Xiaowei Xu is a research scientist in Siemens AG, Corporate Technology. He specializes in data mining, database systems and their applications in fraud detection, customer relationship management, recommender systems and web information retrieval. He received his BS from Nankai University, China, his MS from Shenyang Institute for Computing Technology, Chinese Academy of Sciences, and his PhD from the University of Munich, Germany.

Yu-Hen Hu is a faculty member of the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, Wisconsin. His research interests include digital signal processing, computer architecture, and artificial neural networks. He has published more than 170 technical papers. His recent research interests have focused on image and video processing and human computer interface. Dr Hu has been an active member in IEEE. He has held various posts as associate editors and technical committee chairs. He is a fellow of IEEE.

Shaoping Ma graduated from the Computer Science and Technology Department of Tsinghua University, and received his bachelor's degree and master's degree in engineering in 1982 and 1984, respectively. He received his PhD degree from Tsinghua at 1997. He is a professor and vice-dean of the CS Department. His interests are knowledge engineering, Chinese information retrieval, Chinese character recognition and its post-processing technologies. He is the head of the Chinese National Natural Science Fund, '863' high-tech and international cooperation projects.

Correspondence and offprint requests to: Qiang Yang, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6. Email: qyang@cs.sfu.ca