A Novel Page Ranking Algorithm for Search Engines Using Implicit Feedback

Shahram Rahimi, Bidyut Gupta, Kaushik Adya, *Southern Illinois University, USA*

***Abstract*—The World Wide Web (WWW) is a fast growing network of information with billions of documents covering nearly every possible topic. Finding the relevant information efficiently within such a huge collection, without effective search mechanism, would be almost impossible. With the input of a few keywords a search engine (SE) can return a list of relevant documents by querying its information index for webpages. However, it is quite common to witness irrelevant results and ranking being presented to the user. One way to improve the ordering of the search results is to incorporate user feedback in ranking the pages for relevancy. This paper presents a mechanism for search engine enhancement by using implicit feedback in the form of *ClickThrough* data from the users. The ordering of the query result is re-arranged for the future queries based on the choices made by the majority of the users. An algorithm, with its implementation, is presented and then evaluated to demonstrate its capability as an add-on component for enhancement of the current ranking algorithms.**

***Index Terms*—Web search engines, ranking, feedback, clickthrough**

I. Introduction

The World Wide Web (WWW) [1] has seen a tremendous increase in size in the last two decades. The amount of information and resources available on WWW today has grown exponentially and almost any kind of information is present if the user looks long enough. Google [2] suggests that it has indexed over eight billion web pages, and that may be a fraction of all available web pages. The popularity of WWW can be attributed mainly to the uniform access method it provides to various information services and its hypermedia support which links a wide range of multimedia data physically distributed all around the world into a single gigantic virtual database. It provides a powerful medium for almost any user on the Internet to disseminate information about any topic. More and more information has become available on-line through WWW, from the latest news to scientific reports and satellite images. However, this information explosion has lead to an inevitable resource discovery problem.

In order to find relevant pages, a user has to browse through many WWW sites that may contain the information. Users may either browse the pages through entry points such as the popular portals, Yahoo, MSN and AOL or use a search engine to look for specific information. Beginning the search from one of the entry points is not always the best approach, since there is no particular organized structure for the WWW, and not all pages are reachable from others. In the case of using a search engine, a user submits a query, typically a list of keywords, and the search engine returns a list of the web pages that may be relevant according to the keywords. In order to achieve this, the search engine has to search its already existing index of all web pages for the relevant ones. Search engines are constantly engaged in the task of crawling through the WWW for the purpose of indexing. When a user submits keywords for search, the search engine selects and ranks the documents from its index in the decreasing order of relevance and usefulness in relation to the keywords.

The task of ranking the documents, according to some predefined criteria, falls under the responsibilities of the *ranking algorithms*. A ranking algorithm is one of the most crucial components of any search engine and usually requires much attention during the engine's development. Different search engines use different classes of ranking algorithms with varying degree of effectiveness and efficiency. Intuitively, a good information retrieval system should present relevant documents higher in the ranking, with less relevant documents following them. Although the ranking algorithms, following the search process, strive hard to achieve this goal, it is common to witness many irrelevant among the relevant queried information. This sub-optimal result has led to several researches in the area of search engine ranking algorithms.

This work presents a system for automatically rearranging the query results of an arbitrary search engine using implicit feedback obtained from users in the form of what is known as "ClickThrough" data [11]. Such ClickThrough data is easily available and can be recorded at a very low cost. The presented model combines the feedbacks of the users who query for the same term to maximize the quality of the ranking while minimizing the noise (produced by irrelevant clicking).

In the following sections, we discuss the architecture of a typical web search engine, the various ranking algorithms currently employed, the inadequacies with the current approaches, the proposed approach along with some implementation details, and the results. This will be followed

by the evaluation and verification of the results and possible future enhancements.

II. Genaral Architecture of a Web Search Engine

Figure 1 illustrates the architecture of a simple WWW search engine. In general, a search engine usually consists of three major modules:

a) Information gathering
b) Data extraction and indexing
c) Document ranking

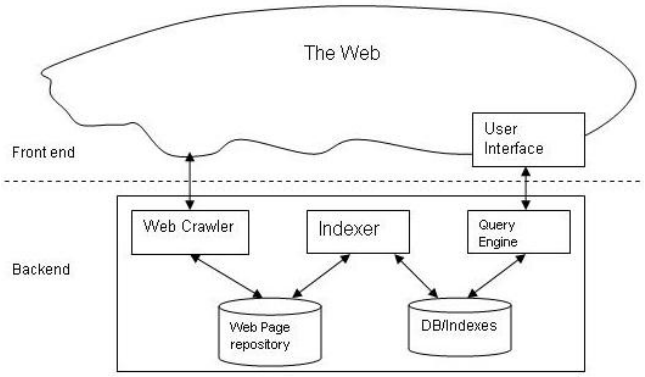

Figure 1. Architecture of a simple Web Search Engine

A. Information Gathering

Information gathering is sometimes more specifically known as web crawling in the context of a web search engine. The web crawlers (also referred to as spiders, worms or wanderers) are programs that are instructed to crawl the web autonomously in a methodical manner, to gather information from the content and structure of web pages. They browse the Web on behalf of the search engine, quite similar to how a human user follows links to reach different pages. First, a crawler reads from a list of seed URLs and visits the documents at these URLs. Each visited page is processed, and the URLs contained within a page are extracted and added to a queue. The crawler then selects the next URL from the queue and continues the process until a satisfactory number of documents are downloaded or the local computing resources are exhausted. To improve speed, the crawler usually connects simultaneously to multiple web servers in order to visit documents in parallel, either using multiple threads of execution or using asynchronous input/output. The crawler passes the information that it gathers to a page repository until either all the documents in the collection are exhausted or until the retrieval has reached some predefined criteria. Documents retrieved by the crawler are stored in a repository of web pages. To reduce the needed storage space, the pages are often compressed before being stored.

This basic algorithm for crawling the web can be modified or enhanced with many variations that give search engines different levels of coverage or topic bias. For example, crawlers for a particular engine might be biased to visit as many sites as possible, leaving out the pages that are buried deeply within each site. Other crawlers might be specialized on sites in one specific domain, such as governmental pages. The *crawl control module* is responsible for directing the crawling operation.

Once the search engine has been through at least one complete crawling cycle, the crawl control module may be informed by several indexes that were created during the earlier crawl(s). The control module may, for example, use a previous crawl's link graph to decide which links the crawlers should explore currently and which links they should ignore. Crawl control may also use feedback from usage patterns to guide the crawling process. Some websites may have to be crawled very regularly as their contents might change often, for example the newspapers or stock market related websites. In contrast, some other website may change very rarely, for e.g. personal web pages without much web activities. The crawl control module keeps track of such information and depending on the type of the page, the frequency of web crawler visiting it is defined.

B. Data extraction and Indexing

This is the second major module of a search engine. An indexer processes the pages in the repository and builds an underlying index of the search engine. The indexer tokenizes each page into words and records the occurrence of each word in the page. The next step is to extract the keywords from the pages and to create an index of keywords and the corresponding pages in which they were found. The indexer is also used to calculate scores such as the term and document frequencies of each word, which can be used for ranking the results or for further processing.

Keywords are practically innumerable in count and therefore having a comprehensive list of them would have a detrimental effect on the search engine. The engine should be able to gauge the importance of a word as a keyword, based on some observations. For instance, by observing the HTML code of a document we may find some words are formatted using <H1> and </H1> or bold faced, which signifies their importance in relation to the page content. Some of the keywords are predefined in the meta-tags of the documents while some others can be found in the page title. Based on this kind of heuristic observations, the keywords and their importance are identified.

Once the keywords are identified, they are indexed. The search engines usually maintain an index table and an inverted index table. The index table consists of the keyword and reference to the page in which it is present. The inverted index table contains the same field but in reverse order, reference to a page and all the keywords presented in it.

The Indexing module observes the crawled pages for the

presence of both old and new keywords, if any, and indexes them. This index is used by the ranking algorithm to project the pages in a non-decreasing order of their usefulness/relevance. The results from the indexer are then converted into an "inverted index". While the original indexing results maps a document to a list of words contained within it, an inverted index maps a word, to a list of documents containing the word. This allows fast retrieval of documents when a search query is passed to the search engine. The resulting searchable indexes are usually stored in a database.

C. Document Ranking

A search engine accepts queries from users and performs search on the available indexes. After retrieving search results from the indexes, the engine is also responsible for ranking the search results according to content analysis and link analysis scores. Search engines may also catch the results of popular search queries, which would benefit the performance for popular queries. After all the processing, the engine generates and renders the search results and submits them to the user interface to be displayed to the user.

Ranking algorithm is one of the most important components of any search engine that plays an important role in the successful response to a query. It ranks the documents in which the keywords given by the user are presented in a decreasing order of usefulness/relevance. The satisfaction of the user lies on the quality of rankings returned by this ranking algorithm. Ideally, the user should find the links she expects for a specific query on the top of the result set.

Different search engines use different ranking algorithms and most of them are proprietary and a well-kept secret. In the next section, we discuss some of the popular ranking algorithms and what they lack as the state of the art.

III. Ranking Algorithm – The State of The Art

In this section, some of the more popular ranking algorithms, such as Boolean Spread Activation, Most-cited, TFxIDF, Vector Spread Activation, PageRank and HITS, are first discussed and then evaluated. The terminologies used to define the ranking functions in this paper are as follow:

M : number of the query words
Q_j : the j^{th} query word, for $(1 < j < M)$
N : number of the pages in the index database
P_i : the i^{th} page or its ID number for $1 < i < N$
$R_{i,q}$: the relevance score of P_i with respect to query q
$L_{i,k}$: the occurrence of an incoming hyperlink from P_k to P_i, where $L_{i,k}=1$ if such a hyperlink exists, or 0 otherwise.
$C_{i,j}$: occurrence of Q_j, in P_i, where $C_{i,j} = 1$ if P_i, contains Q_j, or 0 otherwise
$S_{i,q}$: TFxIDF score of P_i
α : constant link weight $(0 < \alpha < 1)$
$TF_{i,j}$: the term frequency of Q_j in P_i
$TF_{i,max}$: the maximum term frequency of a keyword in P_i
IDF_j : $\log(N/\sum_{i=1}^{N} C_{i,j})$
U : a Web page
F_u : the set of the pages that U points/links to
B_u : the set of the pages that point/link to U
N_u : $|F_u|$

A. Boolean Spread Activation

This algorithm is based on the Boolean retrieval model, where retrieval is solely based on the occurrence or absence of keywords in the documents [7]. The ranking function is given as

$$R_{i,q} = \sum_{j=1}^{M} C_{i,j} \quad (1)$$

The Boolean Spreading Activation algorithm extends this strategy by propagating the occurrence of a query word in a document to its neighboring documents. This is possible in the WWW because a document usually has hyperlinks to and from one or more other documents, forming a network of documents.

B. Most-Cited

As with Spread Activation, this algorithm takes advantage of the information about the hyperlinks between WWW pages. Each page is assigned a relevance score which is the sum of the number of query words contained in other pages citing, or having a hyperlink to the page [7]. More formally, the relevance score of the page P_i with respect to query q is defined as

$$R_{i,q} = \sum_{k=1,k\neq i}^{N} (Li_{i,k} \sum_{j=1}^{M} C_{k,j}) \quad (2)$$

The objective of this algorithm is to assign, among potentially relevant documents, larger scores to the referenced documents than to the referencing documents.

C. TFxIDF

TFxIDF is another popular ranking methodology used by WebCrawler and Lycos. This makes use of the term frequency (TF) in a document and how often a term is used in a collection of documents (IDF). The TFxIDF algorithm is based on the well known vector space model [5], which typically uses the cosine of the angle between the document and query vectors in a multi-dimensional space as the similarity measure. As described in [15], vector-length normalization can be applied when computing the relevance score, $R_{i,q}$, of page P_i, with respect to query q:

$$R_{i,q} = \sum_{termj \in q} (0.5 + 0.5 \frac{TF_{i,j}}{TF_{i,\max}}) IDF_j \quad (3)$$

Generally speaking, the relevance score of a document is

the sum of the weights of the query term that appear in the document, normalized by the Euclidean vector length of the document. The weight of a term is a function of the word's occurrence frequency (also called term frequency). This weighting function gives higher weights to terms which occur frequently in a small set of documents.

D. Vector Spread Activation

This algorithm combines the vector space model and spread activation model. In this algorithm, each document is first assigned a relevance score using TFxIDF algorithm, and then the score of a document is propagated to the documents it references [7]:

$$R_{i,q} = S_{i,q} + \sum_{j=1, j\neq 1}^{N} \alpha Li_{i,j} \,.S_{j,q} \qquad (4)$$

E. PageRank Algorithm

PageRank computes a ranking for every web page based on a graph of the web and is used by one of the most popular search engines, Google [4]. It relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value. Google interprets a link from page A to page B as a vote, by page A, for page B [6]. But Google looks at more than the sheer volume of votes, or links a page receives; it also analyzes the page that casts the vote. Vote cast by pages that are themselves "important" weight more heavily and help to make other pages "important". The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it (incoming links). A page that is linked by many pages with high rank receives a high rank itself. If there are no links to a web page there is no support for this specific page.

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \qquad (5)$$

F. HITS

Another algorithm for ranking pages is HITS (Hyper textual Induced Topic Selection) [8]. HITS uses two values for each page, the *authority value* and the *hub value*. Authority and hub values are defined in terms of one another in a mutual recursion. Authority value is computed as the sum of the scaled hub values that point to that page, and hub value is the sum of the scales authority values of the pages it points to. Relevance of the linked pages is also considered in some implementations of this algorithm.

G. Drawbacks of the Current Approaches

The above ranking algorithms give importance to a page based on some metrics such as interest, popularity, location, etc (as also discussed in [9]). They rank the documents in real-time, i.e. when the search engine receives a request for documents, the ranking algorithm goes through the set of the pages in which the keywords are presented and orders them, using its ranking strategy. In almost every ranking algorithm, the facts considered are the linkage, keywords, format of the words, position of the words, depth of the page in the domain, and similar parameters. However, consider a case wherein, for a phrase 'q', a ranking algorithm returns a list of pages from which the user finds his/her top selections not among the top few results. This situation occurs because the users' choice, for a given set of keywords, has not been given importance. One way to overcome this would be to use the feedback from the users to re-rank the pages.

As was mentioned before, none of the ranking algorithms discussed here consider the feedbacks of the users to project the rankings. One evident reason for this lapse could be that users can not be relied on to leave enough proper feedback explicitly, no matter how easy the process is. One way to solve this problem is to collect implicit feedbacks based on the behaviors of the users. In this scenario, ClickThrough data is the key to learn about the users' choices.

In the next section, the ClickThrough data and its use for enhancing the results of a search engine is discussed.

IV. ClickThrough Data and The Proposed Algorithm

ClickThrough data in search engines can be thought of as triplets (q, r, c), consisting of the query q, the ranking r presented to the user, and the set c of links the user clicked on. Figure 2 illustrates this with an example: the user submits the query "computer science" (q) to a search engine and receives the ranking shown in the figure (r). The links that are bold faced are the links that the user has clicked on (c), upon receiving the results (this example is studied later in this section). In the sections to come, we show that the best description of the user's choice could be obtained from this observation.

Clearly, users do not click on links at random, but make somewhat informed choices. While ClickThrough data typically consist of some amount of noise and clicks are not perfect relevance judgment metrics, they are likely to convey overall reliable information regarding the relevancy of pages to particular search queries. Since every query corresponds to one triplet, the amount of data that is potentially available is virtually unlimited.

A. Incorporating the ClickThrough Data

ClickThrough data can be recorded with little overhead and without compromising the functionality and usefulness of the search engine. In particular, compared to explicit user feedback, it does not add any overhead for the user. The query q and the returned ranking can be easily recorded whenever the resulting ranking is displayed to the user. For recording the clicks, a simple proxy system is employed. The general architecture of the proposed system is illustrated in Figure 3.

1. The Collection of Computer Science Bibliographies
   liinwww.ira.uka.de/bibliography/
2. **School of Computer Science/Carnegie Mellon University**
   www.cs.cmu.edu/
3. Computer Science Corporation
   www.csc.com/
4. **Welcome to the MIT Laboratory for Computer Science**
   www.lcs.mit.edu/
5. **Stanford Computer Science Department**
   www-cs.stanford.edu/
6. Department of Computer Science
   www.cs.umd.edu/
7. Computer Science
   library.albany.edu/subject/csci.htm
8. University of Washington Computer Science & Engineering
   www.cs.washington.edu/
9. Computer Science Division| EECS at UC Berkley
   www.cs.berkeley.edu/
10. **Department of Computer Science| University of Illinois at Urbana ...**
    www.cs.uiuc.edu/

Figure 2. Results of a search for "Computer Science" from a popular Search Engine

There are strong dependencies between the three parts of the triplet (q, r, c). The presented ranking r depends on the query q as determined by the retrieval function implemented in the search engine. Furthermore, the set c of clicked links depends on both the query q and the presented ranking r. A user is more likely to click first on a link if it is relevant to q [10, 11]. While the overall dependency is desirable and interesting for analysis, the dependency of the clicks on the presented ranking r may be actually detrimental. In particular, a user is less likely to click on a link low in the ranking, independent of how relevant it is. In the extreme case, the probability that the user clicks on a link with the least rank is virtually zero even if it is the document most relevant to the query. No user will scroll down the ranking so far enough to observe this link. Therefore, in order to get interpretable and meaningful results from ClickThrough data, it is necessary to consider and model the dependencies of c on q and r appropriately.

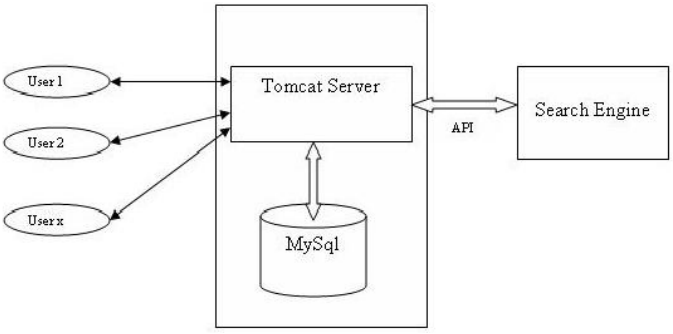

Figure 3. The general architecture of the model

Consider the example from Figure 2. The user has clicked on $link_2$, $link_4$, $link_5$ and $link_{10}$ ($link_k$ represent the k^{th} link, or k^{th} rank, in an ordered list). While it is not possible to infer that the links 2, 4, 5 and 10 are relevant on an absolute scale, it is much more plausible to infer that $link_2$ is more relevant than $link_1$ and so on. Assuming that the user scanned the ranking from top to bottom, she must have observed $link_1$ before clicking on $link_2$, making a decision not to click on $link_1$. Given that the abstract presented with the links are sufficiently informative, this gives some indication of the users' preferences. Similarly, it is possible to infer that $link_4$ is more relevant to this particular user than $link_1$ and $link_3$. This means that clickthrough data does not convey absolute relevance judgments, but partial relative relevance for the links the user browsed through. A search engine ranking the returned links according to their relevance to q should have ranked $link_2$ ahead of $link_1$ and $link_4$ ahead of $link_1$ and $link_3$ and so on.

$link_2 <_{ri} link_1$	$link_4 <_{ri} link_1$	$link_5 <_{ri} link_1$	$link_{10} <_{ri} link_1$
	$link_4 <_{ri} link_3$	$link_5 <_{ri} link_3$	$link_{10} <_{ri} link_3$
			$link_{10} <_{ri} link_6$
			$link_{10} <_{ri} link_7$
			$link_{10} <_{ri} link_8$
			$link_{10} <_{ri} link_9$

Figure 4. Information extracted from a user's implicit feedback. In this figure $link_j <_{ri} link_k$ states that $link_j$ has a higher rank than $link_k$.

Denoting the ranking preferred by the user i with r_i (user i's ranking), we get partial information of the form represented in Figure 4. It is this ranking with which user i might have been most satisfied. It has to be kept in consideration that this information can be potentially noisy.

This strategy of extracting the kind of information represented in figure 4 is summarized in the algorithm given below (in this paper $link_k$ is also represented as l_k):

For r: (l_1, l_2, l_3,…, l_n), an ordered set of links returned by a search engine, and a set r_k, containing user k's clicked-on links, extract $l_i <_{rk} l_j$ for all $l_i \in r_k$ and all $l_j \notin r_k$, where $l_i <_{rk} l_j$ states that link l_i has a higher rank than link l_j.

This algorithm gives a partial relative ordering based on a user's choice. However, it does not take into the consideration the links that the user has not clicked on for the purpose of the total ordering. One modification could be to maintain the same relative positions of the un-clicked links as in the original ranking while enforcing the ClickThrough feedback. This approach helps to get a complete relative ordering for each user. So, the modified algorithm would be as follows:

If a user clicks on links l_i and l_j in r, where i<j and there exists no other clicked links in between i and j then
{
If i < j-1 then,
in r_k the partial order will be …l_i, l_j, $l_{(i+1)}$,… $l_{(j-1)}$, $l_{(j+1)}$, $l_{(j+2)}$,…
Else maintain the same order as in r
}

Else maintain the same order as in r.

The above algorithm can also be phrased as follows:
If a user clicks on links l_i and l_j and there are no other clicked links in between them, and if $i<(j-1)$, then for all $m \in [i+1, j-1]$, $l_j <_{rk} l_m$, while all the links belong to $[l_{i+1}, l_{j-1}]$ retain their original ordering from r. By following this procedure, the desired r_k is formed, i.e. r_k: $l_i <_{rk} l_j <_{rk} l_{(i+1)} <_{rk} \dots <_{rk} l_{(j-1)} <_{rk} l_{(j+1)} <_{rk} l_{(j+2)} <_{rk} \dots$.

Following this modified algorithm, the complete relative ordering, based on the users' choice, can be produced. For the example discussed in figures 2, the result is as follows:
link2 <rk link4 <rk link5 <rk link10 <rk link1 <rk link3 <rk link6 <rk link7 <rk link8 <rk link9

B. *The Proposed Multi-User Feedback Approach*

Similar to the algorithm proposed in the previous subsection, [10] and [11] have proposed approaches to collect implicit feedback using ClickThrough data. The drawback of these algorithms is that they do not have a mechanism to consider multiple users inputs for rearrangement purpose. When the inputs of only one user is considered for ordering (for a particular query term), not only the outcome will be sensitive to noise (irrelevant clicking), but also the ordering will not be mature enough to be used for other users with the same query term. If the ClickThrough data is collected from multiple users and combined appropriately, it would be comprehensible that the quality of the ordering would improve.

The proposed approach makes use of the algorithm presented in the previous section. For a query, q, we obtain r_is for a considerable number of users, say 'n'. After obtaining the required data, the average displacement of the documents is calculated according to the feedback obtained in the above fashion. Once n such instances are collected, the returned documents for query q are reordered with ranking r^*. The procedure to calculate r^* is explained later in this section. To proceed with our discussion, some formal descriptions are presented.

For a ranking function F and a document set D, the query phrase q results in the ranking r: $(l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8 \dots, l_k)$, such that

$F(q) = r$: $l_1 <_r l_2 <_r l_3 <_r l_4 <_r l_5 <_r l_6 <_r l_7 <_r l_8 <_r \dots .. <_r l_k$

Rank of a document d, for the query q, can be defined as the corresponding position of its link in the ranking function $F(q)$; i.e. for the document d, in $F(q)$, if the relating link is l_j, then the rank of d is j.

Δ is an integer array of size k, initially set to 0. This is used to store the displacements of the links in r^* with respect to r. Displacement of the document d in r_i compared to r is given by the change in position of its link l. It can be either negative or positive.

Every time a user queries for q using F and clicks on some of the returned links, a new r_i is generated. For each one of these r_is the relative changes in the positions of the links are calculated and then inserted into the corresponding position in Δ (for r_i it would be $\Delta[i]$). After n such instances for q, the re-ranking procedure is started. The average displacement (AD_q), for each link in the array, will be calculated as

$$AD_q[i] = \Delta[i] / n \quad (6)$$

Next, the original rank of each link is added to the corresponding displacement, AD_q. This is to increase the weight of the original ordering (r) to downgrade the possible noise for small n. The final step in the re-ranking procedure is to re-rank the links for q in the increasing order of their ($AD_q + r$) values. From here onwards, for query q the newly ranked list will be shown to the users.

By following this algorithm, the outcome of each user query is arranged in such a way that the results expected by the user moves toward the top of the ranking and the relative order for the un-clicked links is retained as per the ranking function. In the upcoming sub-section, for illustration purposes, the presented procedure is used to rearrange the results of a query with feedback from four users.

C. *Illustration of the re-ordering procedure*

In this subsection, a simple example is given (Table 1). Suppose for query q, the original ranking from F is given by r, where r: (l_1, l_2, l_3, l_4, l_5, l_6), and the ranking which might have best suited each of the users is given by r_i ($1 \le i \le 4$) as depicted in Table 1. For each r_i, the algorithm calculates the displacements for the links, when compared to the original r, and then form Δ by adding these displacements together. Then for each link, the average displacement (represented by AD_q) is calculated and added to its original position in r. Finally, the links are re-arranged based on the increasing order of (AD + r) to get the order r^*: (l_1, l_3, l_2, l_4, l_5, l_6).

V. IMPLEMENTATION AND VERIFICATION

In this section, first, the implementation of the system is described, and then, it is verified and evaluated using Kendall's Tau statistical measurement.

As was mentioned before, instead of developing a Web Search Engine from scratch to record the activities of the users, we have come up with an architecture that utilizes an already existing SE resource. A web-based interface was

TABLE 1. AN EXAMPLE OF APPLYING THE PRESENTED ALGORITHM.

										Displacement for the links					
	r:	l1	l2	l3	l4	l5	l6		Init:	0	0	0	0	0	0
1)	r1:	l_1	l_3	l_5	l_2	l_4	l_6		1)	0	2	-1	1	-2	0
2)	r2:	l_3	l_4	l_1	l_2	l_5	l_6		2)	2	2	-2	-2	0	0
3)	r3:	l_1	l_3	l_2	l_4	l_5	l_6		3)	0	1	-1	0	0	0
4)	r4:	l_1	l_2	l_4	l_3	l_5	l_6		4)	0	0	1	-1	0	0
									Λ	2	5	-3	-2	-2	0
								ADq	÷5	0.4	1	-0.6	-0.4	-0.4	0
								r		1	2	3	4	5	6
								r + AD		1.4	3	2.4	3.6	4.6	6
								r*		l_1	l_3	l_2	l_4	l_5	l_6

developed to take the query results from an existing SE and to project them to the user while recording ClickThrough data and executing the presented algorithm. This requires that the SE provides Application Programming Interface (API) in order for our implementation to utilize its services. Google in particular provides a very rich and comprehensive API for external usage and is the SE choice of this project.

Basically, the interface performs the duty of receiving a user query and retrieving the results from the SE (in this case Google) and returning them to the user like any other SE's interface would. Without interrupting the users' activities, this interface silently records these links and when the user closes the interface window, the results are sent to the server hosting the interface. The server manages an intermediate database, in which it stores the ClickThrough data and uses them to rearrange the results for the future query, using the presented algorithm. The general architecture of the system was shown earlier in Figure 3.

A. Implementation

The major component of the implementation is a proxy (interface) server that serves as an interface between users and Google. In order to decide whether a query should be directly answered by the search engine or intercepted by the proposed algorithm, a simple mechanism is used. If the term was queried for less than '*n*' times (where *n* is a threshold value set by the administrator), the proxy server uses the SE's API and directly fetches the results from it. Then upon receiving the results from the SE, the server stores them in the intermediate database for the future use. If the number of the times the term was queried exceeds '*n*', then the server looks up for the results which have been already processed and available locally in the intermediate database.

Each time the results are sent back to the user, the GUI keeps track of the user's activities and responses. It is during this interaction of the user with the displayed results that the ClickThrough data is recorded and when the user completes the session, the recorded data is sent back to the proxy server, which executes the presented algorithm.

The technologies used for the implementation are JSP, JavaBeans, JavaScript, MySql 4.0 database, MySql-java-connector and Google API. Apache Tomcat server 5.5 and NetBeans IDE 4.0 were used to develop the software. For the front-end and the middleware, JSP and JavaBeans were used. To store the information, MySql database server was utilized in the backend.

B. Processing of the ClickThrough Data

After the successful implementation of the presented model and the proxy server, for evaluation and verification purposes, data was collected from several interacting users. Concurrently, information was gathered regarding the keywords and the links returned by Google in response to the queries. The ClickThrough data was then processed to calculate the displacement in the position of the links for each user and was recorded in the intermediate database. In this sub-section some examples, based on the real data from the database of our proxy server, is presented to further describe the proposed approach.

First, let us look into a simple example in order to better understand the system. Table 2 illustrates the process and the result of a ranking rearrangement, calculated from the feedback of ten users.

In Table 2, the original ranking from Google is given by *r* and the ranking which might have best suited each of the users is given by r_i ($1 \leq i \leq 10$). For each r_i, the algorithm calculates the displacements for the links when compared to the original *r* and recorded them under the Displacement section. Then the average displacement for each link is calculated and added to its original position in *r*. Finally, the links are re-arranged based on the increasing order of the average displacement plus original rank to get the re-arranged order *r** which in this case is the same as *r*.

Similar to the example in Table 2, for another example illustrated in Table3, the new order of the links are calculated for 20 users by the presented method.

C. Verification

To evaluate the proposed model by verifying the results of re-arrangement, a statistical measurement called Kendall's Tau (τ) [12, 13] is utilized. Kendall's Tau is one of the most frequent used methods in statistics for rank correlation and is defined as follows:

If D is the collection of documents, for two finite strict orderings $r_a \subset D \times D$ and $r_b \subset D \times D$, Kendall's τ can be defined as:

$$\tau\,(r_a, r_b) = \frac{P-Q}{P+Q} = 1 - \frac{2Q}{\binom{m}{2}} \qquad (7)$$

where m is the number of documents in D, P is the number of concordant pairs and Q is the number of dis-concordant pairs (inversions). In a pair of orderings, d_i and d_j are concordant if both r_a and r_b agree in how they order d_i and d_j (where d_i represents a document ordered as the i^{th} element of the list). It is dis-concordant, if they disagree. Using the original r and the modified r^*, it can be shown that the modified r^* provides better τ .

Let us name the initial search result, given by Google for some specific query term, as 'r'. As was described before, after n times collecting the ClickThrough data, for the specified query term, the algorithm is executed and the result in the intermediate database is rearranged such that the links which have been clicked more often stand the chance of climbing up the ranking order. This reordered list is named r^*.

TABLE 2. FIRST EXAMPLE: COLLECTED CLICKTHROUGH DATA FROM TEN USERS AND THE CALCULATED DISPLACEMENT AND RANKING RE-ARRANGEMENT

r	l1	l2	l3	l4	l5	l6	l7	l8	l9	l10
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
r1	1	3	4	2	5	8	6	7	9	10
r2	2	3	4	1	5	6	8	7	9	10
r3	1	3	5	2	4	6	7	8	9	10
r4	1	4	2	3	5	6	7	8	9	10
r5	1	2	3	4	5	6	7	8	9	10
r6	2	1	3	5	4	6	7	8	9	10
r7	1	2	4	3	5	6	7	9	10	8
r8	1	2	6	3	4	5	7	8	9	10
r9	1	3	2	4	5	6	7	8	9	0
r10	1	5	2	3	4	6	7	8	9	10
Displacement										
	0	2	-1	-1	0	1	1	-2	0	0
	3	-1	-1	-1	0	0	1	-1	0	0
	0	2	-1	1	-2	0	0	0	0	0
	0	1	1	-2	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	1	-1	0	1	-1	0	0	0	0	0
	0	0	1	-1	0	0	0	2	-1	-1
	0	0	1	1	1	-3	0	0	0	0
	0	1	-1	0	0	0	0	0	0	0
	0	1	1	1	-3	0	0	0	0	0
Average Displacement										
	0.4	0.5	0	-0.1	-0.5	-0.2	0.2	-0.1	-0.1	-0.1
Avg Disp + original rank										
	1.4	2.5	3	3.9	4.5	5.8	7.2	7.9	8.9	9.9
Reordered links r*										
	l1	**l2**	**l3**	**l4**	**l5**	**l6**	**l7**	**l8**	**l9**	**l10**

τ can be calculated for the sets of displacements, using r and r^* separately. If our model is affective, then $\tau\,(r^*, r_i)$, Tau value calculated with r^* and r_i, should be larger than the Tau calculated with r, $\tau\,(r, r_i)$. Saying that, it is not always possible that $\tau\,(r^*,r_i) > \tau\,(r, r_i)$ is valid for every i $(1 \le i \le n)$ for obvious reasons such as noise and less reliable ClickThrough data collected from some users. Therefore, if $\tau\,(r^*,r_i) > \tau\,(r, r_i)$ materializes for the majority of r_i values then the objective of the presented approach is verified.

TABLE 3. SECOND EXAMPLE: COLLECTED CLICKTHROUGH DATA FROM TWENTY USERS AND THE CALCULATED DISPLACEMENT AND RANKING RE-ARRANGEMENT

r	l1	l2	l3	l4	l5	l6	l7	l8	l9	l10
r1	l2	l3	l1	l5	l6	l4	l7	l8	l9	l10
r2	l2	l1	l3	l5	l4	l6	l7	l10	l8	l9
r3	l1	l2	l3	l4	l5	l6	l7	l8	l9	l10
r4	l2	l7	l3	l1	l4	l5	l6	l8	l9	l10
r5	l1	l3	l2	l4	l5	l6	l8	l7	l9	l10
r6	l2	l5	l6	l8	l1	l3	l4	l7	l9	l10
r7	l1	l3	l4	l5	l2	l6	l7	l8	l9	l10
r8	l2	l3	l4	l5	l1	l6	l7	l8	l9	l10
r9	l2	l3	l1	l4	l5	l6	l7	l8	l9	l10
r10	l2	l3	l1	l5	l6	l7	l8	l9	l4	l10
r11	l2	l1	l3	l4	l5	l8	l9	l10	l6	l7
r12	l1	l3	l2	l4	l5	l6	l8	l7	l9	l10
r13	l2	l3	l1	l4	l5	l6	l7	l8	l9	l10
r14	l1	l2	l3	l4	l5	l6	l7	l8	l10	l9
r15	l2	l5	l1	l3	l4	l6	l7	l8	l9	l10
r16	l2	l3	l1	l5	l4	l6	l7	l8	l9	l10
r17	l2	l1	l3	l5	l4	l6	l7	l8	l9	l10
r18	l4	l2	l5	l1	l3	l8	l9	l7	l6	l10
r19	l1	l2	l5	l3	l4	l6	l7	l8	l9	l10
r20	l2	l3	l7	l1	l5	l6	l4	l8	l9	l10
Displacement										
	2	-1	-1	2	-1	-1	0	0	0	0
	1	-1	0	1	-1	0	0	1	1	-2
	0	0	0	0	0	0	0	0	0	0
	3	-1	0	1	1	1	-5	0	0	0
	0	1	-1	0	0	0	1	-1	0	0
	4	-1	3	3	-3	-3	1	-4	0	0
	0	3	-1	-1	-1	0	0	0	0	0
	4	-1	-1	-1	-1	0	0	0	0	0
	2	-1	-1	0	0	0	0	0	0	0
	2	-1	-1	5	-1	-1	-1	-1	-1	0
	1	-1	0	0	0	0	0	0	0	0
	0	1	-1	0	0	0	1	-1	0	0
	2	-1	-1	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	-1	1
	2	-1	1	1	-3	0	0	0	0	0
	2	-1	-1	1	-1	0	0	0	0	0
	1	-1	1	0	-1	0	0	0	0	0
	3	0	2	-3	-2	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	3	-1	-1	3	0	0	-4	0	0	0
Avg Disp Δ										
	1.60	-0.40	-0.15	0.60	-0.70	-0.20	-0.35	-0.30	-0.05	-0.05
Δ + original rank										
	2.60	1.60	2.85	4.60	4.30	5.80	6.65	7.70	8.95	9.95
Reordered links r*										
	l2	**l1**	**l3**	**l5**	**l4**	**l6**	**l7**	**l8**	**l9**	**l10**

Consequently, Kendall's Tau was applied to the data in Tables 2 and 3, collected by our proxy server, in order to

verify the advantage of the arraignments provided by the presented model. For Table 2, Tau values were calculated for the original ranking (r) and the reordered ranking (r*) against r_1, r_2,…, and r_{10} and the outcome was compared in Table 4.

Table 4. Values of Tau for Table 2

Value of i	1	2	3	4	5	6	7	8	9	10
Tau(r,ri)	0.87	0.82	0.82	0.93	0.93	0.82	0.87	0.91	0.96	1
Tau(r*,ri)	0.82	0.87	0.87	1	1	0.87	0.91	0.96	0.91	0.96

As it is illustrated, τ (r*,r_i) is larger than τ (r,r_i) for seven cases out of ten, which indicates better ordering by the proposed system in 70% of the cases. Figure 5 illustrates the difference between the Tau values for each r_i reflected in table 4. In Table 5, the calculated values of Tau, for the entries in Table 3 are demonstrated. Yet again for this data, τ (r*,r_i)> τ (r, r_i) is valid for 75% of the cases. Figure 6 illustrates these results graphically.

Fifteen such data sets (as in Tables 2 and 3) were collected for fifteen different sets of keywords and over 20 users interacted with the proxy server for each set of keyword. In an average of %73 of the cases the proposed system provided better ordering compared to the original ordering of Google which is a considerable superiority. This outcome is not surprising. This is obviously for the reason that in the calculation of r*, the users' choices were collected and considered in the rearrangement process.

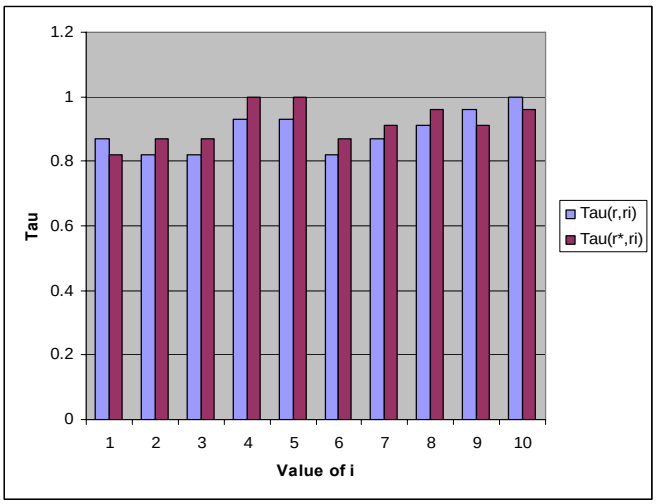

Figure 5. Comparison of Tau's from Table 4

In order to evaluate the proposed system even further, the values of Tau were treated statistically. Table 6 presents the values for the number of times τ (r*,r_i) is greater than its counterpart and vice-versa, mean and the standard deviation of the Tau vectors for data in Tables 4 and 5.

From Table 6, it is observable that in both cases, the mean value for τ (r, r*) is larger than the mean value of τ (r, r_i), with a small standard deviation for both sets of Tau values. The small standard deviations show that the values of Tau for both r and r* are very close to their mean and therefore the computation of the means for satisfactory evaluation is reasonable.

These outcomes were closely followed by the other data sets (total of fifteen), showing that the presented system, by taking the users feedbacks into consideration, provides more reasonable ordering compared to what Google is providing.

Furthermore, since this system does its computation dynamically, as the preferences of the users change for a particular query term, it modifies the ordering to better reflect the users' choice. In other words, the presented mechanism always reflects the current preferences of the users. The main achievement of this model over the contemporary models, such as [10] and [11], is that it is taking into consideration the choice of the majority of the users and allows the links they prefer to come forward in the ranking.

Major search engines like Google have recently started making use of the users' feedback to study their behavior. For this, Google uses the Google Toolbar [14]. This toolbar has a stealth functionality of sending the information of the pages visited by the user to the Google server, if the user enables the advanced features.

VI. Conclusion and Future Work

In this work, a new approach for enhancing the ordering of the query results, applicable to any search engine as an add-on module, is presented. The developed system is general enough to be applied to any search engine to improve the quality of its results. The initial experimental results are very encouraging; nevertheless, the system could be tested after a longer period of usage to be evaluated thoroughly. Since this system forms the query results based on the users' implicit feedback (ClickThrough data), it is bounded to get better results as the volume of its usage increases. The model is designed to give more weight to the choices made by a group of people with the largest number of users, for a given set of keywords. The fact that the lead players in the market such as Google have also started to employ some approaches for using the ClickThrough data, to analyze the choices made by the users, reinforces the argument of this work. The main advantage of this work over other similar models is our approach for combining the collected ClickThrough data, from multiple users, to rearrange the query result that provides a better ordering. The presented model becomes more stable and noise resistant as the number of the users, interacting with the interface, is increased.

Saying that, in the presented system, only feedbacks in the form of links that are visited by the users are considered. There is another form of feedback that could be collected that is the time spent by a user on a linked document from the result set. Adding this feature would make the ordering more accurate, since the time actively spent by an individual on a

document usually reflects the interest of the user in the document.

TABLE 5. VALUES OF TAU FOR TABLE 3

Value of i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Tau(r,ri)	0.9	0.82	0.63	0.9	0.86	0.82	0.82	0.64	0.78	0.86	0.78	0.78	0.94	0.82	0.89	0.78	0.62	0.89	0.78	0.84
Tau(r*,ri)	0.91	0.88	0.7	0.82	0.88	0.87	0.75	0.89	0.87	0.78	0.89	0.86	0.91	0.94	0.94	0.80	0.65	0.88	0.80	0.91

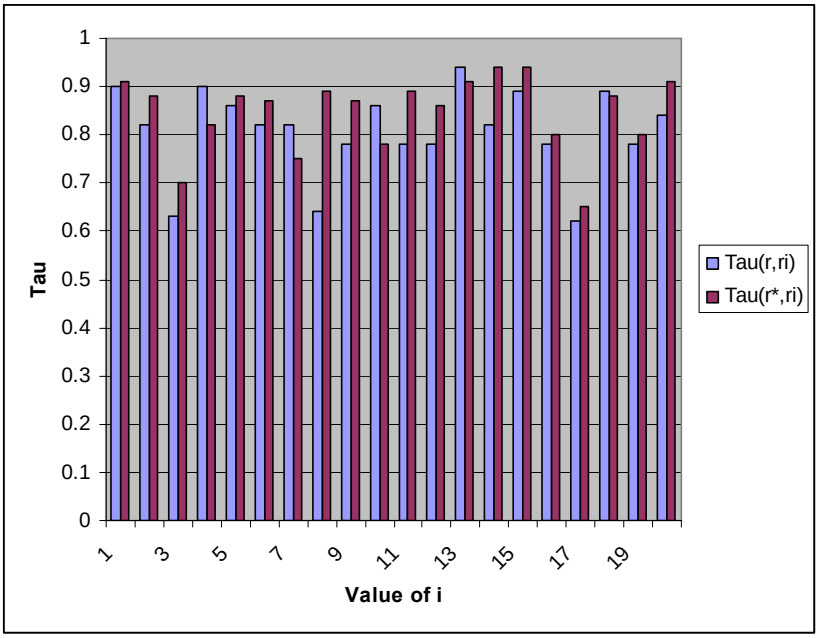

Figure 6. Comparison of Tau's from Table 5

TABLE 6. STATISTICAL COMPARISON OF THE VALUES OF TAU

	For the data in Table 4			For the data in Table 5		
	Num of time r/r* is greater	Mean	Std Dev	Num of time r/r* is greater	Mean	Std Dev
Tau(r,ri)	3	0.893	0.06	4	0.8	0.091
Tau(r*,ri)	7	0.917	0.06	16	0.84	0.006

Another enhancement to the present model could be the personalization of the results to each user or to a category to which a user belongs. In order to personalize the query results, only the feedbacks from the previous queries of a particular group of users should be utilized. At the same time to personalize according to a category, the feedback from the users that belong to the same category could be used to rearrange the results. The categories can be decided according to the initial personal information and interests provided by the user. These can be again sub-categorized into finer groups. In this manner, the results can reflect the choice of people with common interests.

REFERENCES

[1] Bereners-Lee, T., Cailliau, R., gruff, J., and Pollermann B., "World Wide Web: The Information Universe," Electronic Networking: Research, Applications and Policy, 1(2), 1992.

[2] "Pages Indexed," September 2005, http://google.com

[3] "Nielsen NetRatings for Search Engines," August 2005, http://searchengnewatch.com/reports/article.php/2156451.

[4] L. Page, S. Brin. "The PageRank citation ranking: Bringing order to the web," Technical report, Stanford Digital Library Technologies, 1998.

[5] G. Salton, A. Wang, and C. Yang. "A vector space model for information retrieval" In *Journal of the American Society for information science*, volume 18, pages 613-620.

[6] "Google Technology," July 2005, http://www.google.com/technology/

[7] Budi Yuwono, Dik L. Lee. "Search and ranking Algorithms for locating Resources on the World Wide Web", *Data Engineering, Proceedings of the Twelfth International Conference*, March 1996, pp:164 – 171.

[8] J. Kleinberg, "Authoritative sources in a hyperlinked environment," In *Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms*, ACM Press, New York, 1998, pp 668-677.

[9] A Arasu, J Cho, H Garcia-Molina, A Paepcke, "Searching the Web," *ACM Transactions on Internet Technology*, 2001 - portal.acm.org.

[10] T. Joachims, "Optimizing Search Engines using Clickthrough Data," July 2005, http://www.joachims.org.

[11] T. Joachims. "Unbiased Evaluation of retrieval quality using clickthrough data," Technical report, Cornell University, Department of Computer Science, 2002. http://www.joachims.org.

[12] Kendall. "*Rank Correlation Methods,*" Hafner, 1955, pp 86-87.

[13] A. Mood, F. Graybill, and D.Boes. "*Introduction to the theory of Statistics,*" McGraw-Hill, 3rd edition, 1974, pp 243-245.

[14] "Google Toolbar," September 2005, http://toolbar.google.com

[15] Salton, G., and Buckley, C., "Term-Weighting Approaches in Automatic Text Retrieval," *Information Processing & Management, 24(5)*, 1998, pp.513-523.