The Web Host Access Tools (WHAT): Exploring Context Sensitive Information Access

Lillian N. Cassel[1], Satarupa Banerjee[1], Arun Narayana[1], Ursula Wolz[2]

[1] Villanova University, Department of Computing Sciences, 800 Lancaster Avenue, Villanova, PA 19085-1699, USA

[2]The College of New Jersey, Department of Interactive Multimedia
2000 Pennington Road,
Ewing, NJ 08628, USA

Abstract. Search engines are an integral part of web based information gathering and retrieval, but they are significantly dependent upon the words or word phrases input by the end user. The search engine usually contributes no additional semantic data toward the information acquisition process. This paper presents an intelligent search agent, the "Web Host Access Tools" (WHAT), which introduces the notion of queries conducted within a specific contextual meaning. Given a context and adaptive associated keywords that personalize to the search history and preferences of the user, WHAT performs more intelligent resource filtering than conventional search engines, providing more relevant results while filtering irrelevant references.

1 Introduction

Web search is a highly refined activity that has become much more accurate in the little more than 10 years of the web's history. A simple Google query will likely produce relevant results. However, a number of problems remain. Web searches remain remote services, organized and operated for masses of users and without history of prior search experiences to guide later work. Users still may repeat a search and find a favorite response missing from the result list. It is difficult to say "give me the new results for this query; things I have not seen before." The Web Host Access Tools (WHAT) project addresses these difficulties in a unique way. The tool runs on an individual's local computer. WHAT allows a user to define any number of contexts, which are areas of frequent search activity. When WHAT recommends search results to the user, it incorporates knowledge of the user's prior searches in that context. The search history allows a user to recover any previously found result, but also contributes to accurate definition of the context, which is used to enhance future related searches. Because WHAT runs on the user's own computer, the context is enhanced specifically for that user's purposes. User feedback on each search feeds analysis that leads to recognition of characteristics of search results that address the broad category

N. Cassel L., Banerjee S., Narayana A. and Wolz U. (2005).
The Web Host Access Tools (WHAT): Exploring Context Sensitive Information Access.
In *Proceedings of the 1st International Workshop on Web Personalisation, Recommender Systems and Intelligent User Interfaces*, pages 34-42
DOI: 10.5220/0001422100340042

of the context and characteristics that are specific to a particular query and have limited relevance to the overall context definition.

This paper introduces the WHAT project, its overall architecture and internal operation, as well as some very early analysis of its effectiveness in connecting the user to the most relevant search results. Current work focuses on introducing support vector machines to extract important characteristics from search results and the user's response to them so that the context definition is more accurate and future query enhancements are more effective.

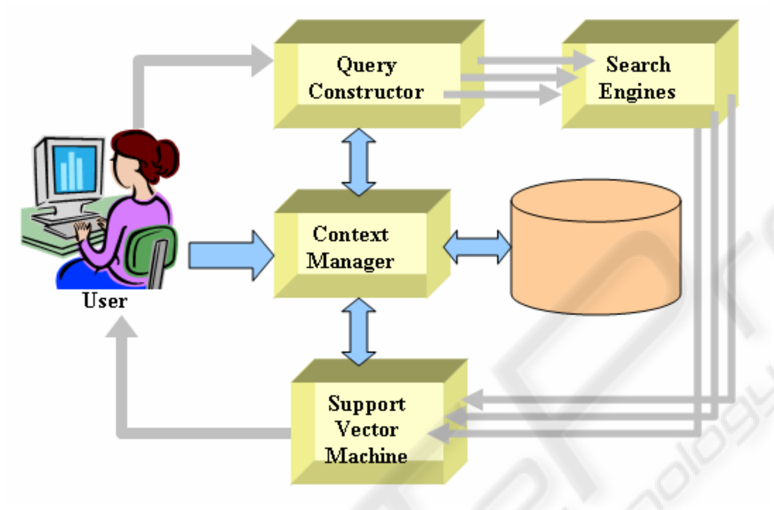

Fig. 1. The block diagram of the W.H.A.T architecture

The tool is implemented by a set of object-oriented Java classes. The user interface begins with an initial window that receives query input and context selection. The query, enhanced with information from prior search experience, is sent to several internet search engines. A second window displays the recommended links, ranked by the WHAT engine. The generated results can then be annotated and rearranged, and recommended URLs can be accessed by links that open the operating system's web browser.

2 WHAT Personalization Context Manager

Before a search is passed through the WHAT engine, the user must select a specific context to apply to the search phrase. Each context is a user-defined categorization that is associated with a number of keywords comprising its semantic definition. These keywords are given weighted values signifying their level of importance to the particular context. Such weights become the key comparisons for relevance calculations in query formulation and result ranking, and they will scale according to techniques employing user feedback and search history. As keyword weights adjust over time, the system is trained to draw inferences from the user about the desired meaning behind user contexts. Since this dynamic information develops on the client machine, these contexts allow the user to conduct more meaningful searches with less effort put into search phrase generation.

To implement this model, the Context Manager stores context categorizations, definition keywords, and search history in a relational database. The relational schema is composed of six 3NF tables, designed by the standard top-down entity-relationship model to relational schema mapping algorithm:

CONTEXT (name, context_number, created)
DEFINITION (context_number, keyword, weight, initial_weight)
SEARCH (context_number, search_string, search_number, last_performed)
RESULT_LOOKUP (search_number, url, result_number)
RESULT_METADATA (result_number, search_engine, title, snippet)
RESULT_USERDATA (result_number, search_query, relevance, ranking, impression, last_visited)

At the highest level, **context** entries are unique by name. Members of the **definition** table are associated with a particular context, thus definitions are distinguished by the concatenated key "context_number, keyword." Likewise, any historic entry in the **search** table is related to a specific context and is ordered by the key "context_number, search_string." Under this table, the "search_number" provides the key for details stored in the **result_lookup** relation, which contains the key for the more specific **result_metadata** and **result_userdata** tables. This model allows the DBMS to manage search strings, URLs, and meta-data which may consist of many long strings. By design, the fundamental engine behind WHAT is not reliant on this specific data storage schema to allow for scalability.

3 Query Constructor and Search Engine Requests

Once the user enters a search phrase and a context is selected, queries are sent to standard search engines (Google, Yahoo, etc.). To take advantage of the context information, the Query Constructor generates and formats multiple search strings for each search engine. Initially, the constructor begins by forming a set of all possible combinations of the keywords for the selected context. The search phrase is then appended to each generated string. However, the number of keyword combinations grows exponentially in relation to the number of keywords, so the constructor selects a reasonable subset of the generated strings for submission. This subset is determined by a user-modifiable parameter called "tolerance." Tolerance is a percentage which is involved in a calculation to filter out less relevant queries. Based on the associated weights of the keywords, each string is compared against the most relevant string generated, and those strings that are significantly irrelevant according to the tolerance factor are removed from the submission set. Thus, the behavior of the Query Constructor is dependent on the client-side personalization of the Context Manager and will improve with continued information gathering.

Submissions sent to search engines provide a dataset for result acquisition. Originally, WHAT interfaced with internet search engines by sending HTTP requests and receiving an HTML response. Using a technique known as text-scraping, these HTML documents needed to be parsed before the significant data could be abstracted. On completion, a successful parse would separate the document from its constituent components such as URL, title, and descriptors. However, using parsers introduces a complication when processing results since each search engine generates documents of different formats. Due to these differences in webpage layouts, individualized parsers for each search engine were developed. Another issue with parsers is that these specific formats themselves change over time. Thus, the parsers needed to be modified whenever the output of a search engine changed.

Another issue with sending HTTP requests involved the Google search engine. Google does not respond to HTTP requests unless the request is received from one of the standard internet browsers. Due to the prevalence of Google in the meta-search engine domain, integrating its capabilities into the framework of WHAT was highly desired. As a result, a new approach was devised for search engine information retrieval.

The new solution for information retrieval involves using the Application Programming Interfaces (APIs) offered by individual search engines to receive the results programmatically. In particular, two different interfaces are implemented currently for both the Yahoo and Google search engines.

3.1 Accessing Google with the SOAP Protocol

To obtain relevant search results from Google, WHAT communicates with the server through the SOAP protocol, which Google offers to programmers implementing Google search services within customized programs. This service is hosted on a different server than the one that handles HTTP requests which is accessed by browsers like Internet Explorer and Netscape Navigator. Using the publicly downloadable WSDL (Web Service Definition Language) file from Google, WHAT creates a client web service project that requests Google search results using the SOAP protocol. The web service client contains Google's definitions for abstract input and output data objects in addition to a proxy program which contains a specific connection URL for the Google server. This approach returns precisely paged results with complete access to additional features like spell checker and cached pages without the overhead of text-scraping.

Since search results for queries to Google are returned as abstract data objects, WHAT does not need to account for future changes to Google HTML documents. Hence, the WHAT engine can focus less on the syntactical issues involved with handling the retrieved information.

3.2 Accessing Yahoo with a Java API

Yahoo's API is slightly different from the SOAP implementation used by Google. Yahoo has developed a specific Web API that can be included in a Java web application project to retrieve their search results. The Java Archive (JAR) file containing the Yahoo API includes a complete Java Doc defining the structure of the package and how to implement the functionality of each class file provided in the JAR. Since Yahoo provides open source code for its Web API, changes were made wherever applicable to improve the efficiency of its implementation within the WHAT framework. Once the compiled JAR files are in place, WHAT's customized proxy can directly communicate with Yahoo web servers. Like Google's servers, requests from Yahoo APIs are handled by different servers than the ones that handle requests from internet browsers. Again, using the Yahoo API causes WHAT to be highly compatible with any future changes or developments Yahoo makes regarding search result delivery.

4 Relevance Calculator and Result Ranking

The Relevance Calculator is responsible for filtering and ranking the results for presentation to the user. After identifying and removing duplicates, an algorithm assigns a ranking value to each result based on occurrences of context-related keywords in titles and descriptors. Keywords with higher weights contribute more to this value than terms of lower weight, and appearances in titles are valued more than occurrences in descriptors. The WHAT engine is not explicitly dependent on any specific algorithm, but a proficient relevance ranking procedure will facilitate successful results. Since ranking considers keyword weights, these results will improve with continued personalization.

After the user has finished viewing a particular website from these results, a dialog prompts the user to rate the site with a simple scale indicating its accuracy to the desired result. This feedback allows keyword weights to be modified accordingly, providing a mechanism for personalization and machine learning. The motivation for the user to spend the small amount of time required to check off an appropriate rating persists with the idea that a small time tradeoff leads to quality results in the long term. Since search results relate to a context with presumed long-term significance, the user is motivated to invest in the accuracy of future search results.

4.1 Support Vector Machine

The task of recognizing characteristics that are common among results relevant to a context, independent of a specific query, falls on a pair of support vector machines (SVM). SVM techniques are used to find characteristics that relate to a context and the results are used to improve the context definition. Secondly, an SVM is used to identify the immediate search results that are most likely to suit the user's immediate search need.

In order to establish a similarity or dissimilarity measure of different words, Latent Semantic Indexing (LSI) [1, 2, 3, 4] has been implemented in the WHAT engine, which helps to construct a semantic vector model based on word co-occurrences. LSI [3, 4] extracts the underlying semantic structure of a text by estimating the most significant statistical factors in the weighted word space. A word or word phrase is represented by a LSI coefficient, which is obtained from key words by removing common endings with porter stemmer algorithms and representing the word in a weighted word space. Initially, the search engine contains a default knowledge base represented in terms of LSI coefficients, where each word has a unique LSI coefficient. A Support Vector Machine (SVM) [5, 6] based classifier algorithm then interacts with a local library of previous search history to analyze search results that the user has previously judged relevant. The result of this analysis identifies features that will help the tool recognize relevant results from new searches according to the user's personalized search history. These features are embodied in the form of LSI coefficients and are stored in the knowledge base of the system. A Radial Basis Function (RBF) based kernel is used in the construction of this SVM.

User feedback provides one of three outcomes for the obtained search result: (1) Relevant, (2) May be Relevant, (3) Not relevant. For "Relevant" results, the LSI coefficients corresponding to the result are increased so that the next similar search will regard the previously identified query category more favorably. Results marked as "May be Relevant" cause no changes to the LSI coefficients. "Not Relevant" feedback changes the LSI coefficients such that references related to the associated category are never obtained in a future search event.

In his pioneering work, Joachims has shown that SVM is an ideal parametric classifier for sparse matrix (e.g. a document matrix) classification [7]. The LSI coefficients in a vector space form a large sparse matrix. Hence, SVM is the ideal tool to generate the query category for a proper classification of search keywords. The most general form of Support Vector Classification is to separate two classes by a function, which is induced from available examples [5]. The main goal of this classifier is to find an optimal separating hyperplane that maximizes the separation margin or the distance between it and the nearest data point of each class. In the case where a linear boundary is inappropriate, the SVM can map the input vector *x* into a high dimensional feature space, *z* [5]. The idea behind kernel functions is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. With the kernel function, the inner product does not need to be evaluated in the feature space, which provides a way of addressing the curse of dimensionality. The theory of Reproducing Kernel Hilbert Space (RKHS) [6] claims that an inner product in feature space has an equivalent Kernel in input space,

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \quad \textbf{(1)}$$

provided that Mercer conditions hold. The feature vector $\phi(x)$ corresponds to the input vector *x*. The Gaussian Radial Basis Function GRBF) has received significant attention and its form is given by

$$K(x, x') = \exp(-\| x- x' \| / 2\sigma^2) \quad \textbf{(2)}$$

Classical techniques using RBFs employ some method of determining a subset of centers; typically a method of clustering is employed to select a subset of centers [5]. The most attractive feature of SVM is its implicit selection process, with each support vector contributing to local Gaussian functions, centered at the respective data point. By further consideration the global basis function width may be selected using the SRM principle. The RBF based kernel helps to construct complex nonlinear decision hyperplanes which help model complex classification tasks and thus contribute toward increased classification accuracy.

5 Empirical Testing

Early testing of the WHAT tool provided encouragement for the approach and suggested areas of further work. This experiment was conducted before the introduction of the SVMs. The results are based on the use of multiple search engines, and an initial ranking function that did not benefit from the experience of prior user search results. These results will form a base for comparison as the more sophisticated approaches are integrated into the tool and further testing is performed. In these initial tests, the performance of WHAT was compared against the performance of several standard internet search engines. Five search strings were devised to measure the relevance of results from Google, AlltheWeb, and Lycos against the results generated by WHAT and a variation of WHAT with a vocabulary supplement tool (WHATWORDNET). The queries tested were:

1. Baking Bread
2. Hiking Stoves
3. Java I/O introduction
4. Jaguars
5. Improving backstroke

Once the results were received, three student researchers took subsets of the data from each search engine ranging from 1200 to 1700 references. These results were combined into a single set by taking the union of each search engine subset and removing duplicates. Analyzing this combined data, the researchers categorized individual results as “relevant” or “not relevant.” Once the students agreed upon a common set of relevant results, they sent each query once again to every search engine. From these results, references were evaluated to measure the metrics *recall* and *precision*. Recall is the percentage of results from the query that also exist in the set of known relevant references; precision is the ratio of decidedly relevant results contained within the top twenty results. These measurements were then averaged among researchers.

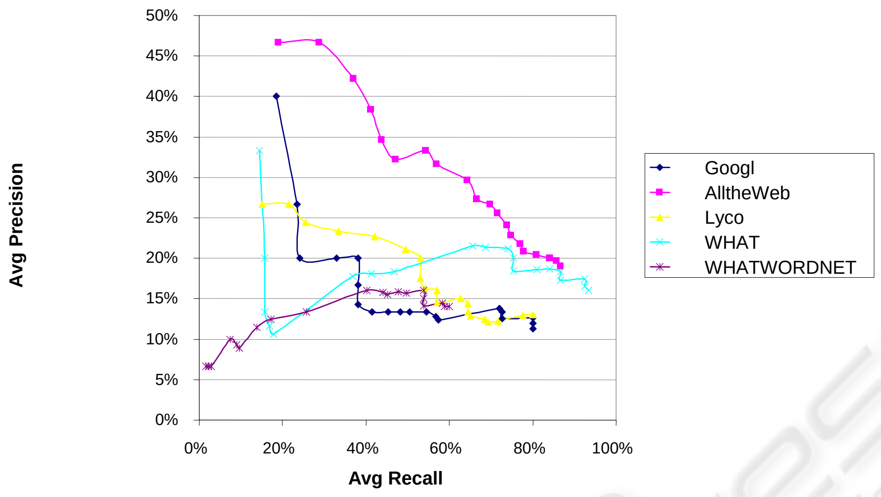

Fig. 2. Average precision is plotted against average recall for each search engine. Averages are taken over all search queries

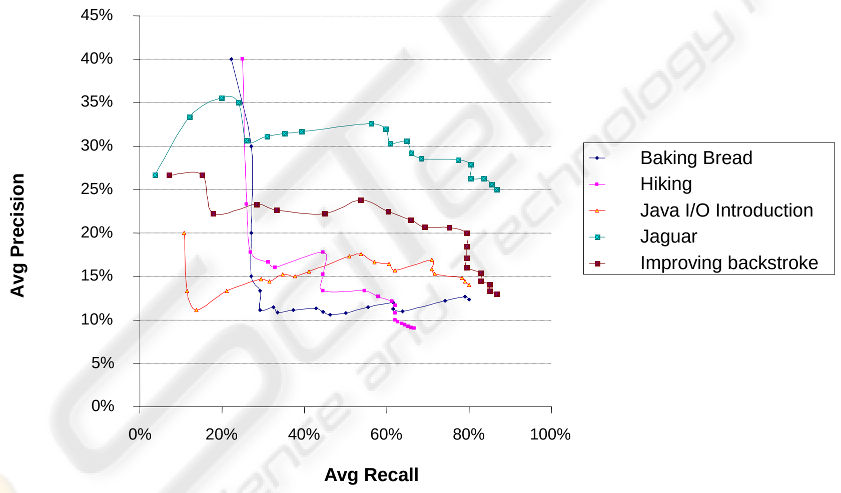

Fig. 3. Average precision is plotted against average recall for each search query. Here, the percentages are averaged over all search engines

In general, the percentages shown in the graphs indicate that on average a high percentage of irrelevant results are returned. Compared with WHAT, the standard search engines generally have a higher precision rating, implying that a higher percentage of relevant results are returned near the top of the result list. However, the WHAT engine retrieves a more consistent set of relevant references. Since recall for WHAT is higher for result sets with higher precision, the tool succeeds in delivering results that are better suited to the context of the intended search query.

Summary

The WHAT project is work in progress. The goal is highly individualized search results resulting from defined search contexts and learning derived from experience with user satisfaction with results over time. The project runs on the user's own computer, therefore eliminating any issue with regard to privacy and also allowing specific attention to the individual user's preferences. Early results show promise of an improved search tool. Current work is enhancing the intelligence of the system in learning user needs.

References

1. Foltz, P. W., Kintsch, W., and Landauer, T. K., (1998) "The Measurement of Textual Coherence with Latent Semantic Analysis", Discourse Processes, 25, pp. 285-307.
2. Foltz, P. W., (1990) "Using Latent Semantic Indexing for Information Filtering". In R. B. Allen (Ed.) Proc. of the Conference on Office Information Systems, Cambridge, MA, pp. 40-47.
3. Landauer, T. K., Foltz, P. W., and Laham, D. (1998) "Introduction to Latent Semantic Analysis", Discourse Processes, 25, pp. 259-284.
4. Landauer, T. K. and Dumais, S. T. (1997) "Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction ", Psychological Review, 104 (2), pp.211-240.
5. Gunn S. R. (1998) "Support Vector Machines for Classification and Regression", Technical Report, Department of Electronics and Computer Science, University of Southampton.
6. Vapnik V., "Statistical Learning Theory", Wiley & Sons, Chichester, GB, 1998.
7. Joachims T., "Text Categorization with Support Vector Machines", in the proceedings of European Conference on Machine Learning ECML-98.
8. Skonnard, A.: Understanding WSDL. In: MSDN Web Services Development Center (2003). Available: http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/understandwsdl.asp
9. Skonnard, A.: Understanding SOAP. In: MSDN Web Services Development Center (2003). Available: http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us//dnsoap/html/understandsoap.asp
10. Skonnard, A.: The XML Files: The Birth of Web Services. In: MSDN Magazine (2002). Available: http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/msdnmag/issues/02/10/xmlfiles/default.aspx
11. Box, D., Cabrera, L. F., Kurt, C.: An Introduction to the Web Services Architecture and Its Specifications. In: MSDN Web Services Development Center (2004). Available: http://msdn.microsoft.com/webservices/understanding/advancedwebservices/default.aspx?pull=/library/en-us/dnwebsrv/html/introwsa.asp