Chapter IV

Retriever: Improving Web Search Engine Results Using Clustering

Anupam Joshi
University of Maryland, USA

Zhihua Jiang
American Management Systems, Inc., USA

Web search engines have become increasingly ineffective as the number of documents on the Web have proliferated. Typical queries retrieve hundreds of documents, most of which have no relation with what the user was looking for. The chapter describes a system named Retriever that uses a recently proposed robust fuzzy algorithm **RFCMdd** to cluster the results of a query from a search engine into groups. These groups and their associated keywords are presented to the user, who can then look into the URLs for the group(s) that s/he finds interesting. This application requires clustering in the presence of a significant amount of noise, which our system can handle efficiently. N-Gram and Vector Space methods are used to create the dissimilarity matrix for clustering. We discuss the performance of our system by comparing it with other state-of-the-art peers, such as Husky search, and present the results from analyzing the effectiveness of the N-Gram and Vector Space methods during the generation of dissimilarity matrices.

INTRODUCTION

Today, the WWW represents one of the largest, distributed, heterogeneous, semi-structured repositories of multimedia content. It is the de-facto medium for electronic commerce. Most e-tailer sites have a large amount of information, and presenting the appropriate information to the user is an important task. Other sites, which primarily deal with information provision (such as news organizations), deal with even larger volumes of information. The state-of-the-practice today is to use existing search engines to provide search functionality to the user. However, typical queries elicit hundreds, sometimes even thousands, of URLs from search engines, forcing the user to wade through them in order to find the URL(s) she needs. The same, at a reduced scale, happens when these engines are used to provide search capabilities at e-tailer sites. In large part, this limitation of search technology can be attributed to the following:

· *Polysemy: the words involved in the search have multiple meanings*. For example, a user searching for windows may be interested in either the operating system or the physical artifact.
· *Phrases: a phrase may be different from words in it*. e.g., the meaning of the phrase "partition magic" (a disk partition management tool) is quite different from the meaning of the individual words "partition" and "magic".
· *Term dependency: words in the terms are not totally independent of each other*. For example, a user may look for details about a product made by a particular company and type in Sun's Enterprise Computer Series. Obviously, each word in this term is dependent on each other.

Notice that these problems are independent of how good the algorithms that associate keywords with the contents of a page are.

One possible solution to this problem is to realize that the responses from search engines to a particular query can be broadly grouped into meaningful categories. If the user is shown these groups, possibly with some keyword type descriptions, they can select one (or more) that fit their perceived interests. Note that this is different from the site oriented grouping that some search engines present, typically in the form of a similar pages from this site link, since the aim here is to group together pages that potentially originate from completely different servers. There has been some prior work along these lines, such as that by Croft (Croft 1978), and more recent work by Cutting et al. (Cutting et al. 1992). However, this work is in the context of general text collections.

The recent work of Etzioni et al. (Zamir and Etzioni 1998) proposes the notion of clustering Web search engine results. To the best of our knowledge, this is the only other work besides our own that seeks to cluster search engine results on the fly. They have proposed an algorithm called Suffix Tree Clustering (STC) to group together snippets from Web pages. Snippets are typically the first few lines of (raw) HTML from the document. Essentially, this algorithm uses techniques from literature that allow the construction of suffix trees in time linear in the number of snippets assuming that the number of words in each snippet can be bounded by a constant. Each node in this tree captures a phrase (some suffix of the snippet string), and has associated with it those snippets that contain it. These nodes are viewed as base clusters since they group documents having a phrase in common. Each cluster is assigned a score based on the number of URLs in the cluster as well as the size of the phrase that they have in common. In order to account for the fact that Web pages in the same group may have more than a phrase in common, they then create a graph that has as its vertices the clusters identified by the suffix tree. They define a binary similarity measure between the clusters that is set to 1 if at least half of the documents in each cluster are common to both. Vertices representing similar clusters are connected by an edge. They then run a connected component finding algorithm, and each connected component is identified as a grouping of documents that are similar.

The rationale behind clustering snippets rather than the Web documents themselves is essentially speed. Clearly, clustering the (much) shorter snippets takes much less time than clustering full pages, and makes it possible to create clusters on the fly in response to a user's search request.

Given that clusters are formed out of snippets the efficacy of the phrase commonality criterion used by STC is not clear. While commonality of phrases may be a valid criterion in grouping large document collections, it is not clear if it is quite as appropriate for grouping snippets. Once common words (e.g. HTTP related terms) (which are treated as stop words) are eliminated from a snippet, what remains are essentially the heading of the page and the first sentence or two. Thus a phrase-based approach will likely do no better than a word commonality based approach, and may even be detrimental. Further, the use of binary similarity definition between the initial clusters leads to arbitrary decisions on whether two clusters should be merged. For example, using 0.5 as the threshold would imply that clusters with 0.49 similarity would not be merged, whereas those with 0.51 similarity would. The aim of clustering the results would be better served by defining a soft similarity measure that takes continuous values in the 0 to 1 range. Fuzzy clustering thus seems to be appropriate in this context. Moreover, clustering snippets involves dealing with a significant amount of noise. One reason for the noise is that the responses from the search engines themselves are noisy - many of the URLs returned have little or no connection with the original query, nor are they a part of any coherent "group" of URLs. The other reason is the use of snippets - often the first few sentences of a document will fail to capture its essence. Thus the clustering technique used must be robust - i.e. able to handle significant noise and outliers.

In this paper, we describe a system to cluster search engine results based on a robust relational fuzzy clustering algorithm that we have recently developed. We compare the use of the Vector Space based and N-Gram based dissimilarity measure to cluster the results from the search engines, such as MetaCrawler and Google. We start by providing a brief background on the clustering algorithm. We then describe our system, and discuss results from our experiments. These include a study of the efficiency on the Vector Space and the N-Gram methods, as well as a comparison with Husky Search (Huskysearch Web Site).

BACKGROUND

The Robust Fuzzy c-Medoids Algorithm (RFCMdd)

In related work, Krishnapuram et al. have recently proposed (Krishnapuram et al. 2001) an algorithm for fuzzy relational clustering based on the idea of identifying k-medoids. This algorithm is called Robust Fuzzy c-Medoids (RFCMdd). The worst case complexity of RFCMdd is O(n^2), but in practice it can be made linear and is an order of magnitude faster than the well known RFCM algorithm (Bezdek et al. 1991). Since we use a fuzzy algorithm, we are able to handle partial membership situations common in this task - in other words when the same URL may belong to two different groups but to different "degrees". Moreover, RFCMdd is highly robust and thus able to handle noise much better than traditional clustering approaches. Note that the data we cluster here (snippets) are highly noisy to begin with in terms of representing the actual documents. In addition, noise is also introduced in our distance generation measures. We now briefly describe RFCMdd as follows (Krishnapuram et al. 2001).

Let $\mathbf{X}=\{\mathbf{x}_i \mid i = 1,..,n\}$ be a set of n objects. Let $r(\mathbf{x}_i, \mathbf{x}_j)$ denote the dissimilarity between object $\mathbf{x}_i$, and object $\mathbf{x}_j$. Let $\mathbf{V}=\{\mathbf{v}_1, \mathbf{v}_2 , \ldots , \mathbf{v}_c\}$, $\mathbf{v}_i$ E $\mathbf{X}$ represent a subset of $\mathbf{X}$ with cardinality c, i.e., $\mathbf{V}$ is a c-subset of $\mathbf{X}$. Let $\mathbf{X}_c$ represent the set of all c-subsets $\mathbf{V}$ of $\mathbf{X}$. Each V represents a particular choice of prototypes for the c clusters in which we seek to partition the data. The Robust Fuzzy Medoids Algorithm (RFCMdd) minimizes the objective function:

$$J_m(\mathbf{V}; X) = \sum_{i=1}^{n} \sum_{i=1}^{c} u_{ij}^{m}\, r(\mathbf{x_j}, \mathbf{v}_i), \qquad (1)$$

where the minimization is performed over all $\mathbf{V}$ in $\mathbf{X}_c$. In (1), u_{ij} represents the fuzzy membership of $\mathbf{x}_j$ in cluster i. The membership u_{ij} can be defined heuristically in many different ways. We use the Fuzzy c-Means membership model given by:

$$u_{ij} = \frac{\left(\frac{1}{r(x_j, v_i)}\right)^{1/(m-1)}}{\sum_{k=1}^{c} \left(\frac{1}{r(x_j, v_k)}\right)^{1/(m-1)}}, \qquad (2)$$

where m E [1, 00) is the "fuzzifier." This generates a fuzzy partition of the data set X in the sense that the sum of the memberships of an object x_j across all classes is equal to 1. Since u_{ij} is a function of the dissimilarities $r(\mathbf{x}_j, \mathbf{v}_k)$, it can be eliminated from (1), and this is the reason J_m is shown as a function of $\mathbf{V}$ alone. Substituting the expression for u_{ij} in (2) into (1), we obtain:

$$J_m(\mathbf{V}; \mathbf{X}) = \sum_{i=1}^{n} \left(\sum_{i=1}^{c} (r(\mathbf{x}_j, \mathbf{v}_i))^{1/(1-m)} \right)^{1-m} = \sum_{j=1}^{n} h_j, \qquad (3)$$

where

$$h_j = \left(\sum_{i=1}^{c} (r(\mathbf{x}_j, \mathbf{v}_i))^{1/(1-m)} \right)^{1-m} \qquad (4)$$

is 1/c times the harmonic mean of the dissimilarities { $r(\mathbf{x}_j, \mathbf{v}_i)$) : i=1,..,c} when c=2. The objective function for the Robust Fuzzy c-Medoids (RFCMdd) algorithm is obtained by modifying (3) as follows:

$$J^T_{\mathrm{m}} (\mathbf{V}; \mathbf{X}) = \sum_{k=1}^{s} h_{k:n} \qquad (5)$$

However, the objective function in (5) cannot be minimized via the alternating optimization technique, because the necessary conditions cannot be derived by differentiating it with respect to the medoids. (Note that the solution space is discrete). Thus, strictly speaking, an exhaustive search over $\mathbf{X}_c$ needs to be used. However, following Fu's (Fu 1982) heuristic algorithm for a crisp version of (1), we describe a fuzzy algorithm that minimizes (5).

In (5) $h_{k:n}$ represents the k-th item when h_j, j=1,..,n, are arranged in ascending order, and s<n. The value of s is chosen depending on how many objects we would like to disregard in the clustering process. This allows the clustering algorithm to ignore outlier objects while minimizing the objective function. For example, when s = n/2, 50% of the objects are not considered in the clustering process, and the objective function is minimized when we pick c medoids in such a way that the sum of the harmonic-mean dissimilarities of 50% of the objects is as small as possible.

The quadratic complexity of the algorithm arises because when looking to update the medoid of a cluster, we consider all n objects as candidates. In practice, the new medoid is likely to be one that currently has a high membership in the cluster. Thus by restricting the search to say k objects with the highest membership in the cluster, the process can be made linear, i.e. O(kn), where k is a low integer. In that case, the complexity will be determined by the sorting operation required to find the smallest s (or equivalently the largest n-s) of the h_js. This is a good result, considering that robust algorithms are typically very expensive.

Notice that the algorithms as described assume that the number of clusters is known a priori, which is not the case here. This is a well known problem in clustering. We use a heuristic to automatically determine the number of clusters by initializing it to some large number, much larger than the expected (final) number of clusters. A SAHN type process is then used to hierarchically reduce the number of clusters. As we ascend up the hierarchy, we have to progressively increase the dissimilarity over which clusters will be merged. We note the change in this distance at each step, and assume the level at which the greatest change occurred has the right number of clusters.

Diagram 1: The Robust Fuzzy c Medoids Algorithm (RCMdd)

```
Fix the number of clusters $c$, and the fuzzifier $m$;
Randomly pick initial set of medoids: $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_1, \ldots, \mathbf{v}_c\}$ from $X_c$;
$iter = 0$;
**Repeat**
    Compute harmonic dissimilarities $h_j$ for $j = 1, \ldots, n$ using (4);
    Sort $h_j, j = 1, \ldots, n$ to create $h_{j:n}$;
    Keep the objects corresponding to the first $s$ $h_{j:n}$;
    Compute memberships for $s$ objects:
    **for** $j = 1$ **to** $s$ **do**
        **for** $i = 1$ **to** $c$ **do**
            Compute $u_{ij:n}$ by using (2);
        **endfor**
    **endfor**
    Store the current medoids:    $\mathbf{V}^{old} = \mathbf{V}$;
    Compute the new medoids:
    **for** $i = 1$ **to** $c$ **do**
        $q = \underset{1 \le k \le s}{\operatorname{argmin}} \sum_{j=1}^{s} u_{ij:n}^{m}\, r(\mathbf{x}_{k:n}, \mathbf{x}_{j:n})$
        $\mathbf{v}_i = \mathbf{x}_q$;
    **endfor**
    $iter = iter + 1$;
**Until** $(\mathbf{V}^{old} = \mathbf{V}$ or $iter = MAX\_ITER)$.
```

SYSTEM DESIGN

Introduction

In this section, we present the design of our Web clustering application called Retriever. Basically, our system is designed as a client-proxy-server system, and its architecture is illustrated in Figure 1. The proxy server is a Perl-based system that connects to regular search engines, such as MetaCrawler(http://www.metacrawler.com) and Google (http://www.google.com). The search term(s) entered by the user are passed onto the proxy via a CGI program. The proxy server then forwards the search term(s) to a specific search engine after transforming the query to a request in a format appropriate for that engine. The results that are returned by the search engine, including the Web links and their brief description (snippets), are trapped by the proxy. They are displayed to the user and are also saved to a file. The proxy then processes these results, applying the Vector Space or N-Gram method to create the (dis)similarity matrix. It then calls the RFCMdd algorithm to generate clusters of snippets. Each cluster is associated with a set of keywords, which is assumed to represent a concept. Next, the search engine proxy extracts the five most frequently occurring keywords as the representation of each cluster.

We will explain the process of query forwarding in section 3.2. The grouping results to user will be described in section 3.3.

Query Term Delivery

We use three different search engines as our URL sources, namely, MetaCrawler, Google, and Husky Search. We have also in the past used Lycos as a source. Note that Husky Search is not really a search engine, but a system to cluster search engine results similar to ours. The reason that we use Husky Search is that we aim to compare our system with it, as discussed in section 4. For the comparison to be fair, both systems must cluster the same set of Web links and snippets.

We point out that different search engines have individual query formats and their returned results are presented in different forms as well. For example, MetaCrawler does not accept a GET based query request, which we may see from the address box of any Web browser, such as Netscape. Instead, we have to send several header lines together with the query request. This information may involve the contents of "Referer" and "Cookie" headers.

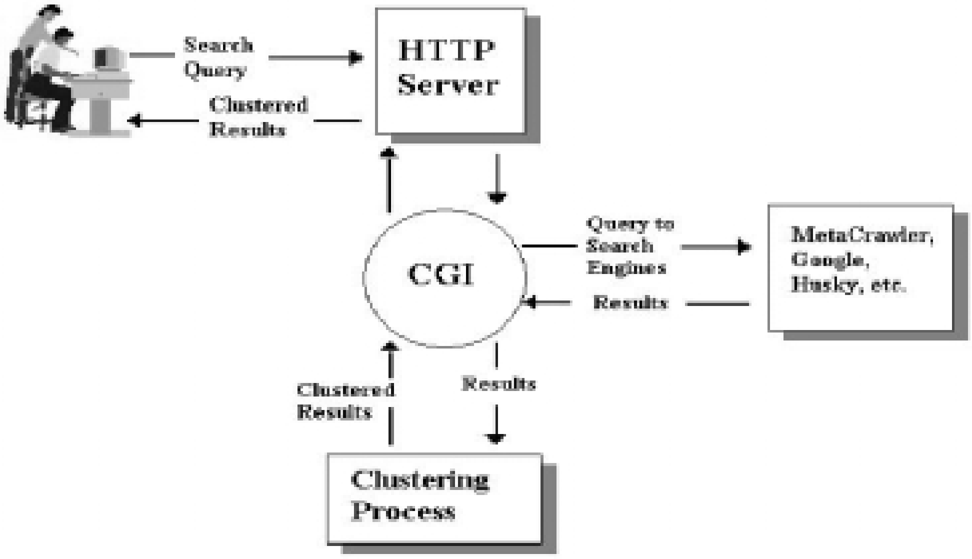
After receiving query request, MetaCrawler will return its results page by page. In order to retrieve the whole set of the query results, our search engine proxy will submit the HTTP request repeatedly until either the end of result or the maximal number of Web links is reached. On the contrary, Google is quite simple and does not require compiling multiple headers during request. However, it also returns results page by page, so similar procedures should be called to retrieve the whole collection of query results. Husky Search does not need either multiple headers or repeat page requests, but sends the results as another URL. Thus, we have to send another HTTP request to gather the actual query results.

In our experiment, extra effort is made to eliminate duplicate Web links in the query responses. We also introduce a special Java program to erase the HTML tags in the snippets and present only the title, URL, and snippets of the returned results. We do not include the set of HTML tags in the stop word list, because of two reasons. One is that we notice that search engines, like Google and Husky Search, may add some tags to highlight the query terms in the results. More importantly, we may treat different tags differently. For instance, we may erase specific tags, such as "&amp;" and "&nbsp;", because we believe that they are irrelevant to the meaning of this page. For others, both the tag and its corresponding close (e.g. <font> and </font) might need to be eliminated.

Illustration of Clustering Results

In response to a query, the system first returns a page quite similar to the page returned from any common search engine. It contains a brief list of titles, URLs, and their descriptions. If users can easily locate the links they want from among the first few paragraphs, they may simply click the link to the destination. Otherwise, they may click the button on the upper right corner, labeled "Clusters", to see the grouped results. After the button is clicked, another Web browser window will pop up to

Figure 1: System Architecture

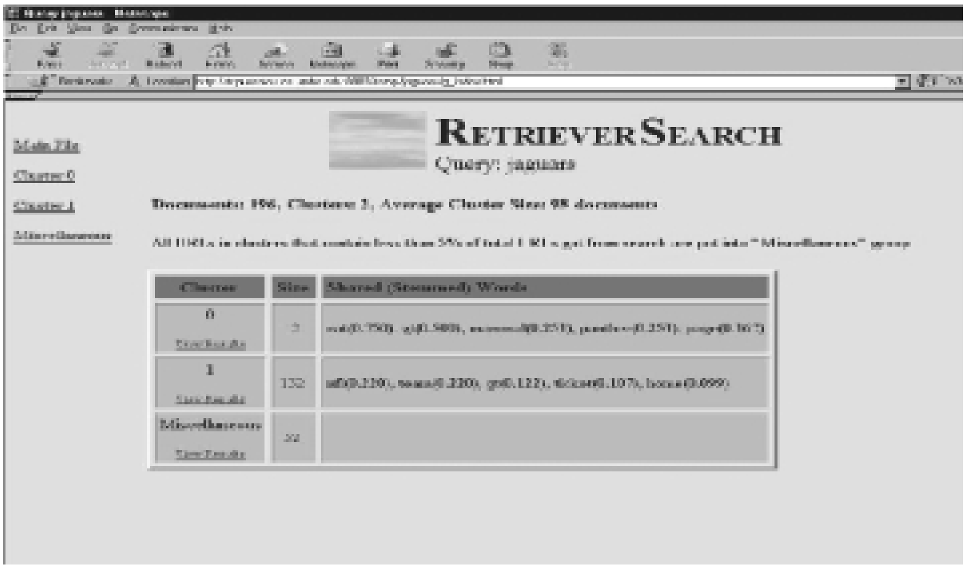


show the results in frames (Figure 2). Users may browse each cluster to pick out topics that they are interested in by following the link in the left frame. This causes the corresponding group of URLs to be displayed in the right frame (Figure 3). In addition, a miscellaneous group is included for each query result page. It could be empty, but if not, it will collect all URLs in clusters that contain less than 5% of total URLs returned from search. On the other hand, if the number of URLs returned is below some amount, say 20, the cluster algorithm will not be initiated, because we believe that with this small number of URLs, users could identify their interests easier and faster rather than waiting for the clustering results. Moreover, clusters formed from this small total may not be very meaningful.

Two different methods for the calculation of distance matrix are included in our system. The first one associated with "VectorSpace Search" button utilizes the TF/IDF method to generate the distance matrix. The second one, invoked by the "N-Gram Search" button, utilizes the N-Gram based measure to compute the dissimilarity matrix. Unlike the Vector Space measure, the N-Gram measure will not erase stop words and will not do word stemming either.

We also have a perl program which will periodically check the directory where all search results reside and clean the directories which contain the query results and exist more than 12 hours. This helps "garbage collect" free disk space. In practice, this method is quite powerful, because it does not compete for CPU time when users are waiting for search results and also it runs in background so it will not affect the running of other functions.

Figure 2: Group Index Page



RetrieverSearch
Query: jaguars

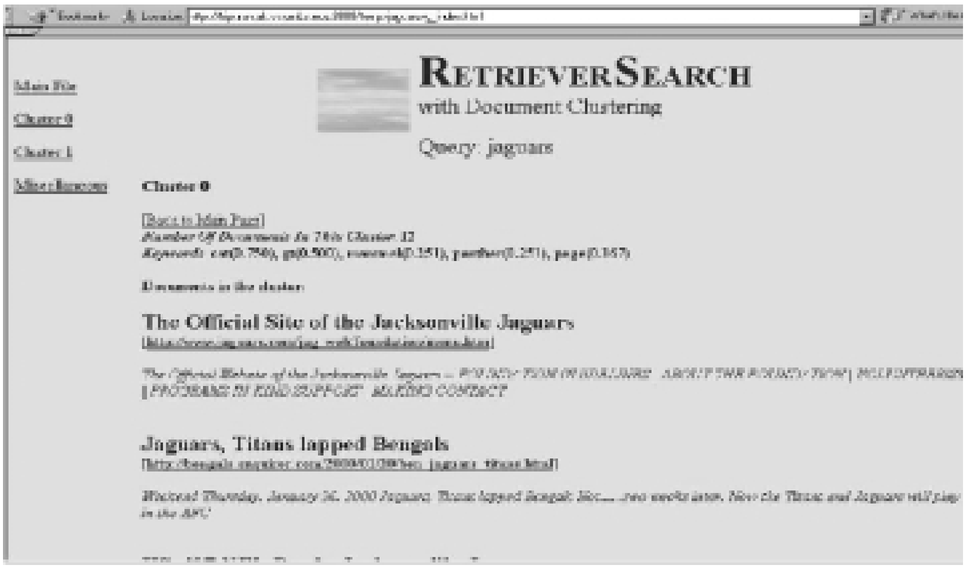
Main File
Cluster 0
Cluster 1
Miscellaneous

Documents: 196, Clusters: 2, Average Cluster Size: 98 documents

All URLs in clusters that contain less than 5% of total URLs got from search are put into "Miscellaneous" group

Cluster	Size	Shared (Stemmed) Words
0 View Results	[illegible]	cat(0.750), g[illegible](0.500), [illegible](0.251), panther(0.251), page(0.167)
1 View Results	132	nfl(0.220), team(0.220), [illegible](0.122), ticket(0.107), home(0.099)
Miscellaneous View Results	32	

Figure 3: Cluster Example Page



RetrieverSearch
with Document Clustering
Query: jaguars

Main File
Cluster 0
Cluster 1
Miscellaneous

Cluster 0

[Back to Main Page]
Number Of Documents In This Cluster: 12
Keywords: cat(0.750), g[illegible](0.500), [illegible](0.251), panther(0.251), page(0.167)

Documents in the cluster:

The Official Site of the Jacksonville Jaguars
[illegible]

The Official Website of the Jacksonville Jaguars — [illegible]
| PROGRAMS [illegible] SUPPORT [illegible] CONTACT

Jaguars, Titans lapped Bengals
[illegible]

[illegible] Thursday, January 2[illegible], 2000 Jaguars, Titans lapped Bengals [illegible] two weeks later. Now the Titans and Jaguars will play in the AFC

EXPERIMENTAL RESULTS

Introduction

This section presents several experimental results from — our comparative analysis of N-Gram vs Vector Space techniques for dissimilarity generation, and a comparison of these to Husky Search (Zamir and Etzioni 1998), which is based on Suffix Tree Clustering (STC). We first evaluate the importance of the value of n in the N-Gram based method in Section 4.2. Next, we compare the efficiency of the N-Gram based method vs the Vector Space based method for distance matrix generation in Section 4.3. We then illustrate the comparison of our system with Husky Search in Section 4.4. Next, we study the effect of noise elimination, achieved via in-process trimming, upon the RFCMdd algorithm in Section 4.5. Finally, we will provide a summary of some users' feedback we obtained from the preliminary survey of our site. We note that we do not require any personal identifying information for participation in the test. Thus, all results come without any individual information associated with specific feedback answer form.

It should be pointed out that lease keep in mind that the cluster numbers are simply arbitrary labels assigned by the clustering algorithm each time it is run. Thus when we show the cluster numbers across different methods (for example, N-Gram vs Vector Space) in the same row of the table — Cluster 1 of the Vector Space based test is not the same as cluster 1 of the N-Gram based one, and so on. The significance of these tables is in showing the distribution of URLs within the clusters generated by the same algorithm. In addition, we have used at least 10 to 15 examples for experiment of each section, but for the purpose of saving space here, we only show three of them as a brief illustration of the experimental results. More detailed results can be found in (Jiang 2000).

The Influence of the Value of n in the N-Gram Based RFCMdd Algorithm

In this section, we evaluate the effect of the value of n in the N-Gram based RFCMdd algorithm in order to discover the best value of n for the RFCMdd algorithm. moon river, mobile robots, and salsa are used as examples. Table 1 shows the different number of clusters generated given different value of n and Table 2 illustrates the number of URLs in each cluster and URL distributions. We notice that in the query, "mobile robots", when n=3, N-Gram RFCMdd creates 9 clusters, its largest number of clusters. This is nearly half the number of clusters generated by Vector Space based RFCMdd. However, when n=2, the least number of clusters are generated. The reason is that when the length of gram is too small, say n=1 or n=2, it is difficult to tell apart two different snippets due to the fact that the probability of the same gram appearing in different snippets becomes larger.

We also did the similar comparison on queries of salsa and moon river. The results do not show dramatic change of the number of clusters given the different length of grams, as in the experiment of mobile robots. But we observe that when n=5 and n=6, the number of clusters will stay similar and the major distribution of URLs in clusters will narrow down to 1-3 clusters. This observation is in consonance with prior work in IR, where 5 grams have been shown to be useful in document identification tasks.

Table 1: Number of clusters generated in each experiment as the N-Gram length is varied

moon river	**2**	**3**	**4**	**5**	**6**
number of URLs	100	100	100	100	100
number of clusters	2	3	3	2	2

Mobile robots	**2**	**3**	**4**	**5**	**6**
number of URLs	200	200	200	200	200
number of clusters	2	9	8	5	4

Salsa	**2**	**3**	**4**	**5**	**6**
number of URLs	200	200	200	200	200
number of clusters	4	4	3	3	4

Table 2: URL distribution and number of URLs in each cluster as the N-Gram length is varied

moon	**2**		**3**		**4**		**5**		**6**	
river	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**
C0	25	0.250	1	0.01	13	0.13	39	0.39	47	0.47
C1	75	0.750	43	0.430	86	0.860	61	0.610	53	0.53
C2			56	0.56	1	0.01				

mobile	**2**		**3**		**4**		**5**		**6**	
robots	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**
C0	105	0.525	9	0.045	7	0.035	28	0.14	42	0.210
C1	95	0.475	3	0.015	16	0.08	120	0.6	18	0.09
C2			48	0.240	47	0.235	45	0.225	131	0.655
C3			6	0.030	17	0.085	1	0.005	9	0.045
C4			4	0.020	15	0.075	6	0.03		
C5			3	0.015	68	0.340				
C6			122	0.610	27	0.135				
C7			4	0.020	3	0.015				
C8			1	0.005						

salsa	**2**		**3**		**4**		**5**		**6**	
	Absolute	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**	**Absolute**	**percentage**
C0	73	0.365	2	0.01	7	0.035	25	0.125	40	0.2
C1	25	0.125	182	0.91	183	0.915	163	0.815	152	0.76
C2	28	0.140	15	0.075	10	0.05	12	0.06	1	0.005
C3	74	0.370	1	0.005					7	0.035

N-Gram vs Vector Space

In this section, we compare the performance of the N-Gram and Vector Space methods for generating the dissimilarities between snippets. One possible evaluation metric is to compute the intra and inter distance among clusters that result from these dissimilarity measures. The intra cluster distance is the average distance between any two snippets within the same cluster. The inter cluster distance is the average distance between snippets in the two clusters. It is computed by averaging the distance of any pair of snippets in which one is from one cluster and the other is from the other cluster. Notice that shorter the intra distance means that the members within the same cluster are more relevant to each other, while longer inter distance interprets that the members of different groups become more irrelevant to one another.

We use moon river, star war, and salsa as examples of comparison. Tables 3, 4 and 5 present the summary of comparative results of intra and inter distance on these examples. Each table includes the total number of clusters generated by each method, the number of clusters after ignoring clusters which only contain one snippet (this is for intra distance calculation only), the average intra distance, the average inter distance, and the difference between the two. Notice that for N-Gram method, we employed two coefficient measures, namely, Dice Coefficient and Overlap Coefficient, which are denoted by formula 6 and 7. **A** is the number of n-grams in one string, **B** is the number of n-grams in the other, and **C** is the number in common. Moreover, for each measure, we computed dissimilarity both with and without stop word elimination prior to computing the n-grams . These are labeled as **SWE** and **WSWE** respectively. Therefore, for every example using N-Gram method, we have four results. Considering the space, we use shortened names on title of columns; for instance, NGOverlapSWE stands for the result from N-Gram method by using Overlap coefficient with Stop Word Elimination, NGOverlapWSWE, means that the result was generated by N-Gram based method using Overlap coefficient measure but without stop word elimination. The Dice Coefficient is

$$\mathbf{Coef}_{\mathbf{Dice}} = 1 - \frac{2 * C}{A + B} \qquad (6)$$

while the Overlap coefficient is

$$\mathbf{Coef}_{\mathbf{Overlap}} = 1 - \frac{C}{min(A, B)} \qquad (7)$$

In general, we note that the difference between intra and inter distance of clusters by four applied N-Gram based methods is much more significant than the one by Vector Space method. Specifically, in the examples of moon river and salsa, we observe that NGOverlapWSWE gives the largest difference between intra distance and inter distance, which are 0.266 and 0.162, and NGOverlapSWE gives the largest difference in the example of star war, which is 0.491, while Vector Space based method only provides 0.035, 0.086, and 0.061, respectively, for the examples of moon river, salsa, and star war. We also observe that normally, N-Gram based RFCMdd regardless of the variant used, usually creates fewer clusters than Vector Space based method. In addition, the URLs are distributed more narrowly across clusters in the result of the N-Gram based RFCMdd than Vector Space based one.

On the other hand, we point out that in the N-Gram method, there is not much difference between the Intra and Inter distance generated with stop word elimination (SWE) and without stop word elimination (WSWE). Sometimes N-Gram method with stop word elimination gives greater difference between Intra and Inter distance, such as in the examples, human resource and CDMA 2000 (Jiang 2000), while sometimes, the N-Gram method without stop word elimination (WSWE) gives greater difference, such as in the examples, moon river and salsa. This conclusion is also reflected in Ekmekcioglu's research (Ekmekcioglu et al. 1996) which claims that stop words and stemming are superior for a word-based system but are not significant for an n-gram based system.

In summary, N-Gram based RFCMdd always generates a fewer number of more focused clusters than the Vector Space based RFCMdd. In most queries, N-Gram based RFCMdd provides greater difference between Intra and Inter distance than Vector Space based RFCMdd. Therefore, it is obvious that N-Gram based (dis)similarity measure is more suitable to this application than Vector Space based measure.

RFCMdd vs Husky Search

We use an implementation of Etzioni et al.'s system (Huskysearch Web site), which is called Husky Search and is based on Suffix Tree Clustering (STC) as a comparison with our Retriever system, which is based on RFCMdd algorithm.

We present the summary result of comparison in Table 6. The keyword list generated from STC and from N-Gram based RFCMdd is presented in Tables from 7 to 9. The URL distribution of three examples is shown in Table 10.

Table 3: Summary of Inter and Intra Distance for moon river

moon river	VectorSpace	NGOverlapSWE	NGDiceSWE	NGOverlapWSWE	NGDiceWSWE
Num of URLs	92	92	92	92	92
Total Clu	21	4	5	20	17
After Adjusted	16	2	3	2	6
Average Intra	0.962	0.805	0.872	0.710	0.827
Average Inter	0.997	0.997	0.998	0.976	0.982
Difference	0.035	0.192	0.126	0.266	0.155

Table 4: Summary of Inter and Intra Distance for salsa

salsa	VectorSpace	NGOverlapSWE	NGDiceSWE	NGOverlapWSWE	NGDiceWSWE
Num of URLs	97	97	97	97	97
Total Clu	21	3	6	7	5
After Adjusted	7	3	4	6	3
Average Intra	0.914	0.859	0.886	0.823	0.865
Average Inter	1.000	0.961	0.986	0.985	0.989
Difference	0.086	0.102	0.100	0.162	0.124

Table 5: Summary of Inter and Intra Distance for star war

Star war	VectorSpace	NGOverlapSWE	NGDiceSWE	NGOverlapWSWE	NGDiceWSWE
Num of URLs	89	89	89	89	89
Total Clu	18	17	20	18	9
After Adjusted	17	3	7	3	3
Average Intra	0.934	0.475	0.817	0.633	0.823
Average Inter	0.995	0.966	0.973	0.974	0.984
Difference	0.061	0.491	0.156	0.341	0.161

For Zamir and Etzioni's STC algorithm, we present the keywords/phrase with the associated strength as reported by their algorithm in the tables. For the N-Gram based RFCMdd algorithm, we present the keywords most often associated with the cluster, as well as its normalized frequency of occurrence. For purposes of displaying these tables within page confines, we have sometimes presented only a part of a phrase or a long word, and indicated that by placing a *.

We observe that N-Gram based search leads to a fewer number of more focused clusters. When the set of snippets is large, such as in the example human resource, (illustrated in Jiang 2000), the major distribution of URLs in clusters will narrow down to 1-3 clusters by N-Gram based search, while neither Vector Space search nor Husky Search could provide such distribution. In addition, N-Gram based method could cluster pages with languages other than English, which neither Vector Space methods nor Husky Search can.

Vector Space Search requires about the same amount of time as Husky Search to return results; however, N-Gram Search takes much longer time. This is because computing n-grams and comparing them needs much more space and computation time. Thus there is a clear trade-off between the better results that the n-gram based approach provides versus the time it takes to compute the results. The precision of the lists of key words in both Retriever Search and Husky Search is similar. The user feedback (Jiang 2000) suggests that neither engine's keyword list is the clear winner-sometimes one could better help users identify the right group of URLs, sometimes the other.

Therefore, we could say that Retriever search is at least as good as Husky search, and can lead to better clusters if the user is willing to wait longer.

Noise Elimination

The RFCMdd algorithm that we mentioned in Section 2.1 uses "in-process " trimming — i.e. trims the outliers while it updates the medoids. The parameter it controls the percentage of data points (URLs in this case) that are thrown out as noise. Figure 4,5, and 6 show the results on three query examples, i.e., moon river,

Table 6: Number of URLs clustered by the two methods for three queries

moon river	Ngram RFCMdd	STC
number of URLs	34	34
number of clusters	3	3

mobile robots	Ngram RFCMdd	STC
number of URLs	28	28
number of clusters	1	4

Source code	Ngram RFCMdd	STC
number of URLs	426	426
number of clusters	2	15

Table 7: Clustering of Moon River Responses

STC	Key words/phrases				
C0	Classroom emc topicals (0.57)	Stand (0.57)	Sons (0.57)		
C1	City (1.0)	Hudson (0.50)	History (0.50)	Historic (0.50)	York (0.50)
C2	Music (1.0)	Theatre (0.5)	Branson (0.5)	Featured (0.5)	

N-Gram	Key words/phrases				
C0	paradis (0.376)	life (0.376)	photograph (0.376)	Gphy (0.251)	Grunion (0.251)
C1	son (1.0)	wheel (0.858)	evolut (0.715)	stephan (0.572)	famou (0.429)
C2	your (1.5)	Fairchild (1.0)	fish (1.0)	kirkwood (1.0)	Moonthousand (1.0)

Table 8: Clustering of Mobile Robots Responses

STC	Key words/phrases					
C0	Mobile Robotics (0.38)	robotics (0.77)	autonomous (0.38)	Learning (0.31)	Robot (0.61)	
C1	mobile robots (1.0)					
C2	Robots Pages (1.0)	Exclusion (0.40)	document (0.40)	people (0.40)	programs (0.40)	Automaticall (0.40)
C3	Research (1.0)	Robotics (0.60)	robots (0.60)	vision (0.40)	sensors (0.40)	

N-Gram	Key words/phrases				
C0	Interfac (0.297)	Grow (0.223)	Approachfrequent (0.186)	Depart (0.186)	Ccd (0.186)

mobile robots, and shuttle. Tables from 11 to 13 present the same results in a clearer tabular form.

From Figure 4, 5 and 6, we notice the number of ejected URLs will increase when the value of option "it" is increased — this is evident from the definition of the parameter. .We also observe that the CPU time that was used to compute the grouping results decreases with the decreasing number of URLs available for clustering, in other words as the algorithm recognizes more points as noise and discards them, it speeds up as well.

Table 9: Clustering of Source Code Responses

STC	Key words/phrases					
C0	Subobject Code (0.11)	payroll unit code (0.07)	source documents (0.07)	Transaction Code (0.06)		
C1	section of the (0.14)	Faculty Members (0.13)	Advisory Committee on (0.10)	Committee on Faculty (0.09)		
C2	Program (1.0)					
C3	Document (1.0)					
C4	Data (1.0)	Funding (0.38)	Distribution (0.38)	Object (0.35)	Budget (0.35)	HEPPS (0.35)
C5	IMAP (1.0)					
C6	Note that the (0.32)	Note (1.0)	data (0.36)	funding (0.36)	record (0.36)	
C7	Morse code (0.56)	input (0.63)	special (0.44)			
C8	Command (0.55)	Subroutine (0.40)	initialization (0.35)	switch (0.35)	function (0.35)	
C9	added (0.79)	additional (0.54)	faculty (0.37)			
C10	Class (1.0)	Documents (0.35)	object (0.35)			
C11	type (1.0)	Funding (0.46)	object (0.46)	time (0.42)	budget (0.42)	data (0.35)
C12	implementation (0.67)	fair (0.44)	paper (0.44)	document (0.38)		
C13	code examples (0.44)	specific (0.69)	programmer (0.63)	directory (0.43)	Request (0.37)	
C14	current (1.0)	distribution (0.55)	department (0.48)	funding (0.44)	units (0.44)	Budget (0.44)

N-Gram	Key words/phrases				
C0	code (4.825)	sourc (3.039)	faculti (0.863)	fund (0.771)	Perl (0.474)
C1	Code (3.289)	Sourc (1.919)	Cours (0.458)	Perl (0.231)	Data (0.224)

Table 10: Number and Percentage of URLs in each cluster

moon river	**Ngram RFCMdd**		**STC**	
/	**Absolute**	**percentage**	**Absolute**	**percentage**
C0	8	0.24	7	0.21
C1	7	0.21	4	0.12
C2	2	0.06	4	0.12
Misc	17	0.50	22	0.65

Mobile robots	**Ngram RFCMdd**		**STC**	
/	**Absolute**	**percentage**	**Absolute**	**percentage**
C0	27	0.96	13	0.46
C1			12	0.43
C2			5	0.19
C3			5	0.19
Misc	1	0.04	6	0.21

source code	**Ngram RFCMdd**		**STC**	
/	**Absolute**	**percentage**	**Absolute**	**percentage**
C0	131	0.31	189	0.44
C1	295	0.69	77	0.18
C2			44	0.10
C3			43	0.10
C4			42	0.10
C5			41	0.10
C6			22	0.05
C7			16	0.04
C8			20	0.05
C9			24	0.06
C10			31	0.07
C11			28	0.07
C12			18	0.04
C13			16	0.04
C14			27	0.06
Misc	0	0.00	131	0.31

Figure 4: Test of it's influence-moon river

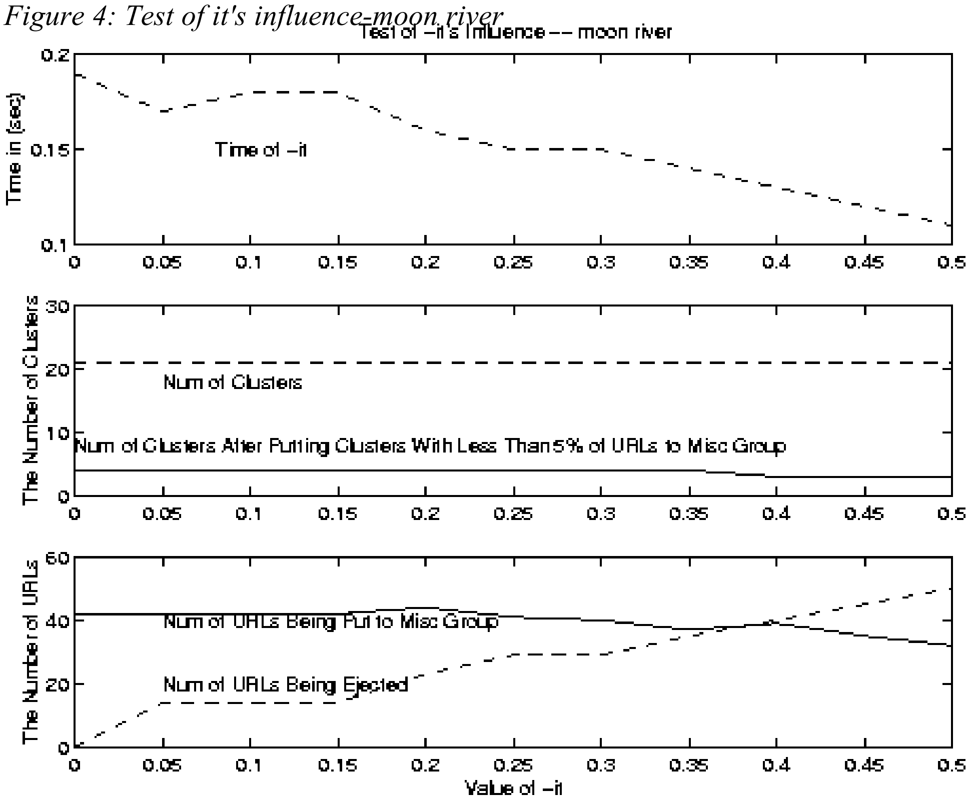


Figure 5: Test of it's influence-mobile robots

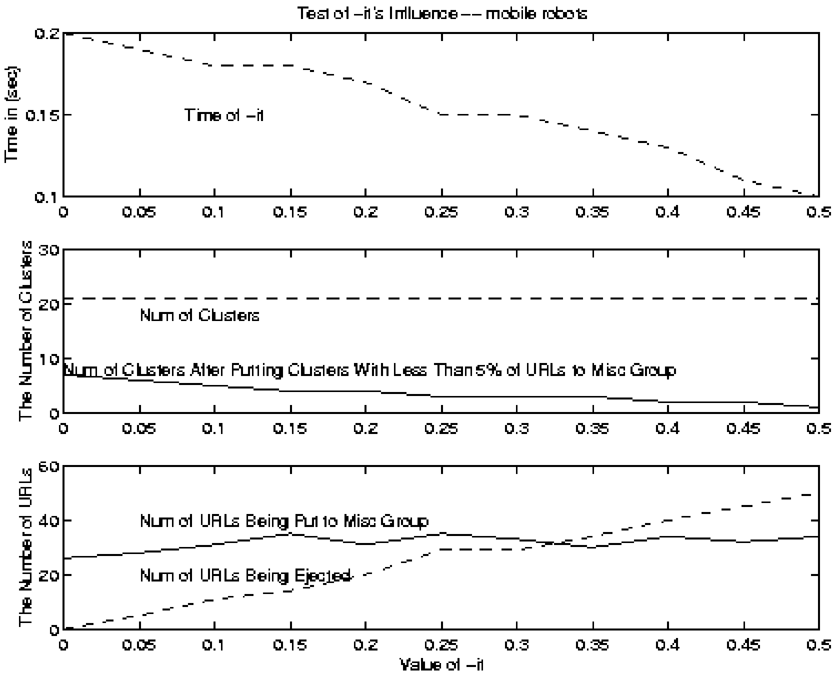
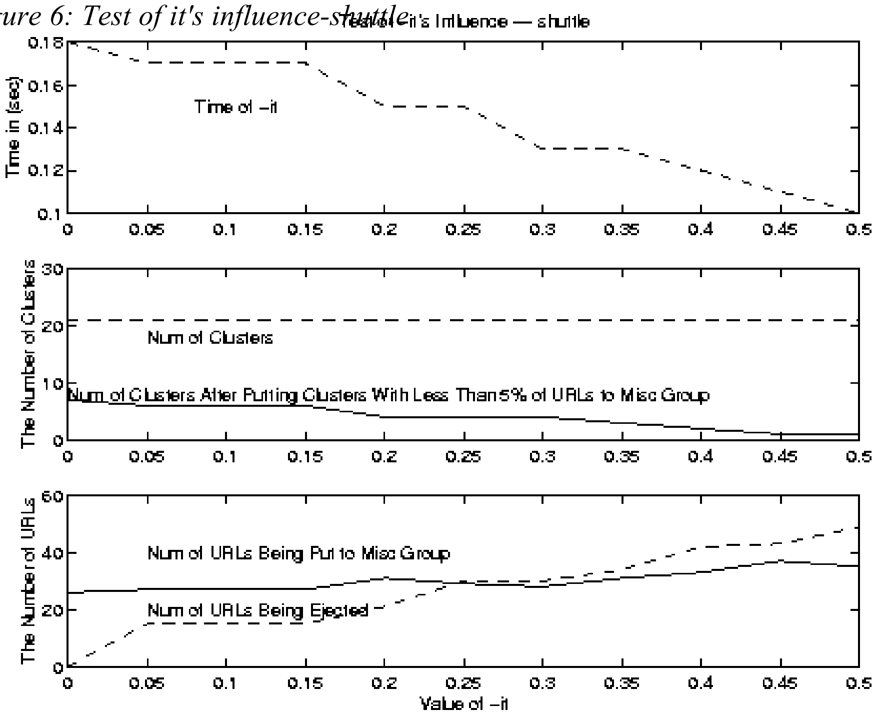


Figure 6: Test of it's influence-shuttle



Total URLs: 100 Total Clusters: 21

After putting clusters with less than 5% of total URLs into Miscellaneous group ("X" means the snippets in this cluster have been moved to Miscellaneous group)

Table 11: Clustering Results: moon river (it)

clu/-it val	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**
C0	30	16	16	16	14	13	13	11	11	11	10
C1	X	X	X	X	X	X	X	X	X	X	X
C2	X	X	X	X	X	X	X	X	X	X	X
C3	X	X	X	X	X	X	X	X	X	X	X
C4	X	X	X	X	X	X	X	X	X	X	X
C5	X	X	X	X	X	X	X	X	X	X	X
C6	X	X	X	X	X	X	X	X	X	X	X
C7	X	X	X	X	X	X	X	X	X	X	X
C8	X	X	X	X	X	X	X	X	X	X	X
C9	X	X	X	X	X	X	X	X	X	X	X
C10	X	X	X	X	X	X	X	X	X	X	X
C11	7	7	7	7	7	7	7	6	X	X	X
C12	15	15	15	15	15	12	11	11	10	9	8
C13	X	X	X	X	X	X	X	X	X	X	X
C14	6	6	6	6	X	X	X	X	X	X	X
C15	X	X	X	X	X	X	X	X	X	X	X
C16	X	X	X	X	X	X	X	X	X	X	X
C17	X	X	X	X	X	X	X	X	X	X	X
C18	X	X	X	X	X	X	X	X	X	X	X
C19	X	X	X	X	X	X	X	X	X	X	X
C20	X	X	X	X	X	X	X	X	X	X	X
Ejected	0	14	14	14	23	29	29	35	40	45	50
Miscell	42	42	42	42	44	41	40	37	39	35	32
CPUTime	0.19	0.17	0.18	0.18	0.16	0.15	0.15	0.14	0.13	0.12	0.11

Total URLs: 100 Total Clusters: 21

After putting clusters with less than 5% of total URLs into Miscellaneous group ("X" means the snippets in this cluster have been moved to Miscellaneous group)

Table 12: Clustering Results: mobile robots (it)

clu/-it val	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**
C0	X	X	X	X	X	X	X	X	X	X	X
C1	X	X	X	X	X	X	X	X	X	X	X
C2	X	X	X	X	X	X	X	X	X	X	X
C3	X	X	X	X	X	X	X	X	X	X	X
C4	X	X	X	X	X	X	X	X	X	X	X
C5	X	X	X	X	X	X	X	X	X	X	X
C6	X	X	X	X	X	X	X	X	X	X	X
C7	X	X	X	X	X	X	X	X	X	X	X
C8	X	X	X	X	X	X	X	X	X	X	X
C9	X	X	X	X	X	X	X	X	X	X	X
C10	X	X	X	X	X	X	X	X	X	X	X
C11	X	X	X	X	X	X	X	X	X	X	X
C12	X	X	X	X	X	X	X	X	X	X	X
C13	X	X	X	X	X	X	X	X	X	X	X
C14	7	7	7	6	6	X	X	X	X	X	X
C15	7	7	7	7	7	7	7	6	X	X	X
C16	7	6	X	X	X	X	X	X	X	X	X
C17	X	X	X	X	X	X	X	X	X	X	X
C18	6	X	X	X	X	X	X	X	X	X	X
C19	32	32	29	29	27	23	23	22	20	17	16
C20	9	9	9	9	9	8	8	8	6	6	X
Ejected	0	5	11	14	20	29	29	34	40	45	50
Miscell	26	28	31	35	31	35	33	30	34	32	34
CPUTime	0.2	0.19	0.18	0.18	0.17	0.15	0.15	0.14	0.13	0.11	0.1

SUMMARY OF SURVEY

We conducted user studies for this system by organizing trials among a small group of students in our school who were conversant with search engines and the Internet, but had little knowledge of our research and the techniques underlying our prototype system. Each participant was asked to use the system over a period of time and fill out a feedback form on-line. Due to privacy concerns, we do not ask or save any personal identifying information associated with specific answers during this survey, so all the feedback is anonymous.

The survey contains four parts. The first one is called "Vector Space vs Common search engine". Its purpose is to solicit a user's comments based on his/her experience of Vector Space Search in our Retriever site and his/her experience of other common search engines, which do not group query results. It involves two groups of questions. One group inquires about user's impression on using this site compared to other common search engines, from the perspective of the employed grouping function, the speed, and the layout. The other queries the user's satisfaction on grouping results, such as the effectiveness of locating the desired URLs and

Total URLs: 97 Total Clusters: 21

After putting clusters with less than 5% of total URLs into Miscellaneous group ("X" means the snippets in this cluster have been moved to Miscellaneous group)

Table 13: Clustering Results: shuttle (it)

clu/-it val	**0**	**0.05**	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**
C0	16	X	X	X	X	X	X	X	X	X	X
C1	5	5	5	5	5	5	5	5	5	X	X
C2	7	7	7	7	7	5	5	5	X	X	X
C3	X	X	X	X	X	X	X	X	X	X	X
C4	X	X	X	X	X	X	X	X	X	X	X
C5	5	5	5	5	X	X	X	X	X	X	X
C6	X	X	X	X	X	X	X	X	X	X	X
C7	X	X	X	X	X	X	X	X	X	X	X
C8	X	X	X	X	X	X	X	X	X	X	X
C9	X	X	X	X	X	X	X	X	X	X	X
C10	5	5	5	5	X	X	X	X	X	X	X
C11	X	X	X	X	X	X	X	X	X	X	X
C12	X	X	X	X	X	X	X	X	X	X	X
C13	X	X	X	X	X	X	X	X	X	X	X
C14	X	X	X	X	X	X	X	X	X	X	X
C15	X	X	X	X	X	X	X	X	X	X	X
C16	X	X	X	X	X	X	X	X	X	X	X
C17	X	X	X	X	X	X	X	X	X	X	X
C18	9	9	9	9	9	6	6	X	X	X	X
C19	24	24	24	24	24	23	23	22	17	17	13
C20	X	X	X	X	X	X	X	X	X	X	X
Ejected	0	15	15	15	21	30	30	34	42	43	49
Miscell	26	27	27	27	31	29	28	31	33	37	35
CPUTime	0.18	0.17	0.17	0.17	0.15	0.15	0.13	0.13	0.12	0.11	0.1

the precision of key word list associated with each group to see if it facilitates users to select the right group of URLs. The second part is similar to the first one, except that it is designed with the comparative questions between N-Gram Search and Common search engines. The third page is to survey users' opinions comparing Vector Space based search vs N-Gram based search. This requires users to first utilize both methods on same or similar queries before filling out the form. The last part of this survey is to compare the performance, speed, precision of the list of key words of each cluster, and other issues related to Retriever search and Husky search. Users were instructed to select "HuskySearch" as the source of URLs in Retriever, and also open another browser window direct to Husky Search (http://zhadum.cs.washington.edu/). Then they type in the same query terms in both search sites and compare the results.

We point out that in the comparison between HuskySearch and Retriever, in order to get the same data set for clustering, we need to first send the query to Husky search and retrieve their results without grouping. This takes about the same amount of time when users type in the same query in Husky search site and obtain the results from it. Users were made aware of this additional delay, and asked to ignore it in so far possible when comparing Retriever's speed with HuskySearch.

The records of this survey show that most users prefer the grouping results introduced by our site, both Vector Space search and N-Gram search. Some users are satisfied with the fewer number of more focused clusters created by N-Gram search, while some of them incline to use Vector Space search, in that N-Gram search requires more time and space than Vector Space search when dealing with the generation of gram-based (dis)similarity matrices. In real life, this is understandable because sometimes users may not intend to obtain the most precise grouping results. They would rather wait for a shorter time to get reasonable but not exact grouping results than wait for longer time, especially when the number of URLs retrieved is not large. However, when the data set becomes bigger, more users seem to turn to N-Gram search.

In the comparison between N-Gram search and Husky search, about 20% of users feel that N-Gram search is slower or slightly slower than Husky search, but most of them agree that N-Gram search returns fewer number of clusters. As for the precision of lists of key words associated with each cluster is concerned, there is no clear winner with user opinion split half-half.

In the "other comments" section, most users recognize that in general, N-Gram search provides fewer number of more focused clusters and it could handle pages with languages other than English, but it needs to shorten the computation time and improve the performance of the function to create better a key word list of each cluster.

CONCLUSIONS

In this paper, we have presented a system that seeks to improve the process for finding relevant URLs for the users. We show that such information personalization is important for E-Commerce. In particular, the results returned from a search engine are clustered on the fly into groups, and these groups and their associated keywords are presented to the users. The user can then choose to examine URLs in one or more of these groups based on the keywords. We have used a new robust relational fuzzy clustering algorithm based on the idea of medoids that has been recently developed (RFCMdd). In addition, we introduce and compare the N-Gram method and the Vector Space method to generate the (dis)similarity distance matrix. Our preliminary results show that the algorithm gives good results on Web snippets. The N-Gram based approach seems to perform better than the Vector Space based approach, and as well as similar systems reported in literature. Moreover, our approach captures the overlapping clusters idea (a URL can belong to more than one group to different degrees) more elegantly and does not force the user to make an arbitrary "binary" choice of declaring two groups to be similar. In addition, our system is robust, i.e. not sensitive to noise and outliers which are the common occurrence in this domain. We realize of course that in order to achieve speed (clustering the results from the search engine as they come back), we are sacrificing accuracy by clustering only the snippets rather than the documents themselves.

ACKNOWLEDGMENTS

Partial support of this work by grants from National Science Foundation (IIS 9801711 and IIS 9875433 to Joshi) is gratefully acknowledged. We also thank Prof. Raghu Krishnapuram from the Colorado School of Mines and his student Liyu Yi, with whom we collaborated in developing the clustering algorithm. The authors would also like to thank Karuna Joshi for proof reading the paper and reformatting it, from latex to MSWord .

REFERENCES

Bezdek, J. C., Hathaway, R. J. and Windham, M. P. (1991). Numerical Comparison of the RFCM and AP Algorithms for Clustering Relational Data, 24, 783-791.

Croft, W. B. (1978). Organizing and Searching Large Files of Documents. PhD Thesis, Cambridge University.

Cutting, D., Krager, D., Pedersen, J. and Tukey, J. (1992). Scatter/gather: A cluster based approach to browsing large document collections. In *Proceedings of the 16th ACM SIGIR Conference*, 318-329.

Ekmekcioglu, F. C., Lynch, M. and Willett, P. (1996). Stemming and N-gram matching for term conflation in Turkish texts. *Information Research News*, 7(1), 2-6.

Fu, K. S. (1982). Syntactic Pattern Recognition and Applications, Academic Press, San Diego, CA.

Huskysearch. Available on the World Wide Web at: http://zhadum.cs.washington.edu/.

Jiang, Z. (2000). Using Robust Clustering Methods to Group Query Responses from Web Search Engines. Master's Thesis, University of Maryland Baltimore County.

Krishnapuram, R., Joshi, A., Nasraoui, O. and Yi, L. (2001). Low complexity fuzzy relational clustering algorithms for Web mining. Accepted for publication in *IEEE Transactions on Fuzzy Systems*.

Zamir, O. and Etzioni, O. (1998). Web document clustering: A feasibility demonstration. In *Proceedings of the ACM SIGIR '98*.