Evaluation of Result Merging Strategies for Metasearch Engines

Yiyao Lu [1], Weiyi Meng [1], Liangcai Shu [1], Clement Yu [2], King-Lup Liu [3]

[1] Dept of Computer Science, SUNY at Binghamton, Binghamton, NY 13902
{ylu0,meng,lshu1}@Binghamton.edu
[2] Dept of Computer Science, U. of Illinois at Chicago, Chicago, IL 60607
yu@cs.uic.edu
[3] Webscalers, LLC, Lafayette, LA 70506
kliu@webscalers.com

Abstract. Result merging is a key component in a metasearch engine. Once the results from various search engines are collected, the metasearch system merges them into a single ranked list. The effectiveness of a metasearch engine is closely related to the result merging algorithm it employs. In this paper, we investigate a variety of resulting merging algorithms based on a wide range of available information about the retrieved results, from their local ranks, their titles and snippets, to the full documents of these results. The effectiveness of these algorithms is then compared experimentally based on 50 queries from the TREC Web track and 10 most popular general-purpose search engines. Our experiments yield two important results. First, simple result merging strategies can outperform Google. Second, merging based on the titles and snippets of retrieved results can outperform that based on the full documents.

1. INTRODUCTION

Metasearch engine is a system that provides unified access to multiple existing search engines. After the results returned from all used component search engines are collected, the metasearch system merges the results into a single ranked list. The major benefits of metasearch engines are their capabilities to combine the coverage of multiple search engines and to reach the Deep Web. A recent survey [2] indicates that the Web has about 550 billion pages and only about 1% of them are in the Surface Web while the rest are in the Deep Web. The coverage of each search engine is limited. For example, *Google*, has indexed about 8 billion pages currently. This provides the basis for increasing search coverage via combining multiple search engines. Documents in the Deep Web are not directly crawlable but are searchable through special search interfaces. Since metasearch engine utilizes other search systems, connecting to multiple Deep Web search engines is an effective mechanism to reach the Deep Web.

Sophisticated metasearch engines also perform *database selection*, which is to identify the most suitable component search engines to invoke for a given user query. If a metasearch engine has a large number of component search engines, then it is very inefficient to pass each user query to every component search engine due to the additional time needed to invoke each search engine and to process the more results. In this

case, database selection should be performed. In order to perform database selection, some information that can represent the main contents of each search engine is collected in advance. When a user query is received, the representative information is used to estimate the *usefulness* of each search engine with respect to this query, which is then used to determine if a search engine should be used to process this query. The above usefulness will be called *search engine score* in this paper. Many methods for database selection exist (see [12] for a survey of some methods) but will not be discussed in this paper. Search engine scores are used by some result merging techniques.

A straightforward way to perform result merging is to fetch the retrieved documents to the metasearch engine site and then to compute their similarities with the query using a global similarity function. This method is used in the *Inquirus* metasearch engine [10] as well in the approach in [20]. The main problem of this approach is that the user has to wait a long time before the results can be fully displayed. Therefore, most result merging techniques utilize the information associated with the search results as returned by component search engines to perform merging [12]. The difficulty lies in the heterogeneities among the component search engines. For example, some component search engines may return a local ranking score (some kind of similarity value) for each result while some don't. As another example, different search engines would rank the same set of documents differently since they adopt different ranking formulas and term weighting schemes.

In this paper, we experimentally evaluate and compare seven different result merging algorithms, five of them are either newly proposed in this paper or revised from existing ones. The main contributions of this paper are two folds. First, it proposes several new algorithms that reflect some fresh ideas about result merging. Second, it provides extensive experimental results to evaluate the effectiveness of newly proposed algorithms and compare them with the existing methods. Two important conclusions are observed from our experimental results. Utilizing the titles and snippets associated with search results can achieve better effectiveness than using the full documents. Moreover, a good result merging algorithm can help a metasearch engine significantly outperform the best single search engine in effectiveness.

The rest of the paper is organized as follows. Section 2 reviews various existing result merging approaches. In Section 3, we present five new result merging algorithms. Section 4 provides the experimental results. Section 5 concludes the paper.

2. RELATED WORK

In the 1990s, most search engines reported the ranking scores of their search results. Therefore, many earlier techniques focused on normalizing the scores to make them more comparable across different search engines (e.g., [4, 16]). By this way, the results retrieved from different systems can be uniformly ranked by the normalized scores.

Nowadays, very few search engines report ranking scores, but it is still possible to convert the local ranks into local ranking scores. Borda Count [1] is a voting-based data fusion method applicable in the context of metasearch. The returned results are considered as the candidates and each component search engine is a voter. For each voter, the top ranked candidate is assigned n points (assuming there are n candidates), the

second top ranked candidate is given $n - 1$ points, and so on. For candidates that are not ranked by a voter (i.e., they are not retrieved by the corresponding search engine), the remaining points of the voter (each voter has a fixed number of points) will be divided evenly among them. The candidates are then ranked in descending order of the total points they receive. Several more complex schemes using ranks to merge results are reported in [5] but they need a larger number of results (say 100) to be retrieved from each search engine and they need substantial overlap among the retrieved results from different search engines to work effectively.

In D-WISE [21], the local rank of a document (r_i) returned from search engine j is converted to a ranking score (rs_{ij}) by using the formula:

$$rs_{ij} = 1 - (r_i - 1) * S_{\min} / (m * S_j) \quad , \tag{1}$$

where S_j is the usefulness score of the search engine j, $S_{\min}$ is the smallest search engine score among all component search engines selected for this query, and m is the number of documents desired across all search engines. This function generates a smaller difference between the ranking scores of two consecutively ranked results retrieved from a search engine with a higher search engine score. This has the effect of ranking more results from higher quality search engines (with respect to the given query) higher. One problem of this method is that the highest ranked documents returned from all the local systems will have the same ranking score 1.

Some approaches also take the search engine scores into consideration (e.g., [3, 7]). For example, the basic Borda Count method can be improved by adjusting the ranking score of a result by multiplying the points it receives with its source search engine's usefulness score so that the results from more useful search engines for the query are more likely to be ranked higher. However, estimating the search engine score usually requires some sample data to be collected from each component search engine beforehand (e.g., [1, 13, 17]). Thus, this type of approaches cannot be applied to situations where a search engine joins a metasearch engine on the fly, such as in the case where customized metasearch engines are created on demand [19]. In our work, we focus on the result merging approaches without collecting any sample data in advance.

Most of the current generation search engines present more informative *search result records* (SRRs) of retrieved results to the user. A typical SRR consists of the URL, title and a summary (snippet) of the retrieved document. As a result, the contents associated with the SRRs can be used to rank/merge results retrieved from different search engines. We are aware of only two works that utilize such SRRs. The work described in [8] is more about determining if a SRR contains enough information for merging so that the corresponding full document need not be fetched rather than a merging technique. The work reported in [13] is the most similar to our work in this paper. The available evidences that can be used for result merging are identified, such as the document title, snippet, local rank, search engine usefulness, etc. The algorithms based on different combinations of these evidences are proposed and their effectiveness is compared. However, their work is significantly different from ours in several aspects. First, while their metasearch system uses news search engines, we concentrate on general-purpose search engines. This makes our work more relevant to the current major metasearch engines as nearly all of them (e.g., Mamma.com, Dogpile.com, Search.com) use general-purpose search engines as their underlying components.

Second, besides the title, snippet and the local rank of each retrieved document, some of our algorithms also consider the frequencies of query terms in each SRR, the order and the closeness of these terms, etc (see Section 3.2). On the other hand, publication date of each result is utilized in the algorithms in [13] but is not in ours because time is not as sensitive in the general web documents as in the news articles. Third, the datasets used are different. In [13], the dataset consists of the queries that are generated specifically for that work and the news items retrieved from the selected news search engines using these queries, while the dataset in our work consists of the queries used for the TREC Web Track and the documents retrieved from the 10 major general-purpose search engines using these queries. Fourth, titles and snippets of SRRs are used differently (different similarity functions are used) is these two approaches. Our experimental results indicate that our approach outperforms the best approach in [13] (see Section 4 for details).

Among all the proposed merging methods in [13], the most effective one is based on the combination of the evidences of document title, snippet, and the search engine usefulness. This method works as follows. First of all, for each document, the similarity between the query and its title, and the similarity between the query and its snippet are computed. Then the two similarities are linearly aggregated as this document's estimated global similarity. For each query term, its weight in every component search engine is computed based on the *Okapi* probabilistic model [14]. The *Okapi model* requires the information of document frequency (*df*) of each term. Since the *df* information cannot be obtained in a metasearch engine context, the *df* of the term *t* in search engine *j* is approximated by the number of documents in the top 10 documents returned by search engine *j* containing term *t* within their titles and snippets. The search engine score is the sum of all the query term weights of this search engine. Finally, the estimated global similarity of each result is adjusted by multiplying the relative deviation of its source search engine's score to the mean of all the search engine scores.

Major general purpose search engines have a certain amount of overlaps between them. It is very possible that for a given query, the same document is returned from multiple component search engines. In this case, their (normalized) ranking scores need to be combined. A number of fusion functions have been proposed to solve this problem and they include *min*, *max*, *sum*, *average*, *CombMNZ*, and other linear combination functions [6, 11, 18].

3. MERGING ALGORITHMS

Algorithm 1: Use Top Document to Compute Search Engine Score (TopD)

This algorithm can be considered as a variation of the method described in [21] (see Formula 1). Let S_j denote the score of search engine j with respect to Q. The method in [21] needs the document frequency of every term to be collected in advance, while the *TopD* algorithm uses the similarity between Q and the top ranked document returned from search engine j (denoted d_{1j}) to estimate S_j. The rationale is that, in general, the highest ranked document is the most relevant to the user query based on the search engine's ranking criteria. Its content can reflect how "good" the search engine is with

respect to the user query. Fetching the top ranked document from its local server will introduce some extra network delay to the merging process, but we believe that this delay is tolerable since only one document is fetched from each used search engine for a query

For the similarity function, we tried both the *Cosine function* [15] and the *Okapi function* [14]. In *Cosine function*, the weight associated with each term in Q and d_{1j} is the *tf* weight [15] (we also tried *tf*idf* weight and the results are similar). The similarity between query Q and d_{1j} using *Okapi function* is the sum of the *Okapi weight* of each query term T. The formula is:

$$\sum_{T \in Q} w * \frac{(k_1 + 1) * tf}{K + tf} * \frac{(k_3 + 1) * qtf}{k_3 + qtf}, \quad (2)$$

$$\text{with } w = \log \frac{N - n + 0.5}{n + 0.5} \text{ and } K = k_1 * ((1 - b) + b * \frac{dl}{avgdl}) ,$$

where *tf* is the frequency of the query term T within the processed document, *qtf* is the frequency of T within the query, N is the number of documents in the collection, n is the number of documents containing T, *dl* is the length of the document, and *avgdl* is the average length of all the documents in the collection. k_1, k_3 and b are the constants with values 1.2, 1,000 and 0.75, respectively. Since N, n, and *avgdl* are unknown, we use some approximations to estimate them. For *avgdl*, we used the average length of the documents we collected in our testbed (see Section 4.1), where the value is 1424.5 (words). We use Google's size to simulate N = 8,058,044,651. For each query term T in our testing queries, we submit it as a single-term query to *Google* and retrieve the number of documents returned as the value of n.

As mentioned before, the ranking scores of the top ranked results from all used search engines will be 1 by using Formula 1. We remedy this problem by computing an adjusted ranking score ars_{ij} by multiplying the ranking score computed by Formula 1, namely rs_{ij}, by S_j. If a document is retrieved from multiple search engines, we compute its final ranking score by summing up all the adjusted ranking scores.

Algorithm 2: Use Top SRRs to Compute Search Engine Score (TopSRR)

This algorithm is the same as the *TopD* algorithm except that a different method is used to compute the search engine score. When a query Q is submitted to a search engine (say search engine j), the search engine returns the search result records (SRRs) of a certain number of top ranked documents on a dynamically generated result page. In the *TopSRR* algorithm, the SRRs of the top n returned results from each search engine, instead of the top ranked document, are used to estimate its search engine score. Intuitively, this is reasonable as a more useful search engine for a given query is more likely to retrieve better results which are usually reflected in the SRRs of these results. Specifically, all the titles of the top n SRRs from search engine j are merged together to form a *title vector* TV_j, and all the snippets are also merged into a *snippet vector* SV_j. The similarities between query Q and TV_j, and between Q and SV_j are computed separately and then aggregated into the score of search engine j:

$$S_j = c_1 * Similarity(Q,\ TV_j) + (1 - c_1) * Similarity(Q,\ SV_j)\ \ , \qquad (3)$$

where c_1 = 0.5 and n = 10 are currently used in our work. Again, both the *Cosine similarity function* with *tf* weight and the *Okapi function* are used. In the *Okapi function*, the average document lengths (*avgdl*) of the title vector TV_j and the snippet vector SV_j are estimated by the average length of the titles and the snippets of the top 10 results on the result page. The values are 46.2 (words) and 163.6 (words), respectively.

Algorithm 3: Compute Simple Similarities between SRRs and Query (SRRSim)

Since each SRR can be considered as the representative of the corresponding full document, we may rank SRRs returned from different search engines based on their similarities with the query directly using an appropriate similarity function.

In the *SRRSim* algorithm, the similarity between a SRR R and a query Q is defined as a weighted sum of the similarity between the title T of R and Q and the similarity between the snippet S of R and Q:

$$sim(R, Q) = c_2 * Similarity(Q,\ T) + (1 - c_2) * Similarity(Q,\ S)\ \ , \qquad (4)$$

where, in the current implementation, c_2 = 0.5. Again both the *Cosine similarity function* with *tf* weight and the *Okapi function* are tried. The *avgdl* of title and snippet used in the *Okapi* weight is set to be 5.0 and 17.6, respectively, which are estimated based on the SRRs of the collected documents in our testbed. (The reason that the average lengths of individual titles and snippets are longer than those can be derived from the numbers given in Algorithm 2, i.e., 5 > 46.2/10 and 17.6 > 163.6/10, is because some result pages contain less than 10 results.) If a document is retrieved from multiple search engines with different SRRs (different search engines usually employ different ways to generate SRRs), then the similarity between the query and each such SRR will be computed and the largest one will be used as the final similarity of this document with the query for result merging.

Algorithm 4: Rank SRRs Using More Features (SRRRank)

The similarity function used in the *SRRSim* algorithm, no matter it is the *Cosine function* or the *Okapi function*, may not be sufficiently powerful in reflecting the true matches of the SRRs with respect to a given query. For example, these functions do not take proximity information such as how close the query terms occur in the title and snippet of a SRR into consideration, nor does it consider the order of appearances of the query terms in the title and snippet. Intuitively, if a query contains one or more phrases, the order and proximity information has a significant impact on the match of phrases versus just individual terms. As an example, suppose a user query contains two terms t1 and t2 in this order. Two documents d1 and d2 have the same length and they have the same numbers of t1 and t2 in it. In d1, t1 and t2 always occur together and t1 always appears in front of t2 while t1 and t2 are scattered all over in d2. The *Cosine function* or

Okapi function will give d1 and d2 the same similarity value. However, intuitively, d1 should be a better match to the query than d2.

To better rank SRRs, we define five features with respect to the query terms. First, the number of distinct query terms appearing in the title and the snippet (NDT). Second, the total number occurrences of the query terms in the title and the snippet (TNT). These two features indicate the overlapping level between the query and the SRR. Generally speaking, the larger the overlap, the more likely they are relevant. Third, the locations of the occurred query terms (TLoc). There are three cases, all in title, all in snippet, and scattered in both title and snippet. This feature describes the distribution of the query terms in the SRR. In real applications, the title is more frequently associated with a returned result than the snippet (some search engines provide titles only). Therefore, title is usually given higher priority than the snippet. Fourth, whether the occurred query terms appear in the same order as they are in the query and whether they occur adjacently (ADJ). And finally, the window size containing distinct occurred query terms (WS). If all the distinct occurred query terms are located in the title or the snippet, it is the smallest number of consecutive words in the title or snippet that contains at least one occurrence of each occurred distinct query term; otherwise, the window size is infinite. The last two features represent how close the query terms appear in the SRR. Intuitively, the closer those terms appear in the SRR, the more likely they have the same meaning as they are in the query.

For each SRR of the returned result, the above pieces of information are collected. The *SRRRank* algorithm works as follows: first, all the SRRs are grouped based on the number of distinct query terms (NDT) in their title and snippet fields. The groups having more distinct terms are ranked higher. Second, within each group, the SRRs are further put into three sub-groups based on the location of the occurred distinct query terms (TLoc). The sub-group with these terms in the title ranks highest, and then the sub-group with the distinct terms in the snippet, and finally the sub-group with the terms scattered in both title and snippet. Finally, within each sub-group, the SRRs that have more occurrences of query terms (TNT) appearing in the title and the snippet are ranked higher. If two SRRs have the same number of occurrences of query terms, first the one with distinct query terms appearing in the same order and adjacently (ADJ) as they are in the query is ranked higher, and then, the one with smaller window size (WS) is ranked higher. After the above steps, if there is any tie, it is broken by the local ranks. The result with the higher local rank will have a higher global rank in the merged list. If a result is retrieved from multiple search engines, we only keep the one with the highest global rank.

Algorithm 5: Compute Similarities between SRRs and Query Using More Features (SRRSimMF)

This algorithm is similar to *SRRRank* except that it quantifies the matches based on each feature identified in *SRRRank* so that the matching scores based on different features can be aggregated into a numeric value. Consider a given field of a SRR, say title (the same methods apply to snippet). For the number of distinct query terms (NDT), its matching score is the ratio of NDT over the total number of distinct terms in the query (QLEN), denoted $S_{NDT} = NDT / QLEN$. For the total number of query terms

(TNT), its matching score is the ratio of TNT over the length of title (i.e., the number of terms in the title), denoted $S_{TNT} = TDT / TITLEN$. For the query terms order and adjacency information (ADJ), the matching score S_{ADJ} is set to 1 if the distinct query terms appear in the same order and adjacently in the title; otherwise the value is 0. The window size (WS) of the distinct query terms in the processed title is converted into score $S_{WS} = (TITLEN - WS) / TITLEN$ (smaller WS leads to larger score). All the matching scores of these features are aggregated into a single value, which is the similarity between the processed title T and Q, using the following formula:

$$Sim(T,Q) = S_{NDT} + \frac{1}{QLEN} * (W_1 * S_{ADJ} + W_2 * S_{WS} + W_3 * S_{TNT}) \quad (5)$$

This formula guarantees that titles containing more distinct query terms will have larger similarities.

For each SRR, the similarity between the title and the query ($Sim(T, Q)$) and the similarity between the snippet S and the query ($Sim(S, Q)$) are computed separately first and then merged into one value as follows:

$$Similarity = \frac{TNDT}{QLEN} * (c_3 * Sim(T,Q) + (1 - c_3) * Sim(S,Q)) \quad (6)$$

where $TNDT$ is the total number of distinct query terms appeared in title and snippet. By multiplying by $TNDT / QLEN$, we guarantee that the SRR containing more distinct query terms will be ranked higher.

A genetic algorithm based training method is used to determine the values of the parameters involved in this method. Among the testing queries, the odd numbered queries are used for the training. The optimal values of W_1, W_2, W_3 and c_3 found by the training are 0, 0.14, 0.41 and 0.2, respectively. $W_1 = 0$ means that the order and adjacency information is not useful for improving result merging in Algorithm 5. One possible explanation for this is that due to the small length of each title and snippet, the terms are already sufficiently close to each other to identify their meanings.

4. EXPERIMENTS

4.1. Testbed

The purpose of this work is to evaluate and compare different result merging algorithms under the context of metasearch over the general-purpose search engines. So we select 10 most popular general-purpose search engines as the underlying component search engine. They are: *Google*, *Yahoo*, *MSN*, *Askjeeves*, *Lycos*, *Open Directory*, *Altavista*, *Gigablast*, *Wisenut*, and *Overture*. The reasons these search engines are selected are: (1) they are used by nearly all the popular general-purpose metasearch engines; (2) each of them has indexed a relatively large number of web pages; and (3) they adopt different ranking schemes. Even though we focus our work in the context of

general-purpose search engines, the result merging algorithms we proposed in this paper are completely independent of the search engine type.

2002 TREC Web Track topics are used as the queries to collect the testbed from the selected search engines. Each web topic contains 4 parts: an index number, a title, a description and a narrative (see Figure 1 for an example). 2002 TREC Web Track has 50 topics indexed from 551 to 600. In this paper, for each topic, only the title part is used as a query to send to the search engines, because the titles are short, similar to most Internet queries submitted by real users. The average length of the titles of these 50 topics is 3.06. The description and the narrative describe what documents should be considered relevant to the corresponding topic. This information is served as the standard criteria for us to judge the relevancy of the collected result documents.

<num> Number: 551

<title> intellectual property

<desc> Description:

Find documents related to laws or regulations that protect intellectual property.

<narr> Narrative:

Relevant documents describe legislation or federal regulations that protect authors or composers from copyright infringement, or from piracy of their creative work. These regulations may also be related to fair use or to encryption.

Fig. 1. An Example TREC Query

Each query is submitted to every component search engine. For each query and each search engine, the top 10 results on the first result page are collected (some search engines may return less than 10 results for certain queries). Totally there are 4,642 result documents, excluding 42 broken links. This number corresponds to approximately 9.3 documents per query and per search engine. Information associated with each returned record is collected, including the URL, title, snippet and the local rank. Besides, the document itself is downloaded. The relevancy of each document is manually checked based on the criteria specified in the description and the narrative part of the corresponding TREC query. The collected data and the documents, together with the relevancy assessment result, form our testbed. The testbed is stored locally so it will not be affected by any subsequent changes from any component search engine.

4.2 Evaluation Criteria

Because it is difficult to know all the relevant documents to a query in a search engine, the traditional *recall* and *precision* for evaluating IR systems cannot be used for evaluating search/metasearch engines. A popular measure for evaluating the effectiveness of search engines is the *TREC-style average precision* (TSAP) [10]. In this paper, TSAP at cutoff N, denoted as *TSAP@N*, will be used to evaluate the effectiveness of each merging algorithm:

$$TSAP@N = (\sum_{i=1}^{N} r_i) / N \quad , \tag{7}$$

where $r_i = 1/i$ if the i-th ranked result is relevant and $r_i = 0$ if the i-th result is not relevant. It is easy to see that *TSAP@N* takes into consideration both the number of relevant documents in the top N results and the ranks of the relevant documents. *TSAP@N* tends to yield a larger value when more relevant documents appear in the top N results and when the relevant documents are ranked higher. For each merging algorithm, the average *TSAP@N* over all 50 queries is computed and is used to compare with other merging algorithms.

4.3 Result Analysis

We first evaluated the average precision of each of the 10 component search engines used in our metasearching system. The results are reported in Table 1. Since for each query, we only collect the top 10 documents on the first result page from each search engine, we compute the effectiveness of each search engine at two N levels only, i.e., N = 5 and 10. It is easy to see that *Google* is the best performer, with *Altavista* and *Yahoo* close behind and others significantly behind.

Table 1. Retrieval Effectiveness of Component Search Engines

Seach Engine	TSAP@N	
	N = 5	N = 10
Google	*0.316*	*0.199*
Yahoo	0.308	0.194
MSN	0.265	0.164
Askjeeves	0.229	0.148
Lycos	0.224	0.145
Open Directory	0.091	0.051
Altavista	0.315	0.199
Gigablast	0.248	0.155
Wisenut	0.289	0.177
Overture	0.161	0.109

Six algorithms are compared and the results are listed in the Table 2. In addition to the five algorithms described in Section 3, the full document fetching method (**DocFetching**, see Section 1) is also included. Both the Cosine function and the Okapi function are considered. For each algorithm to be evaluated, we compute its *TSAP* at different N levels, where N = 5, 10, 20, and 30.

Table 2 shows that *TopD* and *TopSRR* algorithms are the least effective among all the merging algorithms evaluated. Their performances are close to that of the best single search engine (i.e., *Google*) for N = 5 and N = 10. This suggests that the local ranks of retrieved results do not contain enough information to achieve good merging.

We also find a large margin of differences in effectiveness between *Google* and the other four algorithms, including full document fetching method (Okapi). Table 3 shows

the effectiveness of three merging algorithms, *SRRSim*, *SRRRank*, *SRRSimMF,* when Google is not a component search engine (i.e., only the other 9 search engines are used). The exclusion of Google did not noticeably impact the performance, probably because the results from *Google* are also retrieved by other search engines collectively. This experimental result shows that, while it is widely recognized that metasearch engines can increase search coverage, metasearching can improve the search effectiveness over a single search engine as well. One reason why metasearching may achieve better effectiveness is because different individual search engines often yield different relevant documents and a good result merging algorithm can rank more of these relevant documents higher than any single search engine.

Table 2. Retrieval Effectiveness Comparison

Algorithm		TSAP@N			
		N = 5	N = 10	N = 20	N = 30
TopD	Cosine	0.297	0.185	0.112	0.081
	Okapi	0.318	0.198	0.116	0.084
TopSRR	Cosine	0.314	0.194	0.112	0.084
	Okapi	0.311	0.193	0.115	0.084
SRRSim	Cosine	0.366	0.228	0.137	0.098
	Okapi	0.377	0.235	0.140	0.100
SRRRank		0.371	0.230	0.135	0.098
SRRSimMF		0.381	0.238	0.140	0.100
DocFetching	Cosine	0.271	0.177	0.108	0.080
	Okapi	0.338	0.217	0.131	0.094

Table 3. Effectiveness without Using Google

Algorithm	TSAP@N			
	N=5	N=10	N=20	N=30
SRRSim(Okapi)	0.381	0.236	0.139	0.099
SRRRank	0.370	0.230	0.134	0.097
SRRSimMF	0.381	0.237	0.139	0.099

Algorithms that perform merging based on the contents of SRRs (i.e., *SRRSim*, *SRRRank*, *SRRSimMF*) are the top performers among all the merging algorithms. They all outperform *Google* and the document fetching method significantly. It shows that the titles and snippets of SRRs contain good information about the contents of the corresponding documents. Overall, high quality snippets are generated by major general-purpose search engines. Moreover, because good merging algorithms using SRRs only can significantly outperform the document fetching based method, there is no need to fetch the full documents of the retrieved results for result merging. This means that by using SRRs for merging, not only can the merging process be sped up significantly, the effectiveness can also be improved, compared with using the full documents.

It is somewhat surprising that using the full documents yielded lower effectiveness than using the SRRs, but a careful analysis reveals a valid explanation for this phenomenon. Even though it is true that a full document has more information than its

SRR, the information is not fully utilized by most similarity functions that are employed, including the *Cosine function* and the *Okapi function*. For example, neither of these two functions considers the proximity information of the query terms in a document. On the other hand, since titles and snippets are much shorter than the full documents, when they are used for similarity computations with a query, the proximity information is automatically taken into consideration. Furthermore, the search results returned by component search engines are obtained based on the full documents. In other words, the full documents have been used to gather the initial result sets and their SRRs are used for a second round of selection.

The best algorithm reported in [13] (**Rasolofo**) is also a SRR-based method. The main idea of this algorithm is summarized in Section 2. It is similar to Algorithm 3 (*SRRSim*) and the main differences between them are: (1) Rasolofo's algorithm uses search engine scores (which are computed based on the *Okapi model*) to adjust the similarity for each SRR while *SRRSim* does not do that. (2) Rasolofo's algorithm and *SRRSim* use different similarity functions to compute the similarity between a title/snippet and a query. While *SRRSim* uses the *Okapi function*, Rasolofo's algorithm uses $100000 * NQW \Big/ \sqrt{|Q|^2 + |F|^2}$, where *NQW* is the number of query words that appear in the processed field (e.g., title or snippet), and |Q| and |F| are the lengths of the query and the processed field, respectively; if the above similarity is zero for a result, it is substituted by the *rank score* of the result, which is defined to be 1000 – *R*, where *R* is the rank of the result given by the local search engine. We compared our three SRR-based algorithms, namely, *SRRSim*, *SRRank* and *SRRSimMF*, with Rasolofo. Since this method requires the information about the number of documents returned by every local search engine for each query to compute its search engine score and *Overture* does not provide this information, the comparison with *Rasolofo* is based on the results returned from the other 9 search engines. The results are reported in Table 4. It can be seen that all our algorithms outperform the *Rasolofo*'s algorithm. There are two possible reasons for this. First, the use of search engine scores to influence the ranking of SRRs may not be effective when high quality SRRs are generated by component search engines (more experiments are needed to confirm this conjecture). Second, the similarity function employed in *Rasolofo*'s algorithm for computing the similarities of titles and snippets with the query is not as good as the similarity functions or ranking method employed by *SRRSim* (*Okapi*), *SRRRank* and *SRRSimMF*. Our query-by-query comparison between *SRRSim* and the *Rasolofo*'s algorithm shows that *SRRSim* is better for 24 queries, worse for 14 queries and tied with *Rasolofo*'s algorithm for 12 queries.

Table 4. Comparison with Rasolofo's Approach

Algorithm	TSAP@N			
	N=5	N=10	N=20	N=30
SRRSim(Okapi)	0.377	0.235	0.140	0.101
SRRRank	0.372	0.230	0.136	0.098
SRRSimMF	0.380	0.237	0.139	0.100
Rasolofo	0.347	0.217	0.131	0.095

By comparing among the algorithms based on the contents of SRR, we found that the sophisticated ranking schemes employed by *SRRRank* and *SRRSimMF* fail to pay clear dividend over a much simpler scheme employed by *SRRSim* (*Okapi*). A possible reason is that because titles and snippets are already sufficiently short in terms of relating the meanings of the query terms in the SRRs, which makes the additional fine-tuning unnecessary.

5. CONCLUSIONS

In this paper, we reported our study on how to merge the search results returned from multiple component search engines into a single ranked list. This is an important problem in metasearch engine research. An effective and efficient result merging strategy is essential for developing effective metasearch systems. We experimentally compared 7 merging algorithms that utilize a wide range of information available for merging, from local ranks by component search engines, search engine scores, titles and snippets of search result records to the full documents. Ten popular general-purpose search engines and 50 TREC Web Track queries were used to perform the evaluation.

Our experimental evaluations yielded several interesting results. First, simple, efficient and easily implementable merging algorithms can outperform the best single search engine. This should help the cause of metasearch engine researchers/developers. Second, merging based on the titles and snippets of returned search result records can be more effective than using the full documents of these results. This implies that a metasearch engine can achieve better performance than a centralized retrieval system that contains all the documents from the component search engines. Third, a simply result merging algorithm can perform as well as more sophisticated ones. For example, the algorithm SRRSim that performs merging based on the weighted sum of the Okapi similarities of the title and snippet of each result with the query is as good as algorithms that take into consideration the order and proximity of query terms in the results.

A possible reason why the proximity and adjacency information of query terms in the title and snippet did not help improve effectiveness is due to the failure to distinguish different types of adjacency/proximity conditions. For example, named entities and other types of phrases (noun phrases, dictionary phrase, complex phrases, etc.) may need to be identified and dealt with differently. We plan to investigate this issue further in the near future.

Acknowledgement: This work is supported by the following grants from NSF: IIS-0208574, IIS-0208434, IIS-0414981, IIS-0414939 and CNS-0454298.

REFERENCES

1. J. Aslam, M. Montague. Models for Metasearch. ACM SIGIR Conference, 2001, pp.276-284.

2. M. Bergman. The Deep Web: Surfing Hidden Values. White Paper of CompletePlanet. 2001. (Available at http://brightplanet.com/pdf/deepwebwhitepaper.pdf).
3. J. Callan, Z. Lu, W. Croft. Searching Distributed Collections with Inference Networks. ACM SIGIR Conference, 1995, pp. 21-28.
4. D. Dreilinger, A. Howe. Experiences with Selecting Search Engines Using Metasearch. ACM TOIS, 15(3), July 1997, pp.195-222.
5. C. Dwork, R. Kumar, M. Naor, D. Sivakumar. Rank Aggregation Methods for the Web. Tenth International World Wide Web Conference, pp.613-622, 2001.
6. E. Fox, J. Shaw. Combination of Multiple Searches. Second Text Retrieval Conference, Gaithersburg, Maryland, August 1994, pp. 243-252.
7. S. Gauch, G. Wang, and M. Gomez. ProFusion: Intelligent Fusion from Multiple, Distributed Search Engines. Journal of Universal Computer Science, 2(9), pp.637-649, 1996.
8. E. Glover and S. Lawrence. Selective Retrieval Metasearch engine. US Patent Application Publication (US 2002/0165860 A1), November 2002.
9. D. Hawking, N. Craswell, P. Bailey, K. Griffiths. Measuring Search Engine Quality. Information Retrieval Journal, 4(1): 33-59 (2001).
10. S. Lawrence, and C. Lee Giles. Inquirus, the NECi Meta Search Engine. Seventh International World Wide Web Conference, 1998.
11. J. Lee. Analyses of Multiple Evidence Combination. ACM SIGIR Conference, 1997
12. W. Meng, C. Yu, K. Liu. Building Efficient and Effective Metasearch Engines. ACM Computing Surveys, 34(1), March 2002, pp.48-84.
13. Y. Rasolofo, D. Hawking, J. Savoy. Result Merging Strategies for a Current News Metasearcher. Inf. Process. Manage, 39(4), 2003, pp.581-609.
14. S. Robertson, S. Walker, M. Beaulieu. Okapi at trec-7: automatic ad hoc, filtering, vlc, and interactive track. 7th Text REtrieval Conference, 1999, pp.253-264.
15. G. Salton and M. McGill. Introduction to Modern Information Retrieval. Mc-Graw Hill, 1983.
16. E. Selberg, and O. Etzioni. The MetaCrawler Architecture for Resource Aggregation on the Web. IEEE Expert, 1997.
17. L. Si, J. Callan. Using Sampled Data and Regression to Merge Search Engine Results. ACM SIGIR Conference, 2002, pp.19-26.
18. C. Vogt, G. Cottrell. Fusion via a Linear Combination of Scores. Information Retrieval, 1(3), 1999, pp.151-173.
19. Z. Wu, V. Raghavan, C. Du, M. Sai C, W. Meng, H. He, and C. Yu. SE-LEGO: Creating Metasearch Engine on Demand. ACM SIGIR Conference, Demo paper, pp.464, 2003.
20. C. Yu, W. Meng, K. Liu, W. Wu, N. Rishe. Efficient and Effective Metasearch for a Large Number of Text Databases. Eighth ACM CIKM Conference, 1999, pp. 217-224.
21. B. Yuwono, D. Lee. Server Ranking for Distributed Text Resource Systems on the Internet. International Conference on Database System For Advanced Applications, 1997, pp.391-400.