Using Micro Information Units for Internet Search

Xiaoli Li, Bing Liu*, Tong-Heng Phang, and Minqing Hu

School of Computing
National University of Singapore
* Department of Computer Science
University of Illinois at Chicago

ABSTRACT

Internet search is one of the most important applications of the Web. A search engine takes the user's keywords to retrieve and to rank those pages that contain the keywords. One shortcoming of existing search techniques is that they do not give due consideration to the micro-structures of a Web page. A Web page is often populated with a number of small information units, which we call *micro information units* (MIU). Each unit focuses on a specific topic and occupies a specific area of the page. During the search, if all the keywords in the user query occur in a single MIU of a page, the top ranking results returned by a search engine are generally relevant and useful. However, if the query words scatter at different MIUs in a page, the pages returned can be quite irrelevant (which causes low precision). The reason for this is that although a page has information on individual MIUs, it may not have information on their intersections. In this paper, we propose a technique to solve this problem. At the off-line pre-processing stage, we segment each page to identify the MIUs in the page, and index the keywords of the page according to the MIUs in which they occur. In searching, our retrieval and ranking algorithm utilizes this additional information to return those most relevant pages. Experimental results show that this method is able to significantly improve the search precision.

1. INTRODUCTION

One of the most important applications of the Web is the search using search engines, e.g., AltaVista, Google, Yahoo, etc. These search systems allow the user to specify some keywords to retrieve those Web pages that contain the keywords. A major shortcoming of the current techniques is that they do not consider different topic areas of a page. Typically, the contents of a Web page encompass a number of related or even unrelated topics. Each topic usually occupies a separate area in the page. We call each topic area a *micro information unit* (or MIU in short). For example, a bookstore Web page selling books may include other diverse information like stock market quotations and weather forecasting. A personal homepage may contain information on different interests of its owner.

A *micro information unit* (MIU) is a coherent topic area according to its content, and it is usually also a visual block from the display point of view. If the user's query terms (or keywords) occur in a single MIU of a Web page, the pages returned by a search engine are generally relevant and useful. However, if the keywords scatter at different MIUs, it can cause low precision of the returned search results. Although many search engines are able to consider relative distances of keywords [4] (among others, e.g., word frequency, authority and hub scores, etc) in a Web page in their ranking processes, they do not consider whether these words occur in different MIUs or a single MIU.

Let us use an example to illustrate the problem. For instance, we wish to find some free downloadable videos. We issue the search query "free download video" to the search engine Google. Google returns a large number of Web pages. However, most top ranking pages do not offer any free downloadable videos. For example, the first page returned by Google does contain the three keywords "free", "download" and "video" (Figure 1). It is a site that sells software for playing audio and video. It does not have any free video for downloading. From Figure 1, we observe that "free", "download" and "video" (circled in the figure) appear in different MIUs or topic areas.

In this paper, we propose a technique to deal with the problem. The key idea is to segment each Web page to identify different micro information units or topic areas according to its HTML tags and contents. In searching, if the keywords of a query occur in the same MIU, the Web page will be given a higher ranking score. Otherwise, it will be given a lower ranking score. In the proposed technique, page segmentation and indexing according to MIUs in a Web page is done in off-line pre-processing. We show that the additional information on MIUs can be naturally integrated with inverted lists indexing commonly used by Web search engines. In on-line search, our retrieval and ranking algorithm makes use of this MIU information to sort the relevant pages. Due to seamless integration of MIUs with inverted lists, additional computation required during searching is minimum.

The proposed technique is intended to be used as an advanced search option for a search engine (which we also call the *base search engine*). That is, when the precision of the results returned by the base search engine is low, we can employ the proposed technique to re-rank the results.

To evaluate the proposed technique, we used Google as the base search engine. When the precision of the returned results by Google is low, we re-rank its top 200 pages. Experimental results (including comparison with AltaVista) show that our method is able to improve the search precision dramatically, i.e., after re-ranking the number of relevant pages at the top of the list increases significantly.

2. RELATED WORK

The key issue in Web search is how to efficiently retrieve relevant Web pages with high precision for its top ranking results. The main technique used in current search engines is keyword matching. In recent years, a number of works were reported to improve search by using additional information from Web pages. [4] presents the Google search engine, which employs link structures and anchor text in addition to the traditional factors such as word occurrence and frequency to make relevance judgments. [18] presents a similar link-based search method. [16] examines the actual pages suggested by multiple search engines and then displays the results according to the user's query. Clever search engine [6] incorporates several algorithms (i.e. HIT algorithm) that make use of hyperlink structure for discovering high-quality information on the

Figure 1: The first page from Google for the query, "free download video"

Web. [9] describes a Web searching method where the input is the URL of a page. It only uses the connectivity information to identify related pages. These works are all different from ours, as they do not segment each Web page to identify different MIUs and then use these MIUs to aid the search.

[26] presents an algorithm to efficiently retrieve information units, which are logical Web document consisting of multiple physical pages. Our micro information unit is a topic area within a physical Web page.

[5] uses the Document Object Model (or HTML tag tree) and hyperlinks for topic distillation. It segments the HTML tag tree for the purpose of computing authority and hub scores of the intermediate subtrees in relation to other pages and links. This is different from our work as we aim to find coherent topic areas of the current page using both text contents and display properties.

Segmentation of text documents has been studied extensively in information retrieval. Existing techniques roughly fall into two categories: lexical cohesion methods and multi-source methods [7]. The former identifies coherent blocks of text with similar vocabulary [2, 10, 15,23]. The latter combines lexical cohesion with other indicators of topic shift, such as relative performance of two statistical language models and cue words [1, 3]. In [11], Hearst discussed the merits of imposing structure on full-length text documents and reported good results of using local structures for information retrieval.

There are also several works on passage retrieval method although mostly applied in the area of text retrieval [14, 21]. [24] proposes sub-document access: it allows the user to zoom in on parts of the full text that are meaningful to his task. [8] utilizes the reciprocal of the length of each passage as an estimate of its relevance. [20] discusses that the query occurs in unrelated context results in non-relevant document return.

Our Web search based on MIUs is different from the research above. The nature of Web pages differs from a static text document. *Web contents* can switch from one topic to a completely different topic abruptly without requiring additional textual cues to "bridge the topic shift". Gradual topic shifts in text documents are often indicated by certain textual cues (e.g., "next, consider…", "firstly… secondly..."). Such cues, employed by text segmentation, are not applicable to Web pages. In our case, we considered the changes in visual cues (e.g., bold emphasis, sudden increase in font-size, change in font-color) of Web pages. Visual cues offer indications that a topic may have shifted within a Web page. In our Web page segmentation, we make use of both contents and presentation styles or visual features of the Web page to segment the page.

3. PRE-PROCESSING WEB PAGES

We now present the proposed technique. This section focuses on pre-processing of each Web page, i.e., building a HTML tag tree and segmenting the Web page into MIUs using the tag tree. The next section describes our ranking algorithm. All the procedures discussed in this section are done off-line. Since the proposed technique is used as an advanced search method or option for a base search engine, all the required information is assumed to be stored at the base search engine site.

3.1 Building HTML tag trees

Web pages are hypertext documents written in HTML that consists of plain text, tags and links to image, audio, and video files, etc. Like most search engines, our technique only uses plain text and tags in search. Plain text are strings of characters not embedded within any tags. It can have different appearances in terms of color, font, size and style as specified by tags. Tags (enclosed by a pair of angular brackets) define the display properties and characteristics of a Web page. In general, most Web documents are constituted of *opening* and *closing* pairs of HTML tags (indicated by < > and </> respectively). Within each

corresponding tag-pair, it can contain other pairs of tags, resulting in nested blocks of HTML codes.

Based on the nature of nested structure of HTML codes, a *HTML Tag Tree* can be built in a fairly straightforward manner for each Web page using its HTML source. A *node* in the tag tree contains a tag name, content text and its display attributes (color, font, size, etc).

3.2 Segmenting the Tag Tree into MIUs

We now segment the Web page into various MIUs using the tag tree. Although the tag tree already gives us an initial segmentation of the page, it is often too refined and is solely based on presentation features of the page. We need to merge some nodes in the tree to form coherent topic or information units. Our segmentation technique is based on both content and display similarities.

Merging of nodes is done in two steps: (1) merging each heading and its immediate content paragraph (note that a content paragraph may not have the and tags); (2) merging two adjacent text paragraphs. Below, we discuss these steps in turn.

Step 1 - ***Merging each heading and its immediate content paragraph***: In this step, we scan all the sibling nodes of a sub-tree from left to right to find all heading and paragraph pairs. This is performed in 2 sub-steps:

(i)*Identifying all potential heading and content paragraph pairs*: Let A and B be any two different leaf nodes of a sub-tree. We use *Len* to denote the length (number of words) of the text string stored in A or B. We use *tagRank* to denote the *font emphasis* given to the text strings stored in A or B. The value of *tagRank* is based on the *priority*. The highest value of *tagRank* is assigned to header tags (e.g..: <h1>, <h2> and so on), followed by formatting tags (e.g. <b>, , <blink>) and enlarged font sizes (<big>, <size…>). All the other tags are assigned the same rank value that is lower than the three types above. In general, a paragraph heading tends to be more prominent and distinct in terms of font size or appearance as compared to its content paragraph.

We use $Neig(A, B)$ to denote the neighboring relation of A and B, and node A is the left neighbor of B. The following condition is used to determine whether A is a potential heading for B (or B is A's immediate content paragraph). ($(A \cap B) \neq \emptyset$) means at least one word (term) in A (node A) also occur in its immediate text paragraph B).

$$(tagRank(A) >= tagRank(B)) \wedge (Len(A) < Len(B)) \wedge Neig(A, B) \wedge (A \cap B) \neq \emptyset) \quad (1)$$

Here, we use the length of A and B, font size attributes of A and B, and their neighborhood relation to check whether A is potentially a heading for B. Note that this is computed after *stop-words elimination* and *word stemming* have been performed. We use the Porter's algorithm given in [22] for the purposes.

(ii) *Further evaluation*: After (i), we have identified all the potential pairs. This sub-step further evaluates them using their display properties. For each (A, B) pair, we try to find the next pair (C, D) which also has a possible heading and content paragraph relationship as computed in sub-step (i). We then evaluate A, B, C and D using the display similarity, *DisplaySim*. *DisplaySim* counts the number of identical features (display properties) of any two nodes. We use the following condition:

$$\exists(C, D), Neig(C, D) \wedge (DisplaySim(A, C) >= \delta) \wedge (DisplaySim(B, D) >= \delta) \quad (2)$$

where $DisplaySim(X, Y) = |X.features \cap Y.features|$ (which is the size of the intersection). The set of features includes font, size, color, tag name, and default. We set $\delta = 3$ (determined from experimental observations), which means if ($DisplaySim(A, C) >= 3$) we consider they have high display similarity (this also applied to B and D).

This condition basically tries to see whether A and B have a parallel pair (C, D). If so, we confirm the heading and content paragraph relationship of A and B, and that of C and D. We believe that the display property comparison is more meaningful here since people often are able to segment a Web page correctly even they do not know the content of the page.

If conditions (1) and (2) are both satisfied, we merge nodes A and B, and at the same time nodes C and D, i.e., to put the attributes of B into A, and the attributes of D into C. Nodes B and D are deleted.

Step 2 - ***Merging two adjacent text paragraphs***: Here, we wish to join similar text paragraphs (some paragraphs may contain their headings after step 1). Let X and Y be two text paragraph nodes within the same sub-tree. We now compute their degree of content similarity, $ContentSim(X, Y)$. The *inner product* [17] is employed for the purpose (m is the total number of terms or keywords in $X \cup Y$). If term i exists in X, then $x_i = 1$, otherwise $x_i = 0$. If i exists in Y, $x_i' = 1$, otherwise $x_i' = 0$.

$$\text{ContentSim}(X, Y) = \sum_{i=1}^{m} x_i x'_i \quad (3)$$

If $ContentSim(X,Y) \geq \varpi$, we say that nodes X and Y have a high similarity. We can combine their contents, i.e., placing the content Y of into X. We set $\varpi = 2$, which is determined from experiments that reflect the acceptable level of similarity among various nodes well.

The overall algorithm is given in Figure 2. *maxDepth* is the maximum depth of the original tag tree. *treeDepth* is the depth of the tree that is being worked on. *Stree* is the set of all sub-trees at depth *treeDepth*. Each $subtree_i$ only contains leaf nodes, and no sub-trees below.

```
1  for (treeDepth = maxDepth –1; treeDepth < 0; treeDepth--) do
2    Stree ={subtree_i| subtree_i is a sub-tree at level treeDepth};
3    while |Stree| > 0 do  /* |Stree| is the size of the set Stree  */
4      for each subtree_i ∈ Stree do
5        for each Neig(A, B) do
6          if conditions (1) is satisfied  then
7            if ∃pair(C, D) & conditions (2) is satisfied then
8              Merge node A and B;
9            endfor
10     endfor
11     for each subtree_i ∈ Stree do
12       Scan all the nodes and their sibling nodes;
13       if ContentSim(X,Y) ≥ ϖ ,then
14         Merge node X and Y;
15     endfor
16     for each subtree_i ∈ Stree do
17       If A has no sibling then
18         move its content into its parent and delete it
19     endfor
20   endwhile
21 endfor
```

Figure 2: Merging nodes of a tag tree (segmenting a page)

4. THE RANKING ALGORITHM

After obtaining the MIUs from each page through segmentation, we index the Web pages in such a way that they can be retrieved and ranked quickly. As in normal

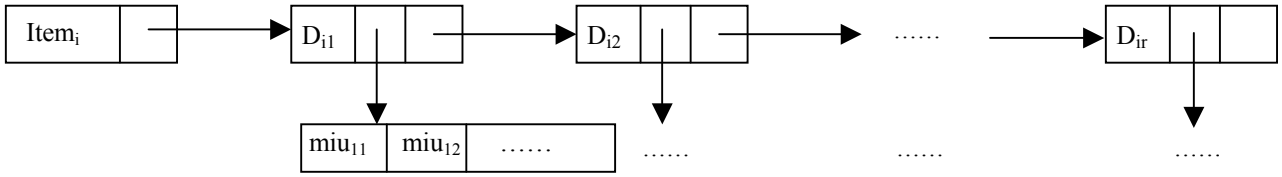

Figure 3: An inverted list with data structures for MIUs

search, we also use inverted lists to store the information of the Web pages. Thus, the search technique we adopted is similar to those in a normal search engine [4]. The main difference is that in our technique we need to index and retrieve MIUs of each page. We simply add an extra data structure to each inverted list node to indicate in which MIUs each word appears. Figure 3 illustrates the inverted lists indexing with the data structures for MIUs:

Here $Item_i$ (i =1, 2, …) is a word, D_{ij} is the j-th document that $Item_i$ occurs in and MIU_{xy} is the y-th MIU in the x-th subtree of the page D_{i1}. Each node includes three fields: *ID* (document *ID*), *seg* (a pointer to all the MIUs of the page containing $Item_i$), *next* (a pointer to next page). Note D_{ij} in an inverted list is stored in the increasing order. For any user query, $Q = \{k_1, k_2 \ldots\ldots, k_n\}$, we will consider if the words occur in the same or neighboring MIUs in the same subtree of the same page.

A search engine typically considers many factors in its ranking algorithm, e.g., *hyperlink information* (such as *authority score* and *hub score*), *word count-weight*, *type-weight* (title, anchor, URL, font size, etc), and *type-prox-weight* (how close multi-words occur in every type) [4]. In our ranking algorithm, we only focus on whether the query terms occur in a single MIU (or 2 neighboring MIUs within the same sub-tree) of a page. Since the proposed technique is intended to be used as an *advanced search* method for a *base search engine*, we utilize our MIU-based information and also the ranking information from the base search engine in our ranking process. The reason that we need ranking information from the base search engine is because we do not need to consider other factors except our MIU-based factor in our ranking algorithm. However, since we do not have access to any existing search engine program, only the ordering information of the pages returned by the base search engine is employed in our current ranking algorithm. If a search engine system is available, all factors should be integrated in a more sophisticated manner. Section 5 shows that even this simple approach is already able to produce remarkably good results.

The proposed ranking method aims to re-rank the results returned by the base search engine when the precision of its results is poor. The number of pages to be re-ranked is specified by the user. In our experiments, we re-rank the first 200 pages from Google. Re-ranking is done on-line at query time. Pre-processing as discussed in Section 3 is done off-line for all the pages at the search engine site, as it is not possible to know what queries will be issued by users, and it is too slow to do pre-processing of the top ranking pages from the base search engine at query time.

Our ranking algorithm basically computes two scores for each page, a *primary score* and a *secondary score*. The primary score is the maximum number of query terms that occur in a MIU of the page p. Let seg_i be the terms contained in i-th MIU of p, and *queryTerms* be the set of terms in the user query. The primary score of page p (denoted by $prScore(p)$) is computed as follows:

$$prScore(p) = \arg\max(|seg_i \cap queryTerms|) \quad (4)$$

If the primary score of page p is less than the number of query terms (i.e., not all query terms are covered), we compute the secondary score, which takes into account of the neighboring MIU on the right of each MIU in the same sub-tree. Let seg_{ji} be the set of terms in the ith MIU of the sub-tree j. The secondary score of p (denoted by $seScore(p)$) is computed with:

$$seScore(p) = \arg\max_j(\arg\max_i(|(seg_{ji} \cup seg_{j,(i+1)}) \cap queryTerms|)) \quad (5)$$

The overall ranking algorithm is given in Figure 4.

1 Create a set of variables $pageSet_i$, i = 1, …, n;
2 $pageSet_i = \emptyset$;
3 **for** all $p \in AllPages$ **do** $prScore[p] = 0$, $seScore[p] = 0$;
4 Retrieve the inverted lists of the query words $k_1, k_2, \ldots\ldots, k_n$;
5 Initialize pointer set: $L=\{p_1, p_2, ..., p_n\}$, here each p_i point to the first node in the corresponding link list;
6 **while** $\exists p_i \neq$ Nil, $p_i \in L$ **do**
7 $md = \min(p_i.id)$, $p_i \in L$;
8 Construct a pointer set LS from L: $\{p_j | p_j.id = md\}$;
9 **if** $|LS| = 1$ **then**
10 $prScore[p_j.id] = 1$, $seScore[p_j.id] = 1$;
11 **else** scan all the MIUs to compute *prScore* and *seScore* of the page by checking if the query words occur in the same or neighboring MIUs.
12 **for** all $p_j \in LS$ **do** $p_j = p_j.next$;
13 **endwhile**;
14 **for** each $p \in AllPages$ **do**
15 **if** $prScore(p) = n$ **then** $pageSet_n = pageSet_n \cup \{p\}$;
16 **else if** $seScore(p) = i$ **then** $pageSet_i = pageSet_i \cup \{p\}$;
17 **endfor**
18 Rank pages in the order of $pageset_n$, $pageset_{n-1}$ and so on. For the pages in each $pageset_i$, we follow their relative ranking in the results produced by the base search engine;

Figure 4: The ranking algorithm

In Figure 4, n is the number of query terms. *AllPages* is the set of top ranking pages (to be re-ranked) from the base search engine. Lines 1 and 2 create and initialize a set of set-variables to store the resulting pages as the first level ranking (which will become clear below). Line 3 initializes two arrays used to store the final Web page scores. Given user's query, lines 4 and line 5 retrieve the inverted lists of the query words and then create a pointer set. Each pointer in the pointer set points to the first Web page of an inverted list. From line 6 to line 13, we compute *prScore* and *seScore* for all the Web pages. The loop ends when all the pointers reach the end of the inverted lists, which means we have already finished processing all the Web pages in the inverted lists. In each loop, for all retrieved inverted lists, we first find the page with smallest document *ID* (*md*). After we process it (give this page the *prScore* and *seScore* scores), we move the pointers that point to the smallest

document *ID*s to the next node to begin the next loop. In a loop, if the page contains only one query word, both *prScore* and *seScore* are given the score of 1. Otherwise, the page contains at least two words in the user's query. Then, we need to check if they occur in the same or neighboring MIUs. In this process, we update the maximal number of query words contained in a single MIU and two neighboring MIUs. In line 15, if $prScore(p) = n$, p should be one of the top ranking pages, stored in $pageSet_n$ (since we believe that a MIU in a Web page that contains all the query terms is very likely to be relevant to the user). If $prScore(p)$ is less than n, we store p into $pageset_{n-i}$ according to its *seScore*, where n-i indicates how many keywords are found in two neighboring MIUs (line 16). Finally, we have a two-level ranking (line 18). The first level ranks the sets of pages in the order of $pageset_n$, $pageset_{n-1}$ and so on. The second level ranks all the pages in each $pageset_l$ according to their relative ranking in the results produced by the base search engine.

The complexity of our algorithm is similar to the complexity of a normal search engine. In a normal search engine, given a query, its main task is to check if the query terms occur in the same page, so the complexity is $|query|*v$ on average (we ignore the other cost in computing *authority score*, *hub score*, *word count* etc). Here $|query|$ is the number of query words, and v is the average length of all inverted lists. In our algorithm, we also need to check in each page whether the query words occur in the same or neighboring MIUs. Thus, it needs to traverse the MIU list of each page. Then complexity of our algorithm is $|query|*v*q$. Here q is the average number of MIUs in all the pages. Since q is normally very small, thus little extra time is needed by our new search technique.

5. EXPERIMENTAL RESULTS

This section evaluates the proposed technique. We first compare the precision results of our method with those from Google, and then discuss its running efficiency.

Evaluation of the ranking effectiveness is difficult in the context of web search because of the difficult tasks in (i) *choosing queries* and (ii) *evaluating the relevance* of search results. Our criteria for *choosing queries* are: they should be from diverse areas and unambiguous. By unambiguous, we mean that the intent of each query is agreed upon by a panel of 3 judges. We used queries from two independent sources, the entire collection of queries (351-400) from TREC-7 [25], and 30 queries from Metaspy of MetaCrawler [19] (which allows users to view others' queries being submitted to the system). For queries from Metaspy, we first collected a list of continuous queries and then removed those queries that are ambiguous, i.e., our panel of judges could not decide the intension of the user.

As for *evaluating the relevance* or correctness of the search results, the web pages produced should satisfy the conditions pre-defined by our judges or correspond to the standard narratives provided by TREC [25]. For example, TREC Query 354: Journalist Risks, the narratives stated are "*any document identifying an instance where a journalist has been killed, arrested or taken hostage in the performance of his work is relevant.*" Our judges evaluate the relevance of the search results with such narratives to obtain a consensus on the search precision.

The choice of using Google as a basis for re-ranking (*base search engine*) is because of its state-of-the-art search mechanism. In general, Google performs very well as a general-purpose search engine. However, there exist query phrases that it fails to perform satisfactorily. Our purpose is to provide advanced re-rankings for queries whose Google's precisions are low. For each query, we re-rank the first 200 search results from Google, after crawling and pre-processing the pages.

In general, information retrieval systems are evaluated using both precision and recall measures. However, in the context of Web search, the precision of the top-ranking results returned by a search engine is more important since most people only see the top 20-30 results [12, 13]. That is, even if a search engine has high recall, but if most of the relevant results are located below 20-30 top ranking results, there is little chance that the user will see them. Thus, many researchers believe that high precision is important even at the expense of recall [4]. In our experiments, we are only using precision of top 20 ranking results to evaluate the performance of our system.

The precisions of the top 20 ranking results from Google and our method (MIU) are compared in Table 1. The first column states the source of queries. The second and third columns list the average precisions of the top 20 results from Google and our method respectively. The fourth column provides the improvement percentage of MIU over Google on the average precision. Tables 3 and 4 in the Appendix list all the search queries and the corresponding precision for both data collections.

Data Collections	Average Precision		Improvement
	Google	MIU	
TREC-7 (351-400)	0.50	0.59	18.00 %
MetaSpy	0.54	0.63	16.67 %

Table 1: Average precision comparison for TREC-7 and MetaSpy

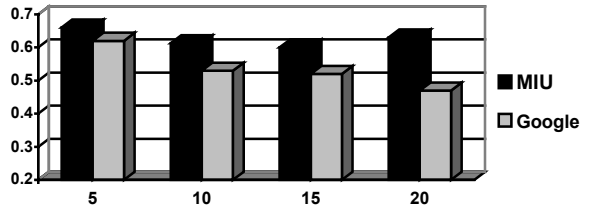

Figure 5: Average precision comparison per 5 returned pages of MIU and Google for TREC7 queries

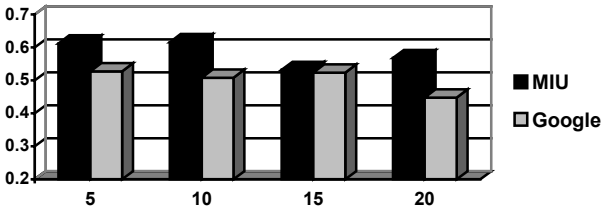

Figure 6: Average precision comparison per 5 returned pages of MIU and Google for MetaSpy queries

From Table 1, we observe that the average precision after our re-ranking is substantially higher. The improvement in precision by our system over that of Google is 18% for TREC-7 queries and 16.67% for

Metaspy queries. Figures 5 and 6 give the graphical comparison of the average precision of every 5 returned pages for both data collections. We observe that in general MIU is superior to Google for any number of top-ranked pages (used in computing precision).

Table 2 presents the won-lost-tied record of MIU against Google. For the Trec-7 data collection, 58% of the total number of queries increased in precision; 20% of queries remained unchanged and 22% of queries decreased in precision after applying our MIU method as compared to Google's ranking results. For the MetaSpy query collection, the precisions of 67% of the queries increased; the precisions of 3% of the queries remain unchanged and 30% of queries decreased. We observe that most instances of MIU performing worse than Google occur when the precisions of Google's results tend to be rather high. For example, for those queries that Google has better results, its average precision is 0.73 for the MetaSpy data collection. This precision value of Google should be highly satisfactory for most users and does not require additional MIU processing. That is why we say that our MIU method can be seen as an advanced search option. It should be used when Google's results are not satisfactory.

Data Collections	Increase	Draw	Decrease
TREC-7 (351-400)	58 %	20 %	22 %
MetaSpy	67 %	3 %	30 %

Table 2: Won-lost-tied record for TREC-7 and MetaSpy queries

We now briefly discuss the running efficiency of our system. We use a single machine (Sun E450 250MH with 500MB memory and a single processor) for all our experiments. In pre-processing, the major operations involved are crawling and indexing. It is difficult to measure how long crawling took overall because of complications like bandwidth limitations, crashed name servers, congested network and others. For indexing, the indexer runs at roughly 4 pages per second. Our indexer is not running in parallel, which affects the speed. All these pre-processing are mostly duplicated works of Google. They can be easily incorporated into Google, which will improve the performance significantly. For ranking, our ranking procedure handles roughly 50 pages per second, or 2 to 10 seconds for each query (which is mostly dominated by disk IO). Improving the efficiency of crawling, indexing and searching was not the main focus of this research. With further optimization and more powerful machines, the running speed can be improved significantly.

6. CONCLUSION

In this paper, we presented a technique to improve the precision of Web search. It is based on the idea of segmenting each web page into different MIUs (topic areas) according to its contents and HTML tags. In searching, only the terms in a single unit or at most two neighboring units of a page are used to match the user's query terms. This is different from existing techniques used by current search engines, which typically employ all the terms in the whole page to match the query terms. From the experiment results shown in Section 5, we observe that the precision of the ranking produced by our method is substantially higher.

7. REFERENCES

[1]. J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. "Topic detection and tracking pilot study final report." *DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[2]. D. Beeferman, A.Berger, and J. Lafferty, "A model of lexical attraction and repulsion." *ACL-97*, 1997.

[3]. D. Beeferman, A. Berger and J. Lafferty. "Statistical models for text segmentation." *Machine learning*, 34(1-3), 1999.

[4]. S. Brin, L. Page, "The anatomy of a large-scale hypertexture Web search engine." *Computer Networks* 30 (1-7), 1998.

[5]. S. Chakrabarti. "Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction." *WWW10*, 2001.

[6]. The CLEVER Project, http://www.almaden.ibm.com/cs/k53/clever.html

[7]. F. Y. Choi. "Advances in domain independent linear text segmentation." *NAACL'00*, Seattle, USA, 2000.

[8]. G. V. Cormack, C.L.A. Clarke, C.R. Palmer and S.S.L. "To Passage-Based Refinement (MultiText Experiments for TREC-6)." In D. K. Harman and Ellen Voorhees, editors, *The Sixth Text REtrieval Conference (TREC-6)*, 1998.

[9]. J. Dean, M. R. Henzinger. "Finding Related Pages in the World Wide Web," *WWW8*, 1999.

[10]. D. Eichmann, M. Ruiz, and P. Srinivasan, "A Cluster-based approach to tracking, detection and segmentation of broadcast news." *DARPA Broadcast News Workshop*, 1999.

[11]. M. Hearst. "Subtopic structuring for full-length document access." *ACM SIGIR 93*, 1993.

[12]. C. Hoelshher. "How Internet experts search for information on the Web." *The World Conference of the World Wide Web, Internet and Intranet*, 1988, Orlando, FL.

[13]. B. J. Jansen, "The effect of query complexity on Web searching results," *Information Research*, 6(1), 2000.

[14]. M. Kaszkiel, J. Zobel, "Passage Retrieval Revisited." *ACM SIGIR* 97, Philadelphia, PA, USA.

[15]. S. Kaufmann. "Cohension and collocation: Using context vectors in text segmentation." ACL-99, 1999.

[16]. S. Lawrence, and L. Giles, "Context and page analysis for improved web search."*IEEE Internet Computing*,2(4) 1998.

[17]. D. D. Lewis, et al. "Training algorithms for linear text classifiers." *ACM SIGIR 96*,1996.

[18]. Y. Li. "Toward a qualitative search engine," *IEEE Internet Computing*, 1998, July.

[19]. MetaCrawler Search Engine www.metacrawler.com Metaspy www.metaspy.com

[20]. M. Mitra, A. Singhal and C. Buckley. "Improving Automatic Query Expansion." *SIGIR98*, 1998.

[21]. E. Mittendorf and P. Schuble. "Document and Passage Retrieval Based on Hidden Markov Models." *SIGIR96.*

[22]. M. Porter. "An algorithm for suffix stripping." *Program*, 14(3): 130-137, 1980.

[23]. J. C. Reynar, "Statistical models for topic segmentation." *ACL-99*, 1999.

[24]. G. Salton, J. Allan and C. Buckley, "Approaches to Passage Retrieval in Full Text Information Systems." *SIGIR-93*, 1993, pp. 49-58.

[25]. Text REtrieval Conference (TREC) Data - English Test Questions (Topics) File List http://trec.nist.gov/data/topics_eng/index.html

[26]. W. S Lee, K. S. Candan, V. Quoc and D. Agrawal. "Retrieval and organizing Web pages by Information Unit." *WWW10*, Hongkong, 2001.

APPENDIX

Search Query	Google	MIU	Search Query	Google	MIU
Falkland petroleum exploration	0.55	0.70	mercy killing	0.10	0.25
British Chunnel impact	0.25	0.50	home schooling	0.20	0.30
journalist risks	0.05	0.60	autombile recalls	0.15	0.25
postmenopausal estrogen Britain	0.10	0.11	dismantling Europe's arsenal	0.20	0.60
human smuggling	0.60	0.85	euro opposition	0.70	0.70
transportation tunnel disasters	0.30	0.40	mainstreaming	0.35	0.35
anorexia nervosa bulimia	0.70	0.80	piracy	0.15	0.15
Food/drug laws	0.70	0.75	in vitro fertilization	1.00	1.00
health insurance holistic	0.30	0.45	rabies	1.00	1.00
Native American casino	0.45	0.60	El Nino	1.00	1.00
encryption equipment export	0.70	1.00	robotics	0.45	0.45
Nobel prize winners	0.85	0.95	tourism	0.00	0.00
hydrogen energy	0.80	0.95	sick building syndrome	0.60	0.60
World Court	0.75	0.90	amazon rain forest	0.10	0.10
obesity medical treatment	0.65	0.85	ocean remote sensing	0.85	0.40
alternative medicine	0.70	1.00	territorial waters dispute	0.60	0.55
mental illness drugs	0.20	0.50	blood-alcohol fatalities	1.00	0.80
space station moon	0.15	0.55	mutual fund predictors	0.50	0.20
hybrid fuel cars	0.65	0.85	drug legalization benefits	0.90	0.79
teaching disabled children	0.45	0.50	clothing sweatshops	0.80	0.75
radioactive waste	0.45	0.55	antarctica exploration	0.70	0.53
organic soil enhancement	0.45	0.60	commercial cyanide uses	0.68	0.63
illegal technology transfer	0.30	0.45	cigar smoking	0.35	0.25
orphan drugs	0.40	0.70	hydrogen fuel automobiles	0.90	0.85
r&d drug prices	0.30	0.60	oceanographic vessels	0.20	0.15

Table 3: Precision comparison using TREC-7 (we re-order the queries in TREC-7, i.e., putting those queries that MIU wins over Google first, then those tied queries and those queries that we lose)

Search Query	Google	MIU	Search Query	Google	MIU
star wars wallpaper	0.85	1.00	free download music	0.30	0.70
Free craft projects	0.60	0.80	information history tomatoes	0.45	0.65
supermodel success stories	0.35	0.50	literary films list	0.10	0.35
laser eye surgery	0.80	0.95	Singapore programming jobs	0.35	0.65
accident death photo	0.25	0.50	red ladies t-shirt	0.40	0.55
motorcycle dealers in Texas	0.45	0.55	html tag tree	0.50	0.50
First Communion letters	0.20	0.50	crime rates and ethnicity	0.55	0.50
entertainment in San Diego	0.45	0.85	Christmas island tour	0.60	0.55
karaoke machine	0.65	0.75	Mickey mouse club	0.80	0.70
growing marijuana	0.40	0.45	internet service provider illinois	0.75	0.65
award winning web sites	0.45	0.70	studies on travel writing	0.45	0.20
gall bladder surgery causes	0.50	0.70	California legal codes	0.90	0.70
alternative music origins	0.35	0.50	plant pathology journals	0.89	0.75
decorative candlestick sale	0.40	0.80	Michael Jordan shoes	0.95	0.70
heavyweight boxing championship	0.55	0.85	organizational industrial psychology	0.70	0.40

Table 4: Precision comparison using MetaSpy queries (the queries are also re-ordered according to the won-tied-lost record against Google)