To appear in *Proceedings of The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD-2001*), August 26-29, 2001, San Francisco, USA.

Discovering Unexpected Information from Your Competitors' Web Sites

Bing Liu
School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543
liub@comp.nus.edu.sg

Yiming Ma
School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543
maym@comp.nus.edu.sg

Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
USA
psyu@watson.ibm.com

ABSTRACT
Ever since the beginning of the Web, finding useful information from the Web has been an important problem. Existing approaches include keyword-based search, wrapper-based information extraction, Web query and user preferences. These approaches essentially find information that matches the user's explicit specifications. This paper argues that this is insufficient. There is another type of information that is also of great interest, i.e., unexpected information, which is unanticipated by the user. Finding unexpected information is useful in many applications. For example, it is useful for a company to find unexpected information about its competitors, e.g., unexpected services and products that its competitors offer. With this information, the company can learn from its competitors and/or design counter measures to improve its competitiveness. Since the number of pages of a typical commercial site is very large and there are also many relevant sites (competitors), it is very difficult for a human user to view each page to discover the unexpected information. Automated assistance is needed. In this paper, we propose a number of methods to help the user find various types of unexpected information from his/her competitors' Web sites. Experiment results show that these techniques are very useful in practice and also efficient.

Keywords
Information interestingness, Web comparison, Web mining.

1. INTRODUCTION
The Web is increasingly becoming an important channel for conducting businesses, for disseminating information, and for communicating with people on a global scale. This is not only true for businesses, but also true for individuals. More and more companies, organizations, and individuals are publishing their information on the Web. With all this information publicly available, it is natural that companies and individuals would like to find useful/interesting information from these Web pages.

Existing research has proposed many approaches for web information finding. Most of them are also widely used. These approaches include keyword-based search, manual browsing, wrapper-based information extraction, Web queries, and user preferences. Keyword-based search [e.g., 5] using search engines, such as Yahoo, Alta Vista, Google, and others, is perhaps the most popular method. A search engine allows the user to specify some keywords, and the system then finds those Web pages that contain the keywords. In manual browsing, a Web browser such as Netscape or Internet Explorer is used by the user to browse specific Web pages to manually find interesting information. Wrapper-based approaches [e.g., 2, 13, 9, 15] enable the user to extract specific pieces of information from targeted Web pages. Web query languages [e.g., 19, 12, 6] give the user the opportunity to query the Web. In the user preference approach [e.g., 26], information is given to the user according to his/her preferences. In essence, all these approaches are based on explicit specifications of the user. They are only able to find information that matches the user's specifications.

The drawback of these approaches is that it is hard for the user to find unexpected information. They can only help the user find *anticipated information* because what the user specifies can only be derived from his/her existing knowledge space. In this paper, we argue that finding only what the user explicitly specifies is not sufficient. Those pieces of information that have not been specified by the user may also be of great interests. It is just that the user does not know about them, or has forgotten about them. Such information may be unexpected and can be of great importance in practice. For instance, it is important for a company to know what it does not know about its competitors, e.g., unexpected services and products that its competitors offer. With this information, the company can learn from its competitors and/or design counter measures to improve its competitiveness. Such business intelligence information is increasingly becoming crucial to the survival and growth of any company. Existing web information extraction techniques cannot find such unexpected information, as it is unlikely (or impossible) for one to specify something that one has no idea of.

Currently, to find such information the user has to manually browse the Web pages of the competitors. However, manual browsing of every Web page can be very time consuming because a typical commercial Web site can have hundreds of pages or more. There may also be many relevant sites (i.e., many competitors) to be analyzed. Automated assistance is thus needed.

Using a computer system to find unexpected information from a Web page (or site) is not a simple task. A piece of information can be unexpected to one company (or user), but not unexpected to another. Hence, whether a piece of information is unexpected is essentially subjective. It depends on the current operations of the company (or the interests of the user), and what it (or he/she) already knows about the competitors.

This problem is analogous to the interestingness problem in data mining [e.g., 21, 25, 17, 18, 20], which is described as follows: Many data mining algorithms often generate a large number of rules from a database, and most of the rules are actually not interesting to the user. But due to the large number of rules, it is very difficult for the user to inspect them manually to identify those truly interesting ones. In the context of the Web, the situation is similar. With a large number of Web pages, finding interesting or unexpected information manually is also a difficult task. In this paper, we propose an approach to help the user find unexpected information from his/her competitors' Web sites.

1.1 Unexpectedness of Information

In the context of the Web, the unexpectedness of a piece of information is defined as follows:

Unexpectedness: A piece of information is *unexpected* if it is relevant but unknown to the user, or it contradicts the user's existing beliefs or expectations.

Note that in this definition, the condition "it is relevant" is important. Not every piece of unknown information is interesting. A piece of information must first be relevant to the user. For example, if the user is a marketing executive, in his/her professional capacity he/she will not be interested in a piece of information on how to plant a tree, although the piece of information may well be unknown to him/her. However, a piece of information on a new marketing strategy will certainly be of interest as it is relevant and unknown to him/her.

1.2 Summary of the Proposed Approach

The proposed approach aims to find interesting/unexpected information from a Web site, which can be a company site (e.g., the site of one's competitor) or a personal home page. This Web site is supplied by the user, and is relevant to the user. We call this site *the competitor site*. From the definition above, we see that to find unexpected information from a Web site, we need to know the user's expectations (or existing knowledge) about the site. From these expectations, the system can find unexpected information from the site. In this work, we use the information contained in the user's own Web site (which can be a company site or the user's home page) and some additional knowledge of the user about the competitor as the existing knowledge or expectations. Our task of finding interesting information from the competitor site thus becomes a problem of comparing the two sites to find similar and different information.

Using the user's own site to represent part of the user's existing knowledge or expectations is appropriate as it allows him/her to find similar and different (or unexpected) information that exist in the competitor site, which is what one is often interested in.

The basic idea of the proposed approach is as follows: Given a user site U, a competitor site C, and some existing knowledge or expectations E about the competitor from the user, our system (called WebCompare) first analyzes U to extract all its key information. It then analyzes C, and compares the information contained in C with that in U and E to find various types of unexpected information from C.

In this work, we only use the textual information in a Web page. Two schemes are used to represent the information in the page [1]:

(1) Vector space representation: This representation is commonly used in the information retrieval community [23, 24, 3, 8]. The assumption of the representation is that similarities, differences and the main contents of text documents can be represented by *keywords* (also called *index terms*) that appear in the documents.

(2) Concepts: These are combinations of keywords that occur frequently in the sentences of a Web page. Such word combinations often represent significant information that the owner wants to emphasize.

In this research, we have designed a number of methods to compare two Web sites to help the user find different types of unexpected information. The proposed techniques are general. They can be used in any domain and for any application. Our implemented system (WebCompare) is also highly interactive due to its efficiency. So far, a number of experiments and application tests have been performed. It is shown that the proposed techniques are useful in practice.

2. RELATED WORK

Although there are many existing techniques that help one find useful information from the Web, to the best of our knowledge, there is still no technique that helps one find unexpected information. Existing approaches to finding useful information on the Web all focus on what the user wants or specifies explicitly. These approaches include keyword-based search, wrapper-based information extraction, user preference specifications, Web and XML queries, and resource discovery. In keyword-based search, the user specifies some keywords, and a search engine (e.g., Yahoo, Excite, Alta Vista, and Google) finds those Web pages that contain the keywords, and ranks them according to various measures. In Web information extraction [e.g., 2, 13, 9, 15, 8], a wrapper or a specific extraction procedure is built automatically or manually for a Web page to extract some specific pieces of information requested by the user, e.g., extracting the prices of some products. User preference based approaches are commonly used in push type of systems [e.g., 26], where the user specifies what categories of information are interesting to him/her. The system then gives him/her only those types of information in the user-specified preference categories. In Web query based approaches, database query language such as SQL is extended and modified so that it can be used to query semi-structured information resources, XML documents and Web pages [e.g., 19, 12, 6, 8]. Web resource discovery aims to find resources (Web pages) related to the user requests [e.g., 16, 7, 8, 9, 10, 13]. This approach uses techniques such as link analysis, and text

[1] So far, we have not given extra considerations to metadata [11, 22] and hyperlinks [16], which will be studied in our future work. Even without such considerations, the proposed methods are already showing very promising results.

classification algorithms to find relevant pages. The pages can also be grouped into authoritative pages, and hubs.

All these existing approaches essentially view the process of finding useful information from the Web as a *query-based process*, although the queries may be of different forms, search query, information extraction query, preference query, Web, XML or semi-structured data query, and resource query. These approaches suffer from the following problems.

1. It is hard to find unexpected information. They can only find anticipated information because queries can only be derived from the user's existing knowledge space. Yet, a lot of information that does not meet the user's requirements or queries may also be of interest to the user. These pieces of information are often unexpected and/or novel.
2. The user often does not know or is unable to specify completely what interest him/her. He/she needs to be stimulated/reminded. The above approaches do not actively perform this task as they only return the requested information. In contrast, our approach also gives those pieces of unanticipated information. If some of them are not truly unexpected, they serve to remind the user what he/she has forgotten, and to stimulate his/her thoughts.

In summary, the proposed technique not only helps the user identify the required information, but also helps him/her find different types of unexpected information. The user is thus exposed to more possible interesting aspects of the Web pages rather than only focusing on his/her current interests (which he/she may not be completely sure).

Our work is related to the interestingness research in data mining. The issue of interestingness of discovered rules has long been identified as an important problem [21, 25, 17, 18, 20]. This is due to the fact that data mining algorithms often produce too many rules, and most of the rules are of no interest to the user. A number of approaches [21, 25, 17, 18, 20] have been proposed to help the user deal with the problem. These approaches are, however, not suitable for the Web. The reason is that rules are structured and have clear syntax and semantics, while information on the Web is semi-structured. Different methods are thus needed for finding unexpected/interesting information from Web pages.

3. VECTOR SPACE REPRESENTATION

In this work, we only use textual information in a Web page to find interesting information. Each page is thus treated as a text document. One of the representation schemes that we employ is the widely used *vector space* model. Here, we review this model.

In vector space representation, each document is described by a set of keywords called *index terms* (or simply *terms*). An index term is simply a word whose semantics helps to remember the document's main themes. An index term is also associated with a weight. Let p be the number of index terms in the collection of documents, and k_i be an index term. $K = \{k_1, k_2, ..., k_p\}$ is the set of all index terms. A weight $w_{i,j} > 0$ is associated with each term k_i of a document d_j. For an term that does not appear in the document d_j, $w_{i,j} = 0$. A document d_j is represented with an index term vector $\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{p,j})$. In information retrieval, the objective is to find a set of relevant or similar documents of a given query document (which is also represented as a vector).

Let the query vector be $\vec{q}$. The similarity of $\vec{q}$ and $\vec{d_j}$ can be computed using the popular cosine measure

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{p} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{p} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{p} w_{i,q}^2}}$$

A number of schemes have been proposed for assigning weights to index terms. Perhaps, the most widely used weighting scheme is the TF-IDF scheme [23, 3], where TF is the *term frequency* and IDF is the *inverse document frequency*.

Definition: Let N be the total number of documents in the system and n_i be the number of documents in which the index term k_i appears at least once. Let $f_{i,j}$ be the raw frequency (or count) of term k_i in document d_j. Then the normalized frequency $tf_{i,j}$ of term k_i in document d_j is given by:

$$tf_{i,j} = \frac{f_{i,j}}{\max_l f_{l,j}}$$

where the maximum is computed over all terms that appear in document d_j. If term k_i does not appear in d_j then $tf_{i,j} = 0$. idf_i, inverse document frequency for k_i, is given by:

$$idf_i = \log \frac{N}{n_i}$$

The TF-IDF term weighting scheme uses the following computation:

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{n_i}$$

For query term weights, Salton and Buckley [24] suggest

$$w_{i,q} = \left(0.5 + \frac{0.5 \times f_{i,q}}{\max_l f_{l,q}}\right) \times \log \frac{N}{n_i}$$

where $f_{i,q}$ is the raw frequency of the term k_i in the query document q.

4. THE PROPOSED TECHNIQUES

We are now ready to present the proposed methods. We will use both the vector space model and keyword combinations (or concepts) to represent the text in a Web page. Below, we first discuss our comparison methods based on these representations. We then describe how the user's previous knowledge can be incorporated in the comparison process.

4.1 Comparing Two Web Sites

We have designed 5 methods to compare the user site U and the competitor site C to help the user find various types of interesting and/or unexpected information from the competitor site. Let $U = \{u_1, u_2, ..., u_w\}$ be the set of pages in the user's Web site, and $C = \{c_1, c_2, ..., c_v\}$ be the set of pages in the competitor's Web site. The 5 methods are discussed below.

1. **Finding the corresponding *C* page(s) of a *U* page**: Here, the user is interested in finding some pages in C that are similar to a page in U. This is useful when the user wants to perform detailed analysis on a specific topic. For example, a newspaper site that has published an article on a particular topic may want to know what has been published on a competitor's site on the same topic.

Our comparison is done as follows: Given a U page u_j, we use the cosine measure to compute the similarity between u_j and each page in C. After the comparison, the pages in C are ranked according to their similarities in a descending order.

Example: In this example, we have 4 pages in U and 3 pages in C, which are shown in Figure 1 (the number in each pair is the raw frequency count of the keyword in the page).

U pages: Upage 1: (data, 1), (predict, 1)
Upage 2: (information, 2), (extraction, 1), (data, 2)
Upage 3: (classify, 2), (probability, 2)
Upage 4: (cluster, 2), (segment, 1)

C pages: Cpage 1: (data, 2), (predict, 2), (classify, 3)
Cpage 2: (association, 3), (mine, 2), (rule, 1)
Cpage 3: (cluster, 3), (segment, 2), (data, 2)

Figure 1: Example U pages and C pages

If we want to find the corresponding page(s) of Upage 1, we obtain the following ranking:

Rank 1: Cpage 1
Rank 2: Cpage 3

Cpage 2 is not shown in the ranking as its similarity value with Upage 1 is 0.

Time complexity: This computation consists of two steps. The first step computes the weight for each term in each C page. Let the maximal number of terms in c_i be G. Computing all the weights takes $O(G|C|)$ time, where $|C|$ is the size of the set C. The second step is the similarity computation, which takes $O(|u_j||C|)$ time (without considering the final ranking). We store the term weights of each page c_i in a hash table. It is reasonable to assume that accessing each term in a hash table takes $O(1)$ time. Thus, the whole operation takes $O(G|C| + |u_j||C|)$ time.

2. **Finding unexpected terms in a C page with respect to a U page:** In many cases, the user wants to know unexpected terms given two similar pages, a C page and a U page (e.g., corresponding pages in method 1 above). These terms allow the user to obtain key differences of the two pages. These differences can help him/her decide whether to browse the C page to find further details.

Given a U page u_j and a C page c_i, we compare the term weights in both documents to obtain those unexpected terms in c_i with respect to the terms in u_j. The unexpectedness value of each term k_r in c_i with respect to u_j, denoted by $unexpT_{r,i,j}$, is computed with:

$$unexpT_{r,i,j} = \begin{cases} 1 - \dfrac{tf_{r,j}}{tf_{r,i}} & tf_{r,j}/tf_{r,i} \leq 1 \\ 0 & otherwise \end{cases}$$

where $tf_{r,i}$ is the tf value of the rth term in c_i, and $tf_{r,j}$ is the tf value of the same term in u_j. $unexpT_{r,i,j}$ is set to 0 if $tf_{r,j}/tf_{r,i}$ > 0, as we are not interested in how a term is unexpected in u_j with respect to c_i. Note that the idf value is not used here as it gets canceled. The algorithm for computing $unexpT_{r,i,j}$ is straightforward, and will not be discussed here.

After the unexpectedness value for each term k_r is computed, all the terms in c_i are ranked according to their $unexpT_{r,i,j}$ values in a descending order. Note that if we reverse the ranking, those top ranking terms are the most expected terms.

Let us continue with our example. We are interested in unexpected terms in Cpage 1 with respect to Upage 1. Note that in the example for method 1, we identified that the two pages are similar. Here, we want unexpected terms. We obtain the following ranking:

Rank 1: classify

The unexpectedness value for term *classify* is 1, as it is not in Upage 1. The terms *data* and *predict* are not shown in the ranking as their term unexpectedness values are both 0.

Time complexity: Without considering the final ranking, the computation is linear in the number of terms in c_i.

Note that this method can be performed between any subset U_{sub} of the pages in U and any subset C_{sub} of the pages in C. For instance, the user may be interested in the comparison of product related pages in both sites. To perform this comparison, we first combine the terms in U_{sub} and the terms in C_{sub} respectively. The combination is done as follows: For each term we sum up the counts of the term in all the pages of U_{sub} (or C_{sub}). After that, we apply the above formula.

3. **Finding unexpected pages in C with respect to U:** The aim of this method is to find the pages in C that are most unexpected with respect to the U site. These pages are often very interesting, as they tell the user that the competitor site may have some useful pages that the user site does not have.

For this comparison, we first combine all the pages in U to form a single document D_u, and all the pages in C to form another single document D_c. We then compute the unexpectedness value of each term k_l in D_c with respect to D_u, i.e., $unexpT_{l,c,u}$. The unexpectedness of a page $c_i \in C$ with respect to U, denoted by $unexpP_i$, is defined as the amount of term unexpectedness contained in c_i. Let the set of terms in c_i be $\{k_1, k_2, \ldots, k_m\}$. $unexpP_i$ is computed as follows:

$$unexpP_i = \frac{\sum_{r=1}^{m} unexpT_{r,c,u}}{m}$$

It is important to note that it is not satisfactory to compute term unexpectedness based on individual pages as information on a topic in a Web site can be concentrated in one page, or distributed over a set of pages. It is thus more appropriate to measure the unexpectedness of a term using aggregate contents of the Web sites, D_c and D_u, for the purpose of finding unexpected pages [2].

After all $unexpP_i$ values are computed, we rank C pages according to their $unexpP_i$ values. Using the C and U pages in Figure 1, we obtain the following ranking:

Rank 1: Cpage 2
Rank 2: Cpage 3
Rank 3: Cpage 1

Cpage 2 is very unexpected with respect to D_u. Such a page is often very useful.

[2] We have also studied a few other formulations for computing $unexpP_i$ in this work but with less satisfactory results.

Time complexity: This computation consists of two steps. The first step combines the terms in all U pages and in all C pages to form D_u and D_c respectively. Let the maximal number of terms in a U page be M_u, and the maximal number of terms in a C page be M_c. Then, the time complexity of the computation is $O(M_u|U|+ M_c|C|)$. The second step computes all $unexpP_i$. The time complexity of this computation is $O(M_c|C|)$. Without considering the final ranking, the computation complexity of the whole operation is $O(M_u|U|+ M_c|C|)$. Again, we assume that the terms in each page (including D_u) are stored in a hash table.

4. **Finding unexpected concepts in a *C* page with respect to a *U* page:** In many cases, keywords alone may not reveal some important information of a Web page. A combination of words or *concept* may be very informative. For example, in a page we find 100 keywords. Within the 100 words we have the words "information" and "extraction". The two words may be placed quite far from each other. If we see them separately among the 98 other words, we may not notice anything interesting. However, if we know that the two words occur together frequently in the page, it reveals something significant, and may be worth investigating.

   Definition: A concept is a set of keywords that occur together in the sentences of a page above a certain user-specified minimum support (or frequency).

   According to this definition, the words sequence in a sentence does not matter. The words also do not have to be next to one another. This definition is reasonable, as we believe it has the advantage of finding more interesting keyword combinations. For example, "information extraction", "extraction of information" and "information is extracted" basically express the same concept of "information extraction".

   In this work, we make use of an existing algorithm for association rule mining [1] to discover frequent word combinations in a page. Association rule mining is defined as follows [1]: Let $I = \{i_1, \ldots, i_n\}$ be a set of *items*, and T be a set of transactions (the dataset). Each transaction consists of a subset of items in I. An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The rule $X \rightarrow Y$ holds in T with confidence c if $c\%$ of transactions in T that support X also support Y. The rule has support s in T if $s\%$ of transactions in T contains $X \cup Y$. The problem of mining association rules is to generate all association rules in T that have support and confidence greater than the user-specified minimum support and minimum confidence.

   A typical association rule algorithm works in two steps: In the first step, it finds all *frequent* itemsets (a set of items, or a set of terms in our context) from the data that satisfy the user-specified minimum support requirement. These itemsets are concepts in our application. In the second step, it generates all if-then rules from frequent itemsets. We do not need this step, as in our application we are only interested in those frequently occurring keywords or terms.

   We use the well-known Apriori algorithm [1] for association mining to discover all concepts. In our context, keywords in each sentence of a page form a transaction. The set of all sentences in the page gives the dataset. If a keyword occurs more than once in a sentence, we consider it only once.

   After all concepts are discovered, the same comparison as that in method 2 above is performed. Here, we simply treat each concept as a term or keyword. We can also use concepts (instead of individual terms) to find unexpected pages in C with respect to U using a similar formulation as in method 3.

   It is worth noting that we mine frequent itemsets from each page in C (or in U) separately. We do not concatenate all pages in C (or U) into one dataset and mine concepts from it. The reason is that each page of a Web site typically focuses on a specific topic. If we mix it with other pages, we may not be able to find some interesting concepts in a page because of the minimum support requirement. For example, a concept may be frequent in one page, but may not be frequent when it is combined with another page as the minimum support is normally specified in percentage.

5. **Finding unexpected outgoing links:** This method finds all outgoing links from the C site. This is useful as these links may indicate some useful resources that are of additional help to their customers. For example, a travel company may have links to sites that provide currency exchange rates, and sites that offer international weather information.

   The outgoing links can be collected by the crawler during crawling (see Section 5). Let the set of outgoing links from U be L_u and the set of outgoing links from C be L_c. The set of unexpected outgoing links in C with respect to U is $L_c - L_u$, which are the links that occur in C but not in U.

4.2 Incorporating User's Existing Knowledge

In almost all situations, the user has some existing knowledge about the application domain and its competitors. Our system allows him/her to express this knowledge, which is then used in comparison. User's existing knowledge serves two purposes:

1. It allows the system to discover truly unexpected information.
2. It allows the user to check whether his/her expectations are correct. For example, the user may expect that a particular data mining company provide clustering tools. However, it may turn out that the company does not have any such tools.

In our framework, user's knowledge is also expressed as keywords, concepts, and hypertext links. Let E be the set of user-specified keywords, concepts and links. E consists of two parts, E_g, and E_s. E_g contains all the general items (keywords, concepts or links) of the domain that the user knows about and does not want them ranked high. E_s contains specific items of the site that the user knows about and does not want them ranked high.

Note that the two sets, E_g, and E_s, are quite different. E_g contains those common items of a domain, which the user does not want them ranked high. E_g can be reused in comparison with any site of the application domain. For example, in the travel domain, "departure" and "date" are very common. They should not be ranked high in any situation. Thus, they should be put in E_g. In the case of a particular site, the user may know some items in it and he/she does not want them ranked high. The user can put them in E_s. For example, if the user knows that a travel company offers "free and easy" tours, he/she can put "free and easy" in E_s in comparing the site. "free and easy" may not be put in E_g because

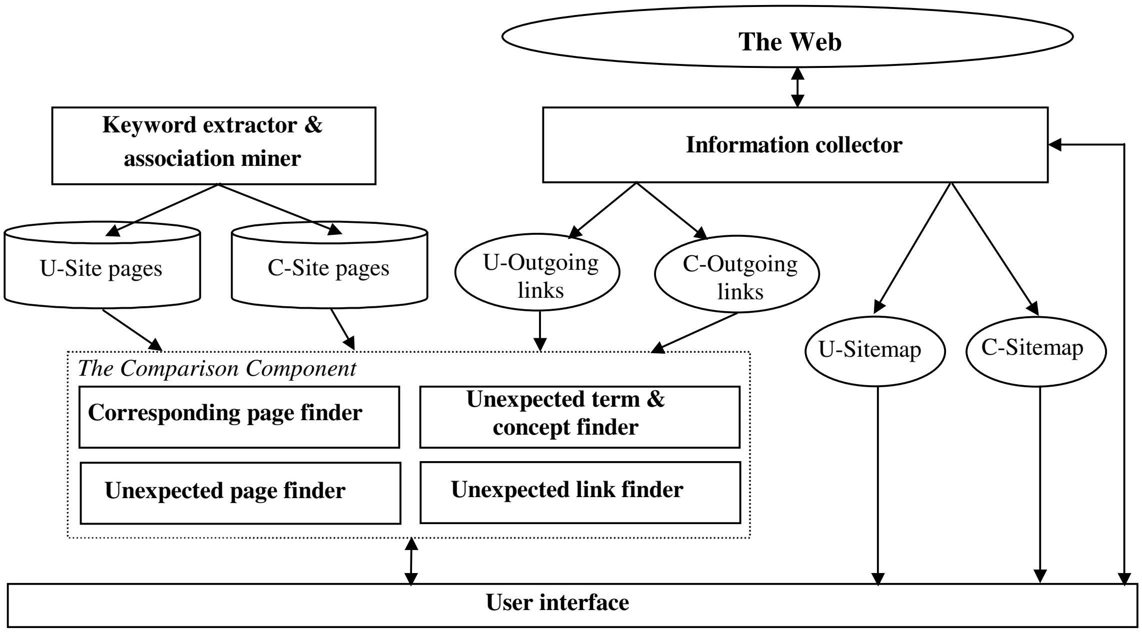

Figure 2: The WebCompare system architecture

not all companies offer such a service, and the user may want to know those that do and those that do not. E_s is only applicable to the pages of the particular *C* site.

In computation, items in *E* are added to the set of items in *U*. Keywords in *E* are used in method 2 and 3, concepts in method 4 and outgoing links in method 5. When a weight is needed for an item, it takes the maximum weight, as we want it to be ranked the lowest and/or to be least significant in comparison. Note that *E* is not used in method 1, as we believe that similarities should be computed objectively. *E* reflects subjective knowledge of the user.

5. SYSTEM ARCHITECTURE

We have implemented a web site comparison system based on the proposed methods. The system is called WebCompare, coded in Visual C++ under the Window's environment. It consists of 4 main components (the system architecture is shown in Figure 2):

1. A multi-thread information collector: It crawls a Web site to download all its pages. During the process, it also records those outgoing links. After crawling, a sitemap is built to allow the user to choose pages for comparison.
2. A keyword extractor and an association miner: It extracts keywords from a Web page, and performs the standard operations of eliminating *stopwords*, and word *stemming*. We use the Smart system [23] for this purpose. After keywords are extracted from a page, the association miner is executed to find all the concepts from the page.
3. A comparison component: It uses the functions discussed in Section 4 to analyze pages from the user site and the competitor site to help the user find various types of unexpected information. (Note that unexpected term and concept finders are combined into one).
4. A user interface: It allows the user to interact with the system.

6. A RUNNING EXAMPLE

We now use a running example to show the working of the system. In this example, we compare two sites, our own DM-II data mining site, http://www.comp.nus.edu.sg/~dm2, and the SGI's MineSet data mining site, http://www.sgi.com/software/mineset.

The crawling interface of our system is shown in Figure 3. The upper part is the starting page of DM-II. Its sitemap is on the right. The lower part is the corresponding information of MineSet. We can click on any link from the two sites for comparison. We can also delete irrelevant links so that they will not appear in the comparison. Throughout the example we use DM-II as the user (or *U*) site and MineSet as the competitor (or *C*) site.

To start comparison, we first select sites (or pages) as the user and competitor sites (or pages). After that, we obtain a screen (not shown here) that has all the comparison functions discussed in Section 4. We can click different buttons for different comparisons. Our system is also connected to Microsoft Excel for reporting the comparison results.

Let us start to compare the two sites. Note that our existing knowledge about data mining is not used in this example.

- **Finding unexpected pages in *C* with respect to *U***: Here we want to find those most unexpected pages in MineSet with respect to DM-II. The resulting ranked pages are given in Table 1 (only the top 15 pages are shown). We can see that these pages are MineSet's documentation pages. This is quite interesting to us, as we do not have any html documentations for our system. We plan to add such pages to our site to help our users.
- **Finding corresponding *C* pages of a *U* page**: We now want to focus on a specific topic for detailed analysis. We are interested in technology comparison of MineSet and DM-II. Our technology page is the research project page,

Figure 3: The crawler interface of WebCompare

http://www.comp.nus.edu.sg/~dm2/research_proj.html. We would like to know where the technology pages of MineSet are. After execution, we obtained the ranked MineSet pages in Table 2. Again only the top 15 pages (their URLs and similarity values) are shown.

These top pages indeed describe the MineSet technologies. Although the MineSet job page (ranked 2) does not specify its technologies, their personnel needs do reflect their technologies. Pages ranked 1 and 3 clearly describe the technologies of MineSet. These pages were all unknown to us before. This shows our system is able to allow the user to quickly focus on those interesting pages.

- **Finding unexpected terms and concepts in a *C* page with respect to a *U* page:** After finding corresponding *C* pages (normally a few pages) of a *U* page, the user typically wants to know what are unexpected terms and concepts so that he/she can decide whether or not to perform further analysis by browsing the pages. Let us continue with our example. We would like to know what are unexpected terms and concepts in the first page of Table 2, i.e., churn.html, with respect to our research_proj.html page.

  The top ranking keywords and concepts found in churn.html are given in Table 3 (only the top 15 items) [3]. Since these terms and concepts do not exist in our page, they are ranked based on their frequency counts in churn.html. After scanning through the whole list (including those not shown in Table 3), we found nothing interesting, as we were very familiar with the topic. Hence, we did not need to browse the page.

  However, when we compared moreproductdetails.html with our research_proj.html page. We found many unexpected/interesting concepts (see the top ranking 15 keywords and concepts in Table 4). Although we knew that MineSet had very good visualization capabilities, *voice-annotation* and the *splat visualizer and scatter visualizer* were quite unexpected to us. This page warranted further analysis. From the page, we found many interesting features and capabilities of MineSet that we were not aware of before.

  Note that for concepts, which are frequent itemsets, we only report the longest itemsets, but not their subsets. A longest itemset is defined as a frequent itemset that is not a subset of any other frequent itemset. A longest itemset is also called a border in association rule mining [4]. Each longest itemset summarizes its subsets. For example, if we have the longest itemset {a, b, c, d}, we will not report any subset of it, although every subset of it must be frequent [1, 4]. This reduces the number of concepts, which facilitates user inspection. We are aware that there are algorithms for finding long itemsets directly from the data [e.g., 4]. We will consider implementing one of them in the future.

[3] The keywords here are given in full words rather than their stems, which can be hard to understand in some cases.

Table 1: The most unexpected *C* pages

	URLs	UnexpPi
1	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-15.html	0.734
2	http://www.sgi.com/software/mineset/docs/tut/MineSetNT_T-1.html	0.731
3	http://www.sgi.com/software/mineset/docs/ref/MineSet_Ref-9.html	0.726
4	http://www.sgi.com/software/mineset/docs/tut/MineSetNT_T-7.html	0.723
5	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-9.html	0.712
6	http://www.sgi.com/software/mineset/docs/ref/MineSet_Ref-7.html	0.709
7	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-8.html	0.695
8	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-9.html	0.687
9	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-1.html	0.683
10	http://www.sgi.com/software/mineset/docs/ref/MineSet_Ref-1.html	0.665
11	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-11.html	0.656
12	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-10.html	0.643
13	http://www.sgi.com/software/mineset/docs/ref/MineSet_Ref-10.html	0.623
14	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-16.html	0.621
15	http://www.sgi.com/software/mineset/docs/int/MineSet_Inter-17.html	0.612

Table 2: Pages that are similar to our research_proj.html

	URLs	Sim. Value
1	http://www.sgi.com/software/mineset/tech_info/churn.html	0.1905
2	http://www.sgi.com/software/mineset/jobs.html	0.1861
3	http://www.sgi.com/software/mineset/moreproductdetails.html	0.1608
4	http://www.sgi.com/software/mineset/docs/tut/MineSetNT_T-4.html	0.1593
5	http://www.sgi.com/software/mineset/overview.html	0.1551
6	http://www.sgi.com/software/mineset/docs/tut/MineSetNT_T-3.html	0.1509
7	http://www.sgi.com/software/mineset/dms.html	0.1453
8	http://www.sgi.com/software/mineset/success.html	0.1434
9	http://www.sgi.com/software/mineset/docs/tut/MineSetNT_T-6.html	0.1388
10	http://www.sgi.com/software/mineset/training_faq.html#comments	0.1360
11	http://www.sgi.com/software/mineset/training_faq.html	0.1360
12	http://www.sgi.com/software/mineset/docs/ref/MineSet_Ref-4.html	0.1352
13	http://www.sgi.com/software/mineset/mineset_data.html	0.1306
14	http://www.sgi.com/software/mineset/tech_info/billing.html	0.1301
15	http://www.sgi.com/software/mineset/docs/ref/MineSet_Ref-9.html	0.1249

Table 3: Unexpected keywords and concepts in churn.html

	Keywords	No.	Concepts	No.
1	customer	60	customer churn	6
2	churn	56	customer profile	6
3	management	34	telecommunication company	6
4	warehouse	29	high performance	5
5	analyst	25	referential integrity	4
6	information	23	segment costomer base	4
7	technology	22	churn model	3
8	visualizer	19	consistent format	3
9	relate	18	cost effectively	3
10	telecommunication	15	effective churn management	3
11	understand	12	hidden pattern trend	3
12	time	11	high value customer	3
13	large	10	hundred million	3
14	profile	10	large volume	3
15	trend	10	long distance	3

Table 4: Unexpected keywords and concepts in moreproductdetails.html

	Keywords	No.	Concepts	No.
1	visualizer	66	click voice annotated	4
2	tree	22	option tree	4
3	map	21	pie chart	4
4	animate	19	size color	4
5	display	16	visualizer movie	4
6	dimension	15	visual tool	4
7	bar	14	bar chart	3
8	color	14	height color	3
9	evidence	13	people earn	3
10	relate	13	record view	3
11	analysis	12	splat visualizer scatter	3
12	show	12	united states	3
13	information	11	add tool analysis	2
14	pointing	11	animation control panel show	2
15	variable	11	animation control panel tracce	2

- **Finding unexpected outgoing links:** We found 31 outgoing links in MineSet that we were not aware of before. Table 5 lists 15 of them. We have decided to include some of these links (e.g., 4, 5, 8 and 10) on our own Web site. Although the old article, jul98_50.htm, is no longer there, the site http://www.dmreview.com still contains a lot of useful data mining related information. Site 7 is also interesting as it has all the standard benchmark data sets for classification research. Although we knew this site before, we did not point to it. We actually used many of the data sets there to compare the accuracy of our system with some existing systems. Thus, we should point to this site.

Table 5: Unknown links to outside

1	http://www.cutter.com/copyrigh.htm
2	http://www.cutter.com/dms/index.html
3	http://www.data-miners.com/
4	http://www.dmreview.com/issues/1998/jul/articles/jul98_50.htm
5	http://www.dmreview.com/master.cfm?NavID=71&EdID=568
6	http://www.hummingbird.com
7	http://www.ics.uci.edu/~mlearn/MLRepository.html
8	http://www.informationweek.com/753/datamine.htm
9	http://www.pkware.com/shareware/pkzip_win.html
10	http://www.stattransfer.com/
11	http://www.twocrows.com
12	http://www.techweb.com/se/directlink.cgi?IWK19980810S0038
13	http://www.vim.org/dist.html
14	http://www.zdnet.com/pcweek/stories/news/0
15	http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=DM10001801

7. EVALUATION

Since the proposed technique deals with subjective interestingness of information, it is difficult to have an objective measure of its performance. There is also no existing system that is able to perform our task. Thus, we could not do a comparison. We have carried out a number of experiments involving domain users to check whether the system is useful in practice. In this section, we first discuss our application experiences and then report experiment results on the time efficiency of the system.

7.1 Application Experiences

Our users are from three different organizations: a travel company, a private educational institution, and a diving company. Each of them compared their company site with a competitor site.

Before using our system, our users all knew something about their competitors, and had browsed their competitor's Web pages before. Interestingly, our system helped them find a lot of information that they did not find previously. The main reason is that their competitors' sites all have a large number of pages. For example, the competitor's site of the travel company has more than 250 pages. Our user did not go through many of them before. Our system helped him find many previously undiscovered interesting pages. Many of these pages describe some tourist attractions in a number of countries that he had never heard of (mainly in those newly opened-up countries). The system also helped him find some previously unknown educational travel destinations. Many interesting (unexpected) links to tourist information services, entertainment companies and small hotels at different destinations were uncovered. The user was also a little surprised to find that their competitor guides their customers to some medical care groups in a number of small destinations. This information is of immediately use to the user company. In the cases of education and diving applications, many pieces of unexpected information were also discovered.

Our system helped the users perform much better analysis due to the following reasons:

- It allowed them to quickly focus on those potentially interesting pages, terms and concepts. Without the system, it was very tedious to browse through many pages to fish for something interesting.
- Due to the difficulties of manual analysis, the users often gave up after browsing some top-level pages. They mainly used anchor texts as a guide to decide whether to visit deeper pages. This heuristic frequently results in many interesting pages not visited. Our system was able to help them perform a more complete analysis of a site.
- If a page is long, the users often do not read it carefully, and thus may miss some useful information. Our system summarizes each page with keywords and concepts, which is much smaller in number and thus easier to inspect manually.
- Our system is able to attract the users as it always provides something unexpected, which can be quite tempting, i.e., people are always curious about unexpected things.

7.2 Efficiency

If we assume that the length (i.e., the number of keywords) of a Web page is constant, all our algorithms are essentially linear in the number of pages involved. It is reasonable to assume the length of a Web page as constant because the average length of a page does not vary a great deal from one site to another.

We have performed many experiments using a number of Web sites. All the experiments were run on a Pentium II 350 PC with 64MB of memory. Table 6 shows the running times for computing different values using 5 Web sites. The time for association mining in each page is given in the final column. Note that association mining is only performed once for each page at the beginning. The results are used in each subsequent comparison.

Since the time complexities of all algorithms are basically linear in the number of pages, Table 6 gives the average running time (in sec.) per page for computing different values. Column 1 lists the URLs used as the *C* sites. The *U* site is our DM-II site. The number next to each URL is the number of pages from the site (dead links are not counted) used in our experiments. Column 2 gives the average execution time per *C* page in computing cosine similarity measures for finding corresponding *C* pages of a *U* page. Column 3 gives the average time for computing $unexpT_{r,i,j}$ for a *U* page and a *C* page (both keywords and concepts comparisons are done at the same time). Column 4 gives the average time for computing $unexpP_i$ per page. The final column gives the average execution time for mining concepts per page

Table 6. Execution times

		sim.	unexpTr,i,j	unexpPi	assoc. mining
1	http://www.bluemartini.com/ (143)	0.0128	0.0156	0.0232	0.0379
2	http://www.datamining.com/ (21)	0.0134	0.0189	0.0213	0.0182
3	http://www.mineit.com/ (66)	0.0113	0.0177	0.0198	0.0206
4	http://www.sgi.com/software/mineset/ (127)	0.0097	0.0201	0.0224	0.0115
5	http://www.spss.com/clementine/ (46)	0.0143	0.0153	0.0188	0.0105

using association mining. Here, the minimum support is set to 1%. Since a small page may have fewer than 100 sentences, then any keyword combination will form a frequent itemset. Hence, we also use a minimum count threshold. In our experiment we set it to 2, i.e., any frequent itemset must appear in a page at least two times. Note that we have also experimented with other sites as *U* sites. The average execution times per page are roughly the same.

From Table 6, we observe that all the computations can be done very efficiently. The system can easily handle a large number of pages in a short time. The time for crawling is not reported (which is done only once), as its efficiency depends on a number of factors, which are well understood. For keyword extraction, we used the Smart system [23]. Outgoing links of each site are recorded during crawling. Comparing the outgoing lists of two sites is straightforward and fast.

8. CONLUSION

In this paper, we argued that only finding information that matches the user's specifications is insufficient for Web information discovery. In many applications, finding information that the user has no idea of is also of great importance. This paper proposed a number of methods to help the user find unexpected information from his/her competitors' Web sites. Experiment results and real-life applications show that the proposed techniques are very useful in practice and efficient.

In our future work, we will study the use of metadata and ontology to provide more information related to keywords to create a more intelligent system. We will also study how links can be used to infer more unexpected information. This research may be extended as a methodology for monitoring a competitor's Web site. In this context, we can treat the old web pages of the site as the existing knowledge. Any unexpected changes to the old pages by the competitor will be reported to the user.

9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. VLDB-94, 1994.

[2] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. SIGMOD Record, 26(4), 1997.

[3] R. Baeza-Yayes, and B. Ribeiro-Neto. Modern Information Retrieval. Addison Wesley, 1999.

[4] R. Bayardo. Efficiently mining long patterns from databases. SIGMOD-98, 1998.

[5] S. Brin, and L. Page. The anatomy of a large scale hypertextual Web search engine. WWW7, 1998.

[6] S. Ceri, S. Comai, E. Damiani, P. Fraternali, and L. Tanca. Complex Queries in XML-GL. SAC (2) 2000: 888-893

[7] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. WWW8, 1999.

[8] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. SIGMOD-98, 1998.

[9] M. Craven, D. DiPasquo. D. Freitag, A. McCallum, T. Mitchell, K. Nigam and S. Slattery. Learning to construct knowledge bases from the World Wide Web. Artificial Intelligence, 118(1-2), 2000.

[10] J. Dean and M. R. Henzinger. Finding related pages in the World Wide Web. WWW8, 1999.

[11] Dulin Core Home Page, http://purl.org/DC.

[12] D. Florescu, A. Levy, A. Mendelzon. Database techniques for the World-Wide Web: a survey. SIGMOD Record 27(3): 59-74 (1998).

[13] R. Feldman, Y. Liberzon, B. Rosenfeld, J. Schler and J. Stoppi. A framework for specifying explicit bias for revision of approximate information extraction rules. KDD-2000, 2000.

[14] D. Gibson, J. Kleinberg, P. Raghavan. Inferring web communities from link topology. Proc. 9th ACM Conference on Hypertext and Hypermedia, 1998.

[15] T. Guan and K. F. Wong. KPS - a Web information mining algorithm. WWW8, 1999.

[16] J. Kleinberg. Authoritative sources in a hyperlinked environment. ACM-SIAM Symposium on Discrete Algorithms, 1998.

[17] B. Liu, and W. Hsu. Post-analysis of learnt rules. AAAI-96.

[18] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. KDD-97, 1997.

[19] A. Mendelzon, G. Mihaila, T. Milo. Querying the World Wide Web, Journal of Digital Libraries 1(1): 68-88, 1997.

[20] B. Padmanabhan, and A. Tuzhilin. Small is beautiful: Discovering the minimal set of unexpected patterns. KDD-2000, 2000.

[21] G. Piatesky-Shapiro & C. Matheus. The interestingness of deviations. KDD-94, 1994.

[22] Resource Description Framework (RDF) Schema Specification, W3C proposed recommendation. 22 Feb, 1999. http://www.w3.org/TR/PR-rdf-schema/

[23] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[24] G. Salton and C. Buckley. Term-weight approaches in automatic retrieval. Information Processing and Management, 24(5):513-523, 1988.

[25] A. Silberschatz, and A. Tuzhilin, What makes patterns interesting in knowledge discovery systems. IEEE Trans. on Know. and Data Eng., 8(6), 1996.

[26] G. Underwood, P. Maglio and R. Barrett. User-centered push for timely information delivery. WWW7, 1998.