Learn from Web Search Logs to Organize Search Results

Xuanhui Wang
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
xwang20@cs.uiuc.edu

ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
czhai@cs.uiuc.edu

ABSTRACT

Effective organization of search results is critical for improving the utility of any search engine. Clustering search results is an effective way to organize search results, which allows a user to navigate into relevant documents quickly. However, two deficiencies of this approach make it not always work well: (1) the clusters discovered do not necessarily correspond to the interesting aspects of a topic from the user's perspective; and (2) the cluster labels generated are not informative enough to allow a user to identify the right cluster. In this paper, we propose to address these two deficiencies by (1) learning "interesting aspects" of a topic from Web search logs and organizing search results accordingly; and (2) generating more meaningful cluster labels using past query words entered by users. We evaluate our proposed method on a commercial search engine log data. Compared with the traditional methods of clustering search results, our method can give better result organization and more meaningful labels.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Clustering, Search process

General Terms: Algorithm, Experimentation

Keywords: Search result organization, search engine logs, interesting aspects

1. INTRODUCTION

The utility of a search engine is affected by multiple factors. While the primary factor is the soundness of the underlying retrieval model and ranking function, how to organize and present search results is also a very important factor that can affect the utility of a search engine significantly. Compared with the vast amount of literature on retrieval models, however, there is relatively little research on how to improve the effectiveness of search result organization.

The most common strategy of presenting search results is a simple ranked list. Intuitively, such a presentation strategy is reasonable for non-ambiguous, homogeneous search results; in general, it would work well when the search results are good and a user can easily find many relevant documents in the top ranked results.

However, when the search results are diverse (e.g., due to ambiguity or multiple aspects of a topic) as is often the case in Web search, the ranked list presentation would not be effective; in such a case, it would be better to group the search results into clusters so that a user can easily navigate into a particular interesting group. For example, the results in the first page returned from Google for the ambiguous query "jaguar" (as of Dec. 2nd, 2006) contain at least four different senses of "jaguar" (i.e., car, animal, software, and a sports team); even for a more refined query such as "jaguar team picture", the results are still quite ambiguous, including at least four different jaguar teams – a wrestling team, a jaguar car team, Southwestern College Jaguar softball team, and Jacksonville Jaguar football team. Moreover, if a user wants to find a place to download a jaguar software, a query such as "download jaguar" is also not very effective as the dominating results are about downloading jaguar brochure, jaguar wallpaper, and jaguar DVD. In these examples, a clustering view of the search results would be much more useful to a user than a simple ranked list. Clustering is also useful when the search results are poor, in which case, a user would otherwise have to go through a long list sequentially to reach the very first relevant document.

As a primary alternative strategy for presenting search results, clustering search results has been studied relatively extensively [9, 15, 26, 27, 28]. The general idea in virtually all the existing work is to perform clustering on a set of top-ranked search results to partition the results into *natural* clusters, which often correspond to different subtopics of the general query topic. A label will be generated to indicate what each cluster is about. A user can then view the labels to decide which cluster to look into. Such a strategy has been shown to be more useful than the simple ranked list presentation in several studies [8, 9, 26].

However, this clustering strategy has two deficiencies which make it not always work well:

First, the clusters discovered in this way do not necessarily correspond to the interesting aspects of a topic from the user's perspective. For example, users are often interested in finding either "phone codes" or "zip codes" when entering the query "area codes." But the clusters discovered by the current methods may partition the results into "local codes" and "international codes." Such clusters would not be very useful for users; even the best cluster would still have a low precision.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.
Copyright 2007 ACM 978-1-59593-597-7/07/0007 ...$5.00.

Second, the cluster labels generated are not informative enough to allow a user to identify the right cluster. There are two reasons for this problem: (1) The clusters are not corresponding to a user's interests, so their labels would not be very meaningful or useful. (2) Even if a cluster really corresponds to an interesting aspect of the topic, the label may not be informative because it is usually generated based on the contents in a cluster, and it is possible that the user is not very familiar with some of the terms. For example, the ambiguous query "jaguar" may mean an animal or a car. A cluster may be labeled as "panthera onca." Although this is an accurate label for a cluster with the "animal" sense of "jaguar", if a user is not familiar with the phrase, the label would not be helpful.

In this paper, we propose a different strategy for partitioning search results, which addresses these two deficiencies through imposing a user-oriented partitioning of the search results. That is, we try to figure out what aspects of a search topic are likely interesting to a user and organize the results accordingly. Specifically, we propose to do the following:

First, we will learn "interesting aspects" of similar topics from search logs and organize search results based on these "interesting aspects". For example, if the current query has occurred many times in the search logs, we can look at what kinds of pages viewed by the users in the results and what kind of words are used together with such a query. In case when the query is ambiguous such as "jaguar" we can expect to see some clear clusters corresponding different senses of "jaguar". More importantly, even if a word is not ambiguous (e.g., "car"), we may still discover interesting aspects such as "car rental" and "car pricing" (which happened to be the two primary aspects discovered in our search log data). Such aspects can be very useful for organizing future search results about "car". Note that in the case of "car", clusters generated using regular clustering may not necessarily reflect such interesting aspects about "car" from a user's perspective, even though the generated clusters are coherent and meaningful in other ways.

Second, we will generate more meaningful cluster labels using past query words entered by users. Assuming that the past search logs can help us learn what specific aspects are interesting to users given the current query topic, we could also expect that those query words entered by users in the past that are associated with the current query can provide meaningful descriptions of the distinct aspects. Thus they can be better labels than those extracted from the ordinary contents of search results.

To implement the ideas presented above, we rely on search engine logs and build a history collection containing the past queries and the associated clickthroughs. Given a new query, we find its related past queries from the history collection and learn aspects through applying the star clustering algorithm [2] to these past queries and clickthroughs. We can then organize the search results into these aspects using categorization techniques and label each aspect by the most representative past query in the query cluster.

We evaluate our method for result organization using logs of a commercial search engine. We compare our method with the default search engine ranking and the traditional clustering of search results. The results show that our method is effective for improving search utility and the labels generated using past query words are more readable than those generated using traditional clustering approaches.

The rest of the paper is organized as follows. We first review the related work in Section 2. In Section 3, we describe search engine log data and our procedure of building a history collection. In Section 4, we present our approach in details. We describe the data set in Section 5 and the experimental results are discussed in Section 6. Finally, we conclude our paper and discuss future work in Section 7.

2. RELATED WORK

Our work is closely related to the study of clustering search results. In [9, 15], the authors used Scatter/Gather algorithm to cluster the top documents returned from a traditional information retrieval system. Their results validate the cluster hypothesis [20] that relevant documents tend to form clusters. The system "Grouper" was described in [26, 27]. In these papers, the authors proposed to cluster the results of a real search engine based on the snippets or the contents of returned documents. Several clustering algorithms are compared and the Suffix Tree Clustering algorithm (STC) was shown to be most effective. They also showed that using snippets is as effective as using whole documents. However, an important challenge of document clustering is to generate meaningful labels for clusters. To overcome this difficulty, in [28], supervised learning algorithms are studied to extract meaningful phrases from the search result snippets and these phrases are then used to group search results. In [13], the authors proposed to use a *monothetic* clustering algorithm, in which a document is assigned to a cluster based on *a single* feature, to organize search results, and the single feature is used to label the corresponding cluster. Clustering search results has also attracted a lot of attention in industry and commercial Web services such as Vivisimo [22]. However, in all these works, the clusters are generated solely based on the search results. Thus the obtained clusters do not necessarily reflect users' preferences and the generated labels may not be informative from a user's viewpoint.

Methods of organizing search results based on text categorization are studied in [6, 8]. In this work, a text classifier is trained using a Web directory and search results are then classified into the predefined categories. The authors designed and studied different category interfaces and they found that category interfaces are more effective than list interfaces. However predefined categories are often too general to reflect the finer granularity aspects of a query.

Search logs have been exploited for several different purposes in the past. For example, clustering search queries to find those Frequent Asked Questions (FAQ) is studied in [24, 4]. Recently, search logs have been used for suggesting query substitutes [12], personalized search [19], Web site design [3], Latent Semantic Analysis [23], and learning retrieval ranking functions [16, 10, 1]. In our work, we explore past query history in order to better organize the search results for future queries. We use the star clustering algorithm [2], which is a graph partition based approach, to learn interesting aspects from search logs *given* a new query. Thus past queries are clustered in a *query specific* manner and this is another difference from previous works such as [24, 4] in which *all* queries in logs are clustered in an offline batch manner.

3. SEARCH ENGINE LOGS

Search engine logs record the activities of Web users, which

ID	Query	URL	Time
1	win zip	http://www.winzip.com	xxxx
1	win zip	http://www.swinzip.com/winzip	xxxx
2	time zones	http://www.timeanddate.com	xxxx
...	...	...	...

Table 1: Sample entries of search engine logs. Different ID's mean different sessions.

reflect the actual users' needs or interests when conducting Web search. They generally have the following information: text queries that users submitted, the URLs that they clicked after submitting the queries, and the time when they clicked. Search engine logs are separated by *sessions*. A session includes a single query and all the URLs that a user clicked after issuing the query [24]. A small sample of search log data is shown in Table 1.

Our idea of using search engine logs is to treat these logs as past history, learn users' interests using this history data automatically, and represent their interests by representative queries. For example, in the search logs, a lot of queries are related to "car" and this reflects that a large number of users are interested in information about "car." Different users are probably interested in different aspects of "car." Some are looking for renting a car, thus may submit a query like "car rental"; some are more interested in buying a used car, and may submit a query like "used car"; and others may care more about buying a car accessory, so they may use a query like "car audio." By mining all the queries which are related to the concept of "car", we can learn the aspects that are likely interesting from a user's perspective. As an example, the following is some aspects about "car" learned from our search log data (see Section 5).

1. `car rental, hertz car rental, enterprise car rental, ...`
2. `car pricing, used car, car values, ...`
3. `car accidents, car crash, car wrecks, ...`
4. `car audio, car stereo, car speaker, ...`

In order to learn aspects from search engine logs, we preprocess the raw logs to build a history data collection. As shown above, search engine logs consist of sessions. Each session contains the information of the text query and the clicked Web page URLs, together with the time that the user did the clicks. However, this information is limited since URLs alone are not informative enough to tell the intended meaning of a submitted query accurately. To gather rich information, we enrich each URL with additional text content. Specifically, given the query in a session, we obtain its top-ranked results using the search engine from which we obtained our log data, and extract the snippets of the URLs that are clicked on according to the log information in the corresponding session. All the titles, snippets, and URLs of the clicked Web pages of that query are used to represent the session.

Different sessions may contain the same queries. Thus the number of sessions could be quite huge and the information in the sessions with the same queries could be redundant. In order to improve the scalability and reduce data sparseness, we aggregate all the sessions which contain exactly the same queries together. That is, for each unique query, we build a "pseudo-document" which consists of all the descriptions of its clicks in all the sessions aggregated. The keywords contained in the queries themselves can be regarded as brief summaries of the pseudo-documents. All these pseudo-documents form our history data collection, which is used to learn interesting aspects in the following section.

4. OUR APPROACH

Our approach is to organize search results by aspects learned from search engine logs. Given an input query, the general procedure of our approach is:

1. Get its related information from search engine logs. All the information forms a working set.
2. Learn aspects from the information in the working set. These aspects correspond to users' interests given the input query. Each aspect is labeled with a representative query.
3. Categorize and organize the search results of the input query according to the aspects learned above.

We now give a detailed presentation of each step.

4.1 Finding Related Past Queries

Given a query q, a search engine will return a ranked list of Web pages. To know what the users are really interested in given this query, we first retrieve its past similar queries in our preprocessed history data collection.

Formally, assume we have N pseudo-documents in our history data set: $H = \{Q_1, Q_2, ..., Q_N\}$. Each Q_i corresponds to a unique query and is enriched with clickthrough information as discussed in Section 3. To find q's related queries in H, a natural way is to use a text retrieval algorithm. Here we use the OKAPI method [17], one of the state-of-the art retrieval method. Specifically, we use the following formula to calculate the similarity between query q and pseudo-document Q_i:

$$\sum_{w \in q \bigcap Q_i} c(w,q) \times IDF(w) \times \frac{(k_1+1) \times c(w,Q_i)}{k_1((1-b)+b\frac{|Q_i|}{avdl}) + c(w,Q_i)}$$

where k_1 and b are OKAPI parameters set empirically, $c(w,Q_i)$ and $c(w,q)$ are the count of word w in Q_i and q respectively, $IDF(w)$ is the inverse document frequency of word w, and $avdl$ is the average document length in our history collection.

Based on the similarity scores, we rank all the documents in H. The top ranked documents provide us a working set to learn the aspects that users are usually interested in. Each document in H corresponds to a past query, and thus the top ranked documents correspond to q's related past queries.

4.2 Learning Aspects by Clustering

Given a query q, we use $H_q = \{d_1, ..., d_n\}$ to represent the top ranked pseudo-documents from the history collection H. These pseudo-documents contain the aspects that users are interested in. In this subsection, we propose to use a clustering method to discover these aspects.

Any clustering algorithm could be applied here. In this paper, we use an algorithm based on graph partition: the star clustering algorithm [2]. A good property of the star clustering in our setting is that it can suggest a good label for each cluster naturally. We describe the star clustering algorithm below.

4.2.1 Star Clustering

Given H_q, star clustering starts with constructing a pairwise similarity graph on this collection based on the vector space model in information retrieval [18]. Then the clusters are formed by dense subgraphs that are star-shaped. These clusters form a cover of the similarity graph. Formally, for each of the n pseudo-documents $\{d_1, ..., d_n\}$ in the collection H_q, we compute a TF-IDF vector. Then, for each pair of documents d_i and d_j $(i \neq j)$, their similarity is computed as the cosine score of their corresponding vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, that is

$$sim(d_i, d_j) = \cos(\mathbf{v}_i, \mathbf{v}_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{|\mathbf{v}_i| \cdot |\mathbf{v}_j|}.$$

A similarity graph G_σ can then be constructed as follows using a similarity threshold parameter σ. Each document d_i is a vertex of G_σ. If $sim(d_i, d_j) > \sigma$, there would be an edge connecting the corresponding two vertices. After the similarity graph G_σ is built, the star clustering algorithm clusters the documents using a greedy algorithm as follows:

1. Associate every vertex in G_σ with a flag, initialized as *unmarked*.
2. From those *unmarked* vertices, find the one which has the highest degree and let it be u.
3. Mark the flag of u as *center*.
4. Form a cluster C containing u and all its neighbors that are not marked as *center*. Mark all the selected neighbors as *satellites*.
5. Repeat from step 2 until all the vertices in G_σ are marked.

Each cluster is *star-shaped*, which consists a single *center* and several *satellites*. There is only one parameter σ in the star clustering algorithm. A big σ enforces that the connected documents have high similarities, and thus the clusters tend to be small. On the other hand, a small σ will make the clusters big and less coherent. We will study the impact of this parameter in our experiments.

A good feature of the star clustering algorithm is that it outputs a *center* for each cluster. In the past query collection H_q, each document corresponds to a query. This *center* query can be regarded as the most representative one for the whole cluster, and thus provides a *label* for the cluster naturally. All the clusters obtained are related to the input query q from a different perspective, and they represent the possible aspects of interests about query q of users.

4.3 Categorizing Search Results

In order to organize the search results according to users' interests, we use the learned aspects from the related past queries to categorize the search results. Given the top m Web pages returned by a search engine for q: $\{s_1, ..., s_m\}$, we group them into different aspects using a categorization algorithm.

In principle, any categorization algorithm can be used here. Here we use a simple centroid-based method for categorization. Naturally, more sophisticated methods such as SVM [21] may be expected to achieve even better performance.

Based on the pseudo-documents in each discovered aspect C_i, we build a centroid prototype $\mathbf{p}_i$ by taking the average of all the vectors of the documents in C_i:

$$\mathbf{p}_i = \frac{1}{|C_i|} \sum_{l \in C_i} \mathbf{v}_l.$$

All these $\mathbf{p}_i$'s are used to categorize the search results. Specifically, for any search result s_j, we build a TF-IDF vector. The centroid-based method computes the cosine similarity between the vector representation of s_j and each centroid prototype $\mathbf{p}_i$. We then assign s_j to the aspect with which it has the highest cosine similarity score.

All the aspects are finally ranked according to the number of search results they have. Within each aspect, the search results are ranked according to their original search engine ranking.

5. DATA COLLECTION

We construct our data set based on the MSN search log data set released by the Microsoft Live Labs in 2006 [14]. In total, this log data spans 30 days from 05/01/2006 to 05/30/2006. There are 8,144,000 queries, 3,441,000 distinct queries, and 4,649,000 distinct URLs in the raw data.

To test our algorithm, we separate the whole data set into two parts according to the time: the first 2/3 data is used to simulate the historical data that a search engine accumulated, and we use the last 1/3 to simulate future queries. In the history collection, we clean the data by only keeping those frequent, well-formatted, English queries (queries which only contain characters 'a', 'b', ..., 'z', and space, and appear more than 5 times). After cleaning, we get 169,057 unique queries in our history data collection totally. On average, each query has 3.5 distinct clicks. We build the "pseudo-documents" for all these queries as described in Section 3. The average length of these pseudo-documents is 68 words and the total data size of our history collection is 129MB.

We construct our test data from the last 1/3 data. According to the time, we separate this data into two test sets equally for cross-validation to set parameters. For each test set, we use every session as a test case. Each session contains a single query and several clicks. (Note that we do not aggregate sessions for test cases. Different test cases may have the same queries but possibly different clicks.) Since it is infeasible to ask the original user who submitted a query to judge the results for the query, we follow the work [11] and opt to use the clicks associated with the query in a session to *approximate* relevant documents. Using clicks as judgments, we can then compare different algorithms for organizing search results to see how well these algorithms can help users reach the clicked URLs.

Organizing search results into different aspects is expected to help informational queries. It thus makes sense to focus on the informational queries in our evaluation. For each test case, i.e., each session, we count the number of different clicks and filter out those test cases with fewer than 4 clicks under the assumption that a query with more clicks is more likely to be an informational query. Since we want to test whether our algorithm can learn from the past queries, we also filter out those test cases whose queries can not retrieve at least 100 pseudo-documents from our history collection. Finally, we obtain 172 and 177 test cases in the first and

second test sets respectively. On average, we have 6.23 and 5.89 clicks for each test case in the two test sets respectively.

6. EXPERIMENTS

In the section, we describe our experiments on the search result organization based past search engine logs.

6.1 Experimental Design

We use two baseline methods to evaluate the proposed method for organizing search results. For each test case, the first baselineis the default ranked list from a search engine (rank). The second baseline is to organize the search results by clustering them (cluster-based). For fair comparison, we use the same clustering algorithm as our log-based method (i.e., star clustering). That is, we treat each search result as a document, construct the similarity graph, and find the star-shaped clusters. We compare our method (log-based) with the two baseline methods in the following experiments. For both cluster-based and log-based methods, the search results within each cluster is ranked based on their original ranking given by the search engine.

To compare different result organization methods, we adopt a similar method as in the paper [9]. That is, we compare the quality (e.g., precision) of the best cluster, which is defined as the one with the largest number of relevant documents. Organizing search results into clusters is to help users navigate into relevant documents quickly. The above metric is to simulate a scenario when users always choose the right cluster and look into it. Specifically, we download and organize the top 100 search results into aspects for each test case. We use Precision at 5 documents (P@5) in the best cluster as the primary measure to compare different methods. P@5 is a very meaningful measure as it tells us the *perceived* precision when the user opens a cluster and looks at the first 5 documents. We also use Mean Reciprocal Rank (MRR) as another metric. MRR is calculated as

$$MRR = \frac{1}{|T|}\sum_{q \in T}\frac{1}{r_q}$$

where T is a set of test queries, r_q is the rank of the first relevant document for q.

To give a fair comparison across different organization algorithms, we force both cluster-based and log-based methods to output the same number of aspects and force each search result to be in one and only one aspect. The number of aspects is fixed at 10 in all the following experiments. The star clustering algorithm can output different number of clusters for different input. To constrain the number of clusters to 10, we order all the clusters by their sizes, select the top 10 as aspect candidates. We then assign each remaining search result to one of these selected 10 aspects that has the highest similarity score with the corresponding aspect centroid. In our experiments, we observe that the sizes of the best clusters are all larger than 5, and this ensures that P@5 is a meaningful metric.

6.2 Experimental Results

Our main hypothesis is that organizing search results based on the users' interests learned from a search log data set is more beneficial than to organize results using a simple list or cluster search results. In the following, we test our hypothesis from two perspectives – organization and labeling.

Method	Test set 1		Test set 2	
	MMR	P@5	MMR	P@5
Baseline	0.7347	0.3325	0.7393	0.3288
Cluster-based	0.7735	0.3162	0.7666	0.2994
Log-based	0.7833	0.3534	0.7697	0.3389
Cluster/Baseline	5.28%	-4.87%	3.69%	-8.93%
Log/Baseline	6.62%	6.31%	4.10%	3.09%
Log/Cluster	1.27%	11.76%	0.40%	13.20%

Table 2: Comparison of different methods by MMR and P@5. We also show the percentage of relative improvement in the lower part.

Comparison	Test set 1	Test set 2
	Impr./Decr.	Impr./Decr.
Cluster/Baseline	53/55	50/64
Log/Baseline	55/44	60/45
Log/Cluster	68/47	69/44

Table 3: Pairwise comparison w.r.t the number of test cases whose P@5's are improved versus decreased w.r.t the baseline.

6.2.1 Overall performance

We compare three methods, basic search engine ranking (baseline), traditional clustering based method (cluster-based), and our log based method (log-based), in Table 2 using MRR and P@5. We optimize the parameter σ's for each collection individually based on P@5 values. This shows the best performance that each method can achieve. In this table, we can see that in both test collections, our method is better than both the "rank" and the "cluster-based" methods. For example, in the first test collection, the baseline method of MMR is 0.734, the cluster-based method is 0.773 and our method is 0.783. We achieve higher accuracy than both cluster-based (1.27% improvement) and the rank-based baseline method (6.62% improvement). The P@5 values are 0.332 for rank baseline, 0.316 for cluster-based baseline, but 0.353 for our method. Our method improves over the baseline method by 6.31%, while the cluster-based method even decreases the accuracy. This is because cluster-based method organizes the search results only based on the contents. Thus it could organize the results differently from users' preferences. This confirms our hypothesis of the bias of the cluster-based method. Comparing our method with the cluster-based method, we achieve significant improvement on both test collections. The p-values of the significance tests based on P@5 on both collections are 0.01 and 0.02 respectively. This shows that our log-based method is effective to learn users' preferences from the past query history, and thus it can organize the search results in a more useful way to users.

We showed the optimal results above. To test the sensitivity of the parameter σ of our log-based method, we use one of the test sets to tune the parameter to be optimal and then use the tuned parameter on the other set. We compare this result (log tuned outside) with the optimal results of both cluster-based (cluster optimized) and log-based method (log optimized). Figure 1 shows the comparison. We can see that, as expected, the performance using the parameter tuned on a separate set is worse than the optimal performance. However, our method still performs much better than the *optimal* result of cluster-based method on

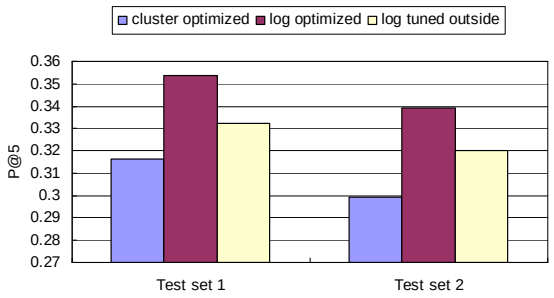

Figure 1: Results using parameters tuned from the other test collection. We compare it with the optimal performance of the cluster-based and our log-based methods.

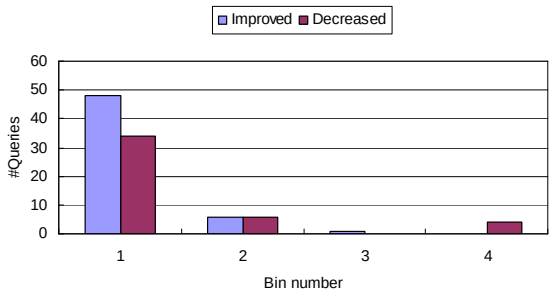

Figure 2: The correlation between performance change and result diversity.

both test collections.

In Table 3, we show pairwise comparisons of the three methods in terms of the numbers of test cases for which P@5 is increased vs. decreased. We can see that our method improves more test cases compared with the other two methods. In the next section, we show more detailed analysis to see what types of test cases can be improved by our method.

6.2.2 Detailed Analysis

To better understand the cases where our log-based method can improve the accuracy, we test two properties: result diversity and query difficulty. All the analysis below is based on test set 1.

Diversity Analysis: Intuitively, organizing search results into different aspects is more beneficial to those queries whose results are more diverse, as for such queries, the results tend to form two or more big clusters. In order to test the hypothesis that log-based method help more those queries with diverse results, we compute the size ratios of the biggest and second biggest clusters in our log-based results and use this ratio as an indicator of diversity. If the ratio is small, it means that the first two clusters have a small difference thus the results are more diverse. In this case, we would expect our method to help more. The results are shown in Figure 2. In this figure, we partition the ratios into 4 bins. The 4 bins correspond to the ratio ranges $[1, 2)$, $[2, 3)$, $[3, 4)$, and $[4, +\infty)$ respectively. ($[i, j)$ means that $i \leq$ ratio $< j$.) In each bin, we count the numbers of test cases whose P@5's are improved versus decreased with respect to rank baseline, and plot the numbers in this figure. We can observe that when the ratio is smaller, the log-based method can improve more test cases. But when the ratio is large, the log-based method can not improve over the baseline. For example, in bin 1, 48 test cases are improved and 34 are decreased. But in bin 4, all the 4 test cases are decreased. This confirms our hypothesis that our method can help more if the query has more diverse results. This also suggests that we should "turn off" the option of re-organizing search results if the results are not very diverse (e.g., as indicated by the cluster size ratio).

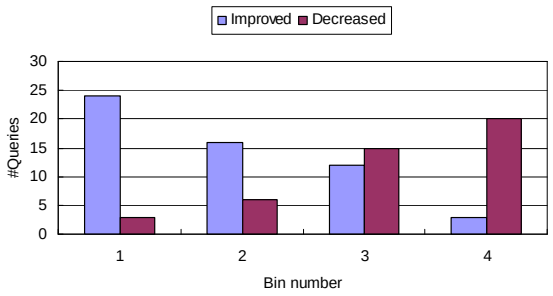

Figure 3: The correlation between performance change and query difficulty.

Difficulty Analysis: Difficult queries have been studied in recent years [7, 25, 5]. Here we analyze the effectiveness of our method in helping difficult queries. We quantify the query difficulty by the Mean Average Precision (MAP) of the original search engine ranking for each test case. We then order the 172 test cases in test set 1 in an increasing order of MAP values. We partition the test cases into 4 bins with each having a roughly equal number of test cases. A small MAP means that the utility of the original ranking is low. Bin 1 contains those test cases with the lowest MAP's and bin 4 contains those test cases with the highest MAP's. For each bin, we compute the numbers of test cases whose P@5's are improved versus decreased. Figure 3 shows the results. Clearly, in bin 1, most of the test cases are improved (24 vs 3), while in bin 4, log-based method may decrease the performance (3 vs 20). This shows that our method is more beneficial to difficult queries, which is as expected since clustering search results is intended to help difficult queries. This also shows that our method does not really help easy queries, thus we should turn off our organization option for easy queries.

6.2.3 Parameter Setting

We examine parameter sensitivity in this section. For the star clustering algorithm, we study the similarity threshold parameter σ. For the OKAPI retrieval function, we study the parameters k_1 and b. We also study the impact of the number of past queries retrieved in our log-base method.

Figure 4 shows the impact of the parameter σ for both cluster-based and log-based methods on both test sets. We vary σ from 0.05 to 0.3 with step 0.05. Figure 4 shows that the performance is not very sensitive to the parameter σ. We can always obtain the best result in range $0.1 \leq \sigma \leq 0.25$.

In Table 4, we show the impact of OKAPI parameters. We vary k_1 from 1.0 to 2.0 with step 0.2 and b from 0 to 1 with step 0.2. From this table, it is clear that P@5 is also not very sensitive to the parameter setting. Most of the values are larger than 0.35. The default values $k_1 = 1.2$ and $b = 0.8$ give approximately optimal results.

We further study the impact of the amount of history

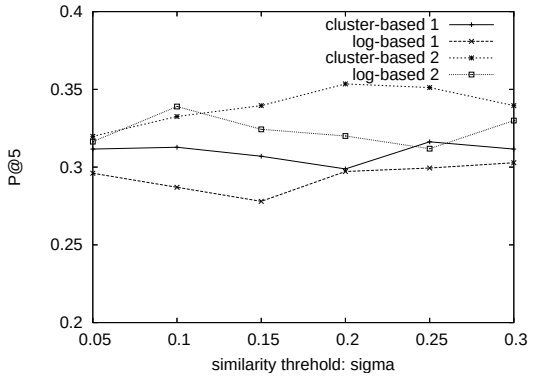

Figure 4: The impact of similarity threshold σ on both cluster-based and log-based methods. We show the result on both test collections.

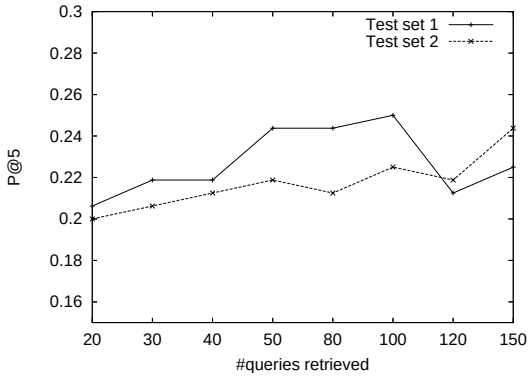

Figure 5: The impact of the number of past queries retrieved.

		b					
		0.0	0.2	0.4	0.6	0.8	1.0
k_1	1.0	0.3476	0.3406	0.3453	0.3616	0.3500	0.3453
	1.2	0.3418	0.3383	0.3453	0.3593	0.3534	0.3546
	1.4	0.3337	0.3430	0.3476	0.3604	0.3546	0.3465
	1.6	0.3476	0.3418	0.3523	0.3534	0.3581	0.3476
	1.8	0.3465	0.3418	0.3546	0.3558	0.3616	0.3476
	2.0	0.3453	0.3500	0.3534	0.3558	0.3569	0.3546

Table 4: Impact of OKAPI parameters k_1 and b.

information to learn from by varying the number of past queries to be retrieved for learning aspects. The results on both test collections are shown in Figure 5. We can see that the performance gradually increases as we enlarge the number of past queries retrieved. Thus our method could potentially learn more as we accumulate more history. More importantly, as time goes, more and more queries will have sufficient history, so we can improve more and more queries.

6.2.4 An Illustrative Example

We use the query "area codes" to show the difference in the results of the log-based method and the cluster-based method. This query may mean "phone codes" or "zip codes". Table 5 shows the representative keywords extracted from the three biggest clusters of both methods. In the cluster-based method, the results are partitioned based on locations: local or international. In the log-based method, the results are disambiguated into two senses: "phone codes" or "zip codes". While both are reasonable partitions, our evaluation indicates that most users using such a query are often interested in either "phone codes" or "zip codes." since the P@5 values of cluster-based and log-based methods are 0.2 and 0.6, respectively. Therefore our log-based method is more effective in helping users to navigate into their desired results.

Cluster-based method	Log-based method
city, state	telephone, city, international
local, area	phone, dialing
international	zip, postal

Table 5: An example showing the difference between the cluster-based method and our log-based method

6.2.5 Labeling Comparison

We now compare the labels between the cluster-based method and log-based method. The cluster-based method has to rely on the keywords extracted from the snippets to construct the label for each cluster. Our log-based method can avoid this difficulty by taking advantage of queries. Specifically, for the cluster-based method, we count the frequency of a keyword appearing in a cluster and use the most frequent keywords as the cluster label. For log-based method, we use the *center* of each star cluster as the label for the corresponding cluster.

In general, it is not easy to quantify the readability of a cluster label automatically. We use examples to show the difference between the cluster-based and the log-based methods. In Table 6, we list the labels of the top 5 clusters for two examples "jaguar" and "apple". For the cluster-based method, we separate keywords by commas since they do not form a phrase. From this table, we can see that our log-based method gives more readable labels because it generates labels based on users' queries. This is another advantage of our way of organizing search results over the clustering approach.

Label comparison for query "jaguar"			
Log-based method		Cluster-based method	
1.	jaguar animal	1.	jaguar, auto, accessories
2.	jaguar auto accessories	2.	jaguar, type, prices
3.	jaguar cats	3.	jaguar, panthera, cats
4.	jaguar repair	4.	jaguar, services, boston
5.	jaguar animal pictures	5.	jaguar, collection, apparel
Label comparison for query "apple"			
Log-based method		Cluster-based method	
1.	apple computer	1.	apple, support, product
2.	apple ipod	2.	apple, site, computer
3.	apple crisp recipe	3.	apple, world, visit
4.	fresh apple cake	4.	apple, ipod, amazon
5.	apple laptop	5.	apple, products, news

Table 6: Cluster label comparison.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we studied the problem of organizing search results in a user-oriented manner. To attain this goal, we rely on search engine logs to learn interesting aspects from users' perspective. Given a query, we retrieve its related

queries from past query history, learn the aspects by clustering the past queries and the associated clickthrough information, and categorize the search results into the aspects learned. We compared our log-based method with the traditional cluster-based method and the baseline of search engine ranking. The experiments show that our log-based method can consistently outperform cluster-based method and improve over the ranking baseline, especially when the queries are difficult or the search results are diverse. Furthermore, our log-based method can generate more meaningful aspect labels than the cluster labels generated based on search results when we cluster search results.

There are several interesting directions for further extending our work: First, although our experiment results have clearly shown promise of the idea of learning from search logs to organize search results, the methods we have experimented with are relatively simple. It would be interesting to explore other potentially more effective methods. In particular, we hope to develop probabilistic models for learning aspects and organizing results simultaneously. Second, with the proposed way of organizing search results, we can expect to obtain informative feedback information from a user (e.g., the aspect chosen by a user to view). It would thus be interesting to study how to further improve the organization of the results based on such feedback information. Finally, we can combine a general search log with any personal search log to customize and optimize the organization of search results for each individual user.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. This work is in part supported by a Microsoft Live Labs Research Grant, a Google Research Grant, and an NSF CAREER grant IIS-0347933.

9. REFERENCES

[1] E. Agichtein, E. Brill, and S. T. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, pages 19–26, 2006.

[2] J. A. Aslam, E. Pelekov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applicatins*, 8(1):95–129, 2004.

[3] R. A. Baeza-Yates. Applications of web query mining. In *ECIR*, pages 7–22, 2005.

[4] D. Beeferman and A. L. Berger. Agglomerative clustering of a search engine query log. In *KDD*, pages 407–416, 2000.

[5] D. Carmel, E. Yom-Tov, A. Darlow, and D. Pelleg. What makes a query difficult? In *SIGIR*, pages 390–397, 2006.

[6] H. Chen and S. T. Dumais. Bringing order to the web: automatically categorizing search results. In *CHI*, pages 145–152, 2000.

[7] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of ACM SIGIR 2002*, pages 299–306, 2002.

[8] S. T. Dumais, E. Cutrell, and H. Chen. Optimizing search by showing results in context. In *CHI*, pages 277–284, 2001.

[9] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *SIGIR*, pages 76–84, 1996.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[11] T. Joachims. *Evaluating Retrieval Performance Using Clickthrough Data.*, pages 79–96. Physica/Springer Verlag, 2003. in J. Franke and G. Nakhaeizadeh and I. Renz, "Text Mining".

[12] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

[13] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW*, pages 658–665, 2004.

[14] Microsoft Live Labs. Accelerating search in academic research, 2006. http://research.microsoft.com/ur/us/fundingopps/RFPs/Search_2006_RFP.aspx.

[15] P. Pirolli, P. K. Schank, M. A. Hearst, and C. Diehl. Scatter/gather browsing communicates the topic structure of a very large text collection. In *CHI*, pages 213–220, 1996.

[16] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *KDD*, pages 239–248, 2005.

[17] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR*, pages 232–241, 1994.

[18] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[19] X. Shen, B. Tan, and C. Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR*, pages 43–50, 2005.

[20] C. J. van Rijsbergen. *Information Retrieval, second edition.* Butterworths, London, 1979.

[21] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, Berlin, 1995.

[22] Vivisimo. http://vivisimo.com/.

[23] X. Wang, J.-T. Sun, Z. Chen, and C. Zhai. Latent semantic analysis for multiple-type interrelated data objects. In *SIGIR*, pages 236–243, 2006.

[24] J.-R. Wen, J.-Y. Nie, and H. Zhang. Clustering user queries of a search engine. In *WWW*, pages 162–168, 2001.

[25] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *SIGIR*, pages 512–519, 2005.

[26] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *SIGIR*, pages 46–54, 1998.

[27] O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to web search results. *Computer Networks*, 31(11-16):1361–1374, 1999.

[28] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to cluster web search results. In *SIGIR*, pages 210–217, 2004.