

# Intelligent support for information retrieval in the WWW environment

Robert Koval' and Pavol Návrat

Slovak University of Technology in Bratislava

Department of Computer Science and Engineering

## Abstract

The main goal of this research was to investigate means of intelligent support for retrieval of web documents. We have proposed the architecture of the web tool system - *Trillian*, which discovers the interests of users without their interaction and uses them for autonomous searching of related web content. Discovered pages are suggested to the user. The discovery of user interests is based on analysis of documents that users had visited in the past. We have created a module for completely transparent tracking of the user's movement on the web, which logs both visited URLs and contents of web pages. The post analysis step is based on a variant of the suffix tree clustering algorithm. We primarily focus on overall Trillian architecture design and the process of discovering topics of interests. We have implemented an experimental version of Trillian and evaluated the quality, speed and usefulness of the proposed system. We have shown that clustering is a feasible technique for extraction of interests from web documents. We consider the proposed architecture to be quite promising and suitable for future extensions.

## Introduction

The World Wide Web (web) has become the biggest source of information for many people. However, many surveys among web users show that one of the biggest problems for them is to find the information they are looking for [Kehoe and Pitkow, 99]. There are many reasons, why searching for relevant information on the Internet is so inefficient. First of all; the web is a vast, distributed, publicly available information space, which contains mostly heterogeneous and unstructured data. The heterogeneous nature of information sources significantly complicates the process called *information retrieval*. Secondly, the amount of data available on the web space grows at remarkable speed. Therefore, there is a need for a tool, which would assist the users with their browsing and make their on-line experience more comfortable, while searching for the topics of their interests. Today's research in this field concentrates on software entities (also called agents) that would help users filter vast amounts of data available on-line and adjust themselves to the particular needs of every user. In our work, we tried to design such a tool, to propose its architecture and to evaluate its quality and usefulness.

The goal of our work is to design a system capable of discovering user's topics of interest, his or her (we shall use the masculine form for short in the rest of the

paper) on-line behaviour, and his browsing patterns, and to use this information to assist him while he is searching for information on the web.

In recent years new systems have been proposed to overcome problems related to information retrieval and to accommodate the web for an average unskilled user. The main idea of these systems is to discover the user's browsing behaviour and his topics of interest. By possessing this information, the system can help the user formulate his search queries, assist him during web browsing and bring more advantages. In our work we intend to devise such a system and evaluate its usefulness in a real world environment.

Such a personalised system has many advantages over common search engines. First of all, the system has closer knowledge about the user and is able to discover his potential information needs with higher probability than a search engine. The system can operate in a multi-user environment and become a basis for collaborative filtering or advanced caching.

In general we can call any software system, which helps the user retrieve, locate and manage web documents, a web tool. Web tools can be classified in many ways. Cheung, Kao and Lee have proposed one such classification in [Cheung, Kao, Lee, 98]. They classify web tools in 5 levels (0-4), from a regular browser to an intelligent web tool (assistant). A level 4 web tool is expected to be capable of learning the behaviour of users and information sources. In our work, we try to be consistent with this classification and we focus our effort on the most advanced kind (level 4) web tool system.

Our vision of what our intelligent web tool will be able to do can be described by following scenario:

The intelligent web tool observes the user's on-line behaviour, his browsing patterns and favourite places. The web tool works as a personal agent and gathers the data needed to identify the user's interests. By analysing the visited documents, their order, structure and any other attributes, it discovers the user's domains of concern. Web tool can locally store documents visited by the user. This yields some useful advantages such as a full text search over visited content, easier information sources behaviour analysis and in a multi-user environment even a basis for advanced custom caching functionality [Haobo and Breslau, 99] or collaborative browsing [Ungar and Foster, 98; Lashkari 95]. When the user's topics of interest are acquired, the tool starts looking for relevant information on web autonomously. When it finds relevant documents, they can be presented to the user in the form of suggestion and he is able to evaluate the quality of them either by implicit or explicit relevance feedback. This feedback can be thought of as a contribution to the tool's total knowledge of the user's profile. Moreover, the tool has to be able to identify the information sources behaviour. As an example, we can notice web sites devoted to news services, which change their content daily or even on a more frequent basis. The system has to discover such regular changes of an information source so that it could inform the user more precisely about content change or even download whole content in advance.

To discover the user's interests on the web, his movements and behaviour have to be monitored so that the knowledge about his profile can be acquired. Software entities called agents can be used to achieve such a functionality.

Software agents can be included in web tool architecture in two main areas. The first important use of agents lies in the field of user's behaviour tracking and monitoring. Agents can also be used for autonomous searching for relevant content on web (search agents).

Low precision of search engines searches disables convenient information retrieval process. As the searches are quite imprecise and are of a varying quality, there is a need for a tool (system) capable to overcome the mentioned problems and improve the overall search engine's performance and the quality. We need improved automatic methods for searching and organising text documents so that information of interest can be accessed fast and accurately.

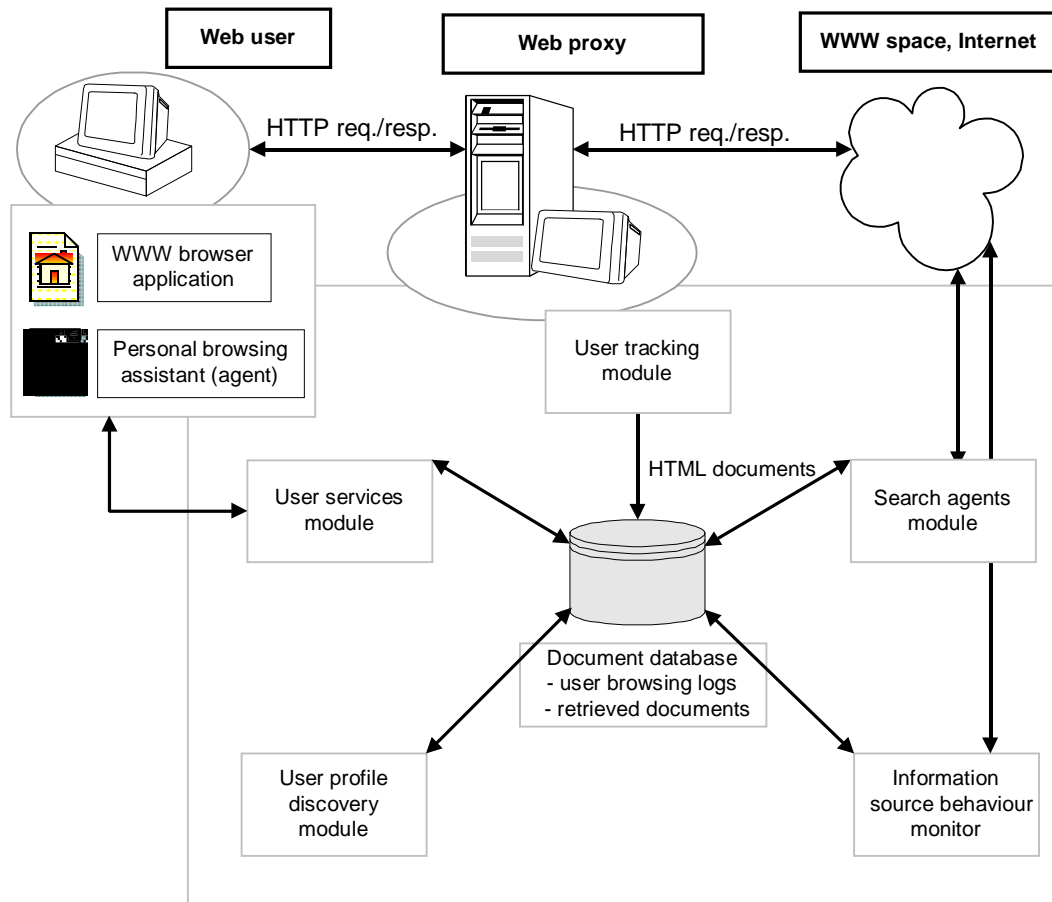
Motivation of this research is to design and evaluate system or architecture able to assist the users in information retrieval and make web experience convenient for them. The basic question asked in this project is: "What is a feasible way of helping users to find documents placed in the web space that match their interests?" Web grows extensively and the users find it sometimes very difficult to locate information they are looking for.

We want to design a complex web tool, its architecture and implement some of the designed modules as prototypes to evaluate usefulness of system and discover related tasks and problems.

## Design

In this part we want to focus on the design of the web tool. We shall introduce the architecture of our web tool system, which we shall call by the code-name "*Trillian*".

The proposed multi user architecture of the Trillian system is shown in Figure 1. It consists of six main modules. The user connects to web pages using his standard web browser. *The user tracking module* records his movement and content of the web pages he visits. *User profile discovery module* is responsible for profile analysis and identification of user's information needs. *Search agents* search for data relevant to the user's preferences and update *the document database*. *Information sources behaviour monitor's* main goal is to identify how the most popular web pages change over time, which is very important for the pre-caching mechanism. *The user services module* is an interface between the user and the system.



**Figure 1. Trillian Architecture**

Figure 1 illustrates the top-level decomposition of the system. In the sequel, we describe the core features and functionality of each individual module together with a description of its sub-modules.

### ***The web browser***

The web browser (e.g. Internet Explorer, Netscape, Mosaic) is a standard software tool for accessing web pages on the internet. The web browser accesses the Internet via a proxy server. The architecture is independent from the version or type of the browser. It is responsible for

- Navigation through web pages
- Graphical user interface to Trillian user
- Web browser extension (optional)
- Client events.

### ***Personal browsing assistant***

This module is the only part of a web tool system, with which the user cooperates. When new pages relevant to the user's interests are discovered, the browsing assistant displays them to the user. It is responsible for

- A user interface for full text search over visited documents.
- Relevance feedback.
- Personal caching control
- Monitoring control.

### ***Web proxy***

In the proposed architecture, proxy is a common connection point where core system components and functions reside. All HTTP traffic passes through the web proxy and this enables access to the content of the web pages that the user visits.

### ***User Tracking Module***

The purpose of this module is to track the user's browsing and behaviour in web space. It can reside anywhere along the HTTP stream. In our architecture the module is placed on the proxy server and therefore can access the user's HTTP responses to record the pages he visited and their content. The module records all information necessary to gain important knowledge about the user's on-line profile. can be used in conditions where the usage of the proxy server is not possible. It is responsible for

- Recording of visited pages and storing them locally in the Trillian document database.
- Recording clickstream data.

### ***User profile discovery***

This module represents probably the most important and difficult part of the whole Trillian architecture. Its main objective is to discover the user's topics of interest. This is done by the analysis of web access logs (clickstream data) and the content of visited web pages. The analysis has to be exhaustive to achieve reasonable results and therefore it has to be executed only in the post processing phase. In our work we have tried to perform some analysis "on the fly", but speed performance was reduced significantly.

The profile discovery process employs several miscellaneous algorithms to reach the desired goal. Methods such as cluster analysis, document cleanup, HTML analysis, browse path construction and others are used.

The module is responsible for:

- Clickstream analysis- the module has to analyse clickstream data to identify information-rich documents among all received pages.

- Discover clusters of user's interests – This process is core part of whole Trillian system. It is very important to produce meaningful and correct clusters, which best describe user's profile and interests. Module should analyse visited web pages and identify clusters of similar documents, words or phrases within them. To achieve this goal several clustering algorithms can be used. We have chosen a variant of the suffix tree clustering for this purpose.
- Use relevance feedback from users – During the analysis of visited documents, we can use relevance feedback provided by users from previous analysis results. Overall knowledge of the user's interests can be improved in this way. E.g. we can consider clusters, words, phrases or even whole documents as negative examples, which means, during the subsequent analysis we will not identify similar content as important for the user. The system can learn and better understand the user's needs this way.

### ***Search agents module***

This part of the system performs an autonomous pre-fetching of documents from the web space or web exploration. The module uses a softbot-based mechanism to retrieve documents from the web. It uses information obtained using profile discovery to search for pages relevant to the user's needs.

### ***Information sources behaviour monitor***

The web space and its dynamic behaviour cause frequent changes of many documents. The frequency of changes in web content is variable and depends on the particular web site. It varies from several years to several seconds (e.g. stock news). It is responsible for

- Discovery of page update patterns.
- Pre-fetch pages in advance.

### ***User services module***

The user services module transforms manipulated and analysed data into a form suitable for the user. It allows the user to provide content relevance feedback and feedback for monitoring and search agents. It is the interface to the core system parts through which the user gets data or controls processes, sets the parameters or explicitly describes his preferences. It is responsible for

- Full text search.
- Attribute search.
- Feedback mechanism.

### ***Central Database***

The central database stores all the data required for successful analysis and monitoring of the user and web pages. The system uses a repository architecture

template. All of the core modules work with the data from the central database and update its contents.

## **User tracking**

A goal of the user tracking module is to record user's movement and contents of the documents the user visits. As our empirical experiments shows, the main requirements for this module are speed and robustness.

## **Suffix tree clustering (STC)**

The discovery of the user's profile is probably the most important part of the Trillian's architecture. Its goal is to discover main document clusters using the analysis of visited documents (their contents) and the analysis of clickstream data. To discover the interests of the user we need to perform a complicated analysis composed of several steps. The main step in the analysis process is called *clustering*.

The clustering is used to extract groups of words (terms) or phrases, which tend to be of similar meaning. These groups – *clusters* are the final outcome of the clustering process. We want to accomplish the following: to extract all textual information from the documents and - analysing their contents - to form groups of similar documents or topics. Similar documents are those, which have something in common (e.g. share a common phrase or topic). This is based on the clustering hypothesis, which states that the documents having a similar content are also relevant to the same query [van Rijsbergen, 79]. The query in our case has the meaning of an information need of the user.

In the Trillian system, we have chosen to use a variant of a clustering method first introduced by [Zamir and Etzioni, 98] called the suffix tree clustering. The STC is used in the post-retrieval step of the information retrieval.

The suffix tree clustering algorithm relies on a data structure called a suffix tree. The suffix tree of strings is composed of suffixes of those strings. This formulation assumes only existence of one input sequence of strings. In our case we want to distinguish among string sequences from different documents. Therefore for this purpose there is a slightly modified structure called a generalised suffix tree, which is built as a compact trie of all the words (strings) in a document set. Leaf nodes in this situation are marked not only with a identifier of sequence but also carry information about the document from where the sequence originates.

There are several possible ways of building a suffix tree. We employ a version of Ukkonen's algorithm because it uses suffix links for fast traversal of the tree [Ukkonen, 95].

## **STC Process**

Suffix tree clustering uses suffix tree data structure. Building the suffix tree is only one step in whole process of document clustering. The STC is composed of 3 main steps: document cleanup, maximal phrase clusters identification and cluster merging.

*Document cleanup*- Documents have to be preprocessed first before their contents are inserted into the suffix tree. The HTML documents contain a lot of irrelevant tagging information, which has to be removed first.

*Maximal phrase clusters identification* – From efficiency of using the suffix tree structure, we can identify maximal phrase clusters. By maximal phrase cluster we mean such a phrase, which is shared by at least two documents. These maximal phrase clusters are represented in the suffix tree by those internal nodes, leaves of which originate from at least two documents (The phrase is shared by these documents). Afterwards, a score is calculated for each maximal phrase cluster (MPC). The score of the MPC is calculated using following formula:

$$s(m) = |m| \cdot f(|m_p|) \cdot \sum tfidf(w_i)$$

where  $|m|$  is the number of documents in a phrase cluster  $m$ ,  $w_i$  are the words in a maximal phrase cluster and  $tfidf(w_i)$  is a score calculated for each word in the MPC.  $|m_p|$  is the number of non-stop words within the cluster. Function  $f$  penalises short word phrases; it is linear for phrases up to 6 words long and is constant for longer phrases. In this stage we can also apply scores of each word calculated by their position in the HTML document (e.g. in  $\langle H1 \rangle$  tag or  $\langle B \rangle$ ). A final score of each term can be obtained by multiplying its  $tfidf$  score and its HTML position score.

TFIDF – Term frequency inverse document frequency is a calculation, which evaluates the importance of a single word using an assumption that very frequent word in the whole document collection has a lower importance than the one appearing less frequently among the documents.

After the weighting of all maximal phrase clusters we select only the top  $X$  scoring ones and consider them for the following step. This selection prevents the next step from being influenced by a low scoring, and thus presumably less informative phrase clusters [Zamir, 99].

*Cluster merging* – After we have selected maximal phrase clusters, we need to identify those groups, which share the same phrase. By calculating binary similarity measure among each pair of maximal phrase clusters, we can create a graph where similar MPC's are connected by edges. Similarity among clusters is calculated using assumption that if phrase clusters share significant number of documents they tend to be similar.

Each cluster can now be seen as one node in a *cluster merge graph*. When two clusters in this graph are similar they are connected by an edge. The final stage of the cluster merging phase is the selection of groups of connected components in a cluster merging graph. The connected components in this graph now represent the final output of the STC algorithm – the *merged clusters*. Afterwards, the merged clusters are sorted by score, which is calculated as the sum of all scores of the maximal phrase clusters inside the merged cluster. A *connected component* of an undirected graph is a set of nodes such that there is a path between each pair of nodes in the set.

Finally, we report only top 10 merged clusters with the highest score. After all 3 steps of the STC algorithm each merged cluster can contain phrases, which are still

too long. In this case, we have to proceed with the next step, which is the selection of cluster representatives.

Cluster representatives can be selected using two techniques:

- *Term TFIDF score*- TFIDF score is calculated for every word inside a merged cluster. Words appearing with low frequency among maximal phrase clusters, but with high frequency inside maximal phrase cluster are considered as best representatives of a merged cluster.
- *Merged cluster clustering* – We can apply the same clustering mechanism for maximal phrase clusters as we used for the clustering of documents. This technique can identify the maximal phrase clusters within a merged cluster. This allows us to select common phrases inside a merged cluster. The result of this technique will have a higher quality than the previous technique.
- *Combination* – Identification of common phrases within a merged cluster can yield only a small number of phrases and therefore we cannot use them alone as cluster representative. Thus we can use a combination of the two previous techniques to achieve the desired result.

### ***STC Complexity and Requirements***

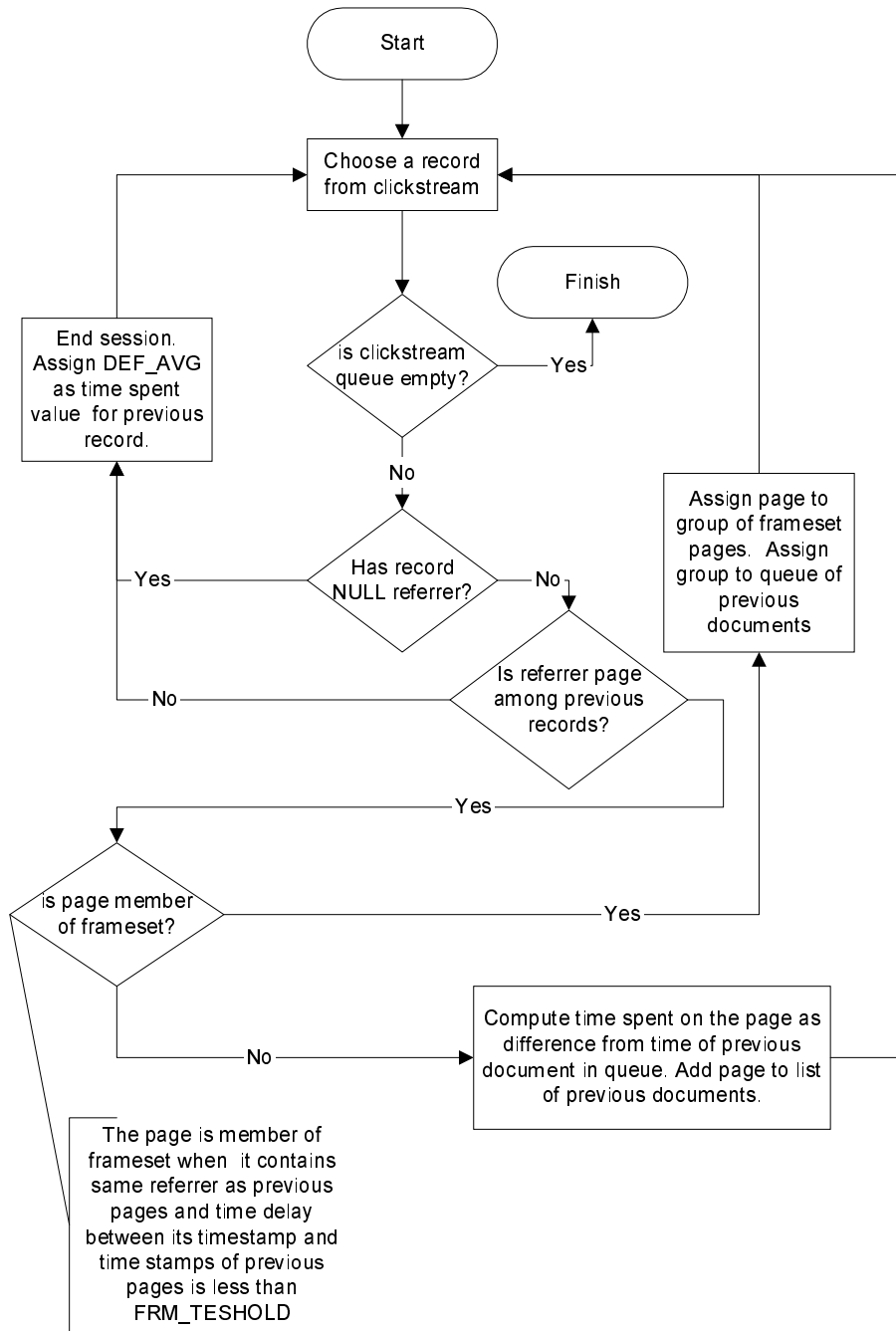
Suffix tree clustering algorithm has a linear time complexity depending on the size of the document set, which has be clustered.

STC can be built incrementally, which means when new documents are added to the document set, they can be directly inserted into the suffix tree without the need to rebuild the tree again.

### ***Clickstream Analysis***

Clickstream is a sequence of log records of user responses or requests, which model his movements and activities on the web. The clickstream data is basically collected on two points of the web communication. Web sites them selves maintain logs of the user's activities. The second point is located on the entry point of the user's connection to the internet, which is in most cases the proxy server. The data collected at the proxy server has a higher meaning for the analysis of the user's behaviour because they track the user's activities among all web sites.

Data mining techniques can also be applied for clickstream analysis to achieve higher quality. Chen, Park and Yu have proposed some promising algorithms for mining for interesting patterns in clickstream data [Chen et al, 96]. In our work we have applied our own simplified version of the clickstream analysis algorithm for determining the important documents within a sequence. A top level description of the algorithm is show on Figure 2.



**Figure 2. Top level description of simplified clickstream analysis algorithm.**

The main goal of this algorithm is to determine the approximate time spent on each page. The algorithm tries to identify groups that belong to a common frameset. This is done by a simple principle: When a page has the same referrer as the previous page in the clickstream and the time difference is less than FRM\_TRESHOLD (in our algorithm 5 seconds) we consider the page to be a member of a frameset. Another important issue is the session. By session we mean a continuous process of searching for information. When two pages in a sequence have a time difference higher than 30 minutes we assume the end of the session. We cannot exactly say how much time the user has spent on the last page in the session because the following record belongs to a different session. In this case we assign such a page a default value (5 minutes).

We have also created a technique to avoid duplicated documents in the analysis document set. For each web page we generate its page digest, which is a short byte sequence that represents the document. If two documents are exactly the same, their page digests will match. Occurrences of duplicates among gathered data will be often, because during web sessions users often return to the previous page. If duplicates would be removed from the analysis document set, our algorithm would identify the topic represented in the duplicated document as more important although it shouldn't. We used MD5 algorithm to generate the page digests and we save them into the document database. Thus, identification of the duplicates gets easier.

### ***Caching Strategies and Collaboration***

Several caching strategies exist and their main goal is to attempt to achieve the highest possible degree of consistency of the web cache. The summary of these strategies is best described in the paper [Haobo and Breslau, 99].

For the Trillian system, we propose a slightly modified idea of the web cache. Traditional web cache systems maintain only the documents that have already been visited by users. Our approach is to employ an information source behaviour monitor, which discovers update patterns of the web sites. With this knowledge, we can send search agents to pre-fetch pages to the local cache and store them locally even though the users haven't seen them yet. This pre-fetch mechanism will be only applied to those pages that are popular among users or pages explicitly requested by user.

Collaborative filtering [Polčicová, 99; Polčicová, Návrat, 00] is a technique for locating similarity of interests among many users. Let's say the interests of users A and B strongly correlate. Afterwards when a newly discovered page is interesting to user A there is a high probability that user B will also be interested. The page is then recommended to user B as well. For filtering purposes we could use two methods for sharing the information and knowledge among users:

- *Automated filtering* – The ratings of the documents and URLs are discovered by the system automatically.
- *Manual documents rating* – Users can manually present their opinions on visited documents. The system recommends the documents based on these manual ratings.

Users can exchange knowledge and pointers to interesting resources on the web, share knowledge or otherwise collaborate. If a collaborative filtering system is used in a community where information requirements of the users are similar (e.g. employees of an IT company) there is a high probability that the collaboration will be very successful and the recommendations will be interesting to many users.

There are many methods for building a collaborative framework, e.g. [Ungar and Foster, 98] [Polčicová, Slovák, Návrat, 00] and [Lashkari 95]. We can say that Trillian's architecture is ready to be extended for collaboration among users.

## **Evaluation and results**

In this part we focus on an evaluation of the implemented modules. We tested every module of the intelligent web tool architecture. Our primary interest was whether a specific module meets its requirements and if its performance and results are satisfactory.

Effectiveness of the clustering depends on relevance of the discovered clusters. Quality of clusters (ability to discover distinct topics of interests or group of the documents that correlate) depends on the user's opinion. This is a common problem in many IR tasks. Therefore we need to use a collection of the test data, which has already been evaluated by its authors. A test collection for information retrieval requires three components: 1) a set of documents, 2) a set of queries, and 3) a set of relevance judgments [Gordon et. al., 98]. We can then evaluate our clustering algorithm comparing its results to human categorization of documents in collection.

Today, many collections are available for academic purposes on the web. They differ in many aspects, but primarily in size, structure and type of data. We were interested in results of the web documents clustering and therefore we required such a collection of documents. Such collections exist, but they are not always publicly available. We had no choice but to use existing collections from other domains and adapt them for our purposes.

In our experiments we used three testing collections: Syskill and Webert web page ratings, LISA collection, Reuters-21578 collection. The testing collections, especially Reuters-21578 were too large for our purposes, thus we selected only subsets from them. Our experiments were based on these subsets.

### ***Evaluation methodology***

For evaluation of clustering quality we used a common IR technique sometimes called "merge then cluster". The common approach is to generate a synthetic data set where the "true" clusters are known. This can be done by generating it based on a given cluster model, or, and more suitable for document clustering, by merging distinct sets of documents into one. The resulting document set is then clustered using different clustering algorithms, and the results are then evaluated given how closely they correspond to the original partition. [Zamir, 99]

For numerical evaluation, we used two basic metrics: The precision factor and pair-wise accuracy. We borrow these metrics and methodology from [Zamir, 99], because we want to compare our results with his work.

**Precision Factor:** For each identified cluster we find the most frequent topic and consider it as "true" cluster topic. The Precision is afterwards calculated as the number of documents that belong to the "true" cluster divided by the total number of the documents inside the cluster. Because not all documents are clustered we also use normalised precision factor, which is a precision factor multiplied by the fraction of the documents that were clustered.

**Pair-Wise Accuracy:** Within each cluster we compare pairs of documents inside. We count true positive pairs (documents originally belong to the same group) and false positive pairs (documents were not originally in the same group).

The pair wise accuracy is calculated as follows, cf. also [Zamir, 99]:

Let  $C$  be a set of clusters,  $tp(c)$  and  $fp(c)$  be the number of true-positive pairs of documents and the number of false-positive pairs in cluster  $c$  of  $C$ . Let  $uncl(C)$  be the number of unclustered documents in  $C$ . We define the pair-wise score of  $C$ ,  $PS(C)$ , as:

$$PS(C) = \sum \text{sqrt}(tp(c)) - \sum \text{sqrt}(fp(c)) - uncl(C)$$

where the summations are over all clusters  $c$  in  $C$ . We use the square roots of  $tp(c)$  and  $fp(c)$  to avoid over-emphasizing larger clusters, as the number of document pairs in a cluster  $c$  is  $|c| \cdot (|c|-1) / 2$ . We subtract the number of unclustered documents, as these documents are misplaced. The maximum pair-wise score of  $C$ ,  $\text{maxPS}(C)$ , is:

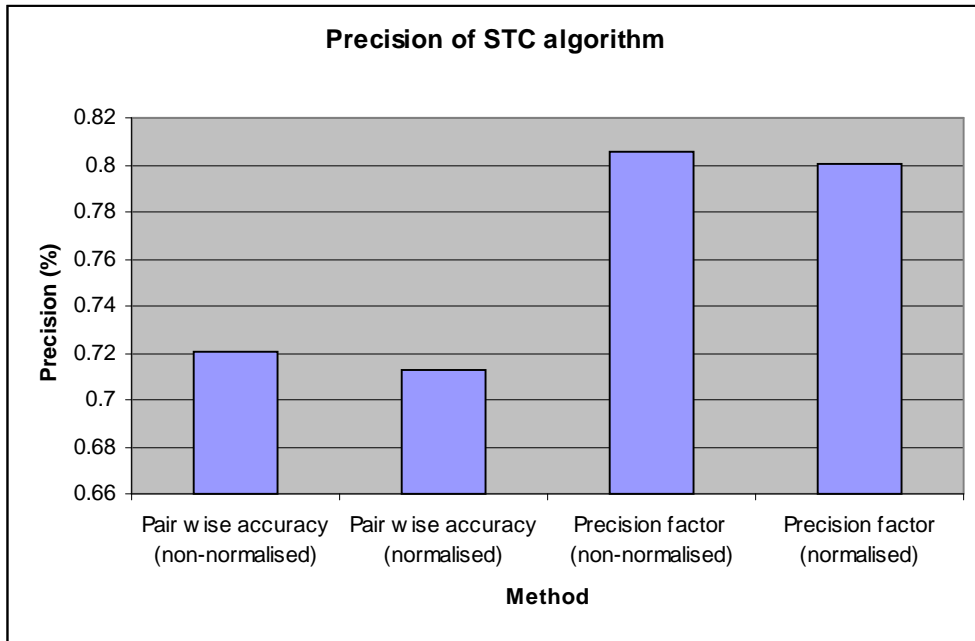
$$\text{maxPS}(C) = \sum \text{sqrt}(|c| \cdot (|c|-1) / 2)$$

where  $|c|$  is the number of documents in cluster  $c$ . This is simply the sum of the square roots of the number of document pairs in each cluster. Finally, we define the pair-wise accuracy quality measure of  $C$ ,  $PA(C)$  as:

$$PA(C) = (PS(C) / \text{maxPS}(C) + 1) / 2$$

### STC Quality

Figure 3 presents results of our variant of the STC using the three mentioned collections. Our results are similar to those reported by Zamir [1999] who compared six clustering algorithms including his version of the STC. They independently endorse the superiority of the STC which,



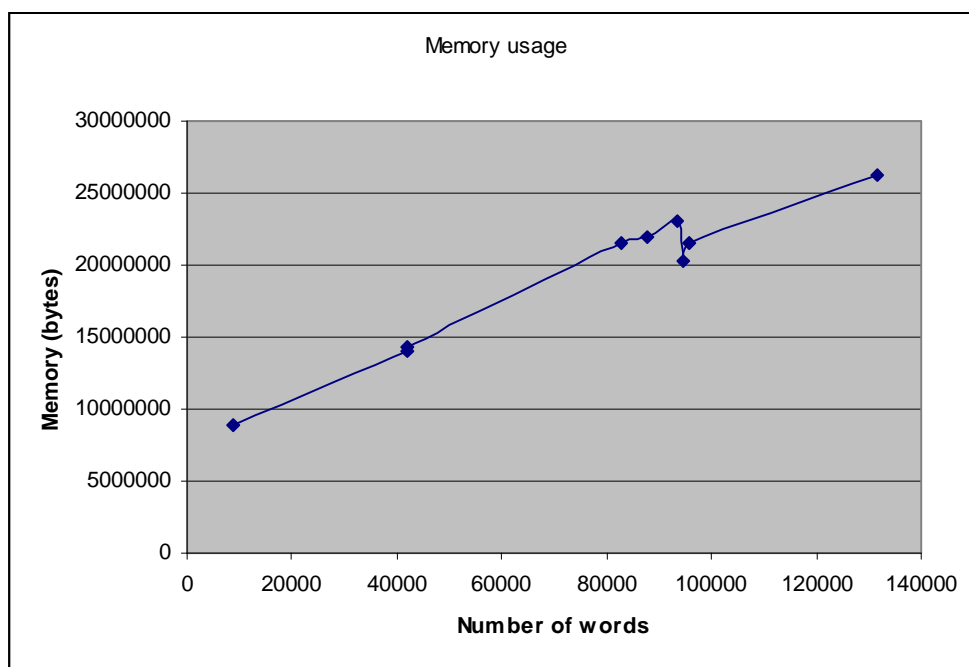
**Figure 3. Precision of Trillian STC algorithm.**

together with the k-means algorithm, outperform the other algorithms.

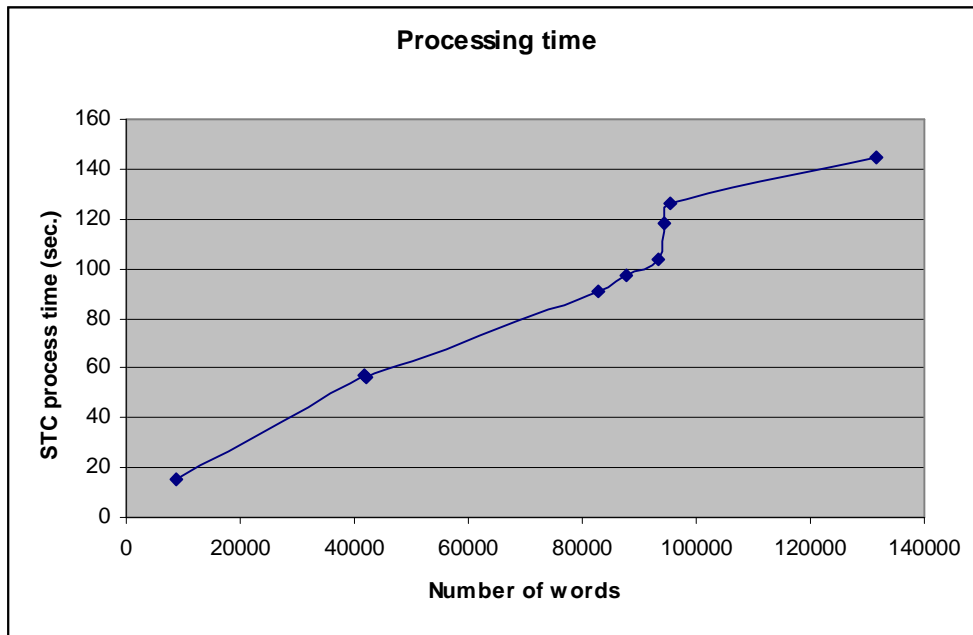
### Speed and Space Requirements

Theoretically, STC time and space complexity should be linear. In our experiments we also measured time and memory used during each test. We were interested whether the theoretical assumption will be confirmed. As Figure 4 and Figure 5 show, the STC meets its theoretical assumptions and is linear depending on size of the document set. Following pictures show how number of words in clustered documents relates to the STC processing time and memory requirements. However, in our case the STC has to cluster huge amount of documents. During our experiments we empirically evaluated reasonable number of documents, which our implementation of STC can handle without any memory problems. We have found that in general our STC implementation can handle up to 800 documents (we used 128 Mb RAM computer) without bigger problems. This number is not very high, when we imagine number of document after 1 week of browsing. It depends on behaviour of user, but in some cases it can much higher than mentioned 800.

Our implementation of the STC holds the whole suffix tree structure in the memory. Thus in big document set memory gets full and the algorithm suffers, because the processor deals mostly with memory management. Therefore we suggest that STC implementation must create a persistence mechanism for storing parts of suffix tree on disk, when they are not needed. This might allow STC to be used on much bigger document collections.



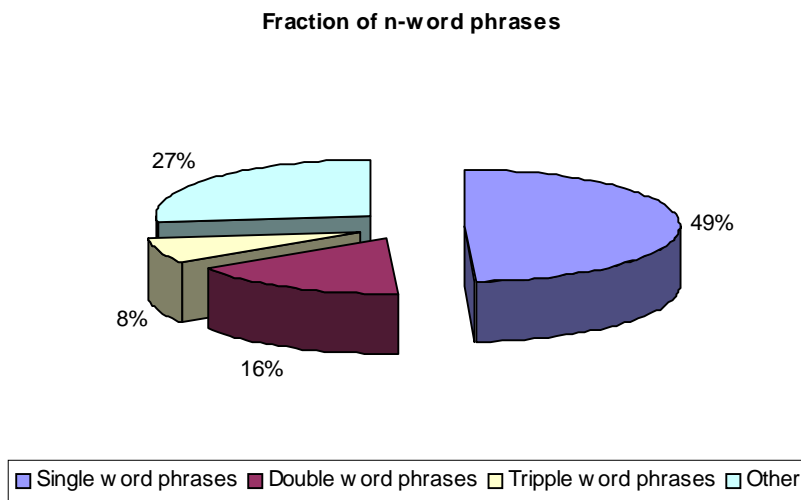
**Figure 4. Relation between number of words in document set and memory requirements.**



**Figure 5. Relation between number of words in document set and stc process time**

### Phrases

One of the advantages of the suffix tree clustering is the usage of phrases. We were therefore interested how much do phrases contribute to the overall quality of the algorithm. In the experiment, we measured how many of the phrases contain more than one word. Figure 6 shows percentage of single, double, triple and longer phrases in clusters on average.



**Figure 6. Average fraction of the n-word phrases in clusters .**

As the results show, more than half of the phrases in MPC's were not single worded. This is a very promising fact, because the phrases carry higher information value in IR systems and tell us more about the context of the discovered topic.

## Conclusions

The result of this project is the designed architecture of an intelligent web tool system, which is able to discover the user's information needs (topics of interest) and help him locate documents from the web potentially relevant to his interests.

The main question of this research was: "What is a feasible way of helping users to find documents located in the web space that match their interests?" Our answer to this question is the design of the Trillian architecture. We believe that this proposed architecture is good for achieving our main goal: intelligent information retrieval. It is not only designed to analyse the user's interests and help him to locate relevant web pages afterwards, but also allows for future enhancements in the form of collaborative filtering or custom caching and the pre-fetching mechanism.

We have identified clustering of web documents as the best way of analysing the user's information needs (profile discovery). We have evaluated the feasibility of the suffix tree clustering algorithm for web tool purposes and we consider it very useful for this purpose. However, some future enhancements have to be done for the improvement of space requirements and speed. We have also identified the clickstream analysis as an important part of whole the analysis process. Clickstream data can significantly improve the system's knowledge of the user's profile. We have identified clickstream analysis as not being an easy task and described the main problems related to this process. We have proposed a simplified version of the analysis algorithm, but we consider the usage of more advanced algorithms, such as data mining techniques or artificial neural networks, to be viable alternatives, too.

The browsing behaviour of users and the activity of search agents update the document database. Search agents are designed to discover the relevant web documents on the internet based on the discovered user profiles. We believe this is the proper method for helping the user to locate documents related to his interests.

## References

[Armstrong, 95] Armstrong et al., WebWatcher: A Learning Apprentice for the World Wide Web. In: Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous Environments, p 6 –12, Stanford University, 1995, AAAI Press.

[Bamshad et al., 97] Bamshad Mobasher, R. Cooley and J. Srivastava, Web Mining: Information and Pattern Discovery on the World Wide Web, Proceedings of the 9th IEEE International Conference on tools with Artificial Intelligence (ICTAI'97), November 1997.

[Catledge and Pitkow, 95] Catledge, D. and. Pitkow J. Characterizing Browsing Strategies in the World-Wide Web. In *The Third International World-Wide Web Conference: Technology, Tools and Applications. April 10-14, 1995. Darmstadt, Germany.* [Available online]: <http://www.igd.fhg.de/www/www95/proceedings/papers/80/userpatterns/UserPatterns.Paper4.formatted.html>. [Accessed: September 11 2000].

[Chen et al, 96] Chen, M. S., Park, J. S., and Yu, P. S., Data mining for path traversal patterns in a Web environment. *In: Proceedings of 16th International Conference on Distributed Computing Systems*, 1996.

[Cheung et al., 98] Cheung, D.W., Kao, B., Lee, J. Discovering user access patterns on the World Wide Web. *Knowledge Based Systems*, 10(1998), 463-470.

[Dorohocenu and Nevill-Manning, 2000] Bogdan Dorohocenu and Craig Nevill-Manning. A practical Suffix-tree implementation for string searches. *Dr. Dobbs's Journal*, pp. 133-140, #314, July 2000.

[Etzioni and Weld, 95] Etzioni, O., Weld, D.S. Intelligent Agents on the Internet: Facts, Fiction and Forecast. *IEEE Expert*, 10,4,1995, 44-49.

[Gordon et. Al, 98] Gordon V. Cormack, Christopher R. Palmer, and Charles L. A. Clarke. Efficient construction of large test collections. In *Proceedings of SIGIR 98, the ACM Annual Conference on Research and Development in Information Retrieval*, pages 282--289, Melbourne, Australia, August 1998.

[Haobo and Breslau, 99] Haobo Yu and Lee Breslau, A Scalable Web Cache Consistency Architecture, in *Proc. of ACM SIGCOMM'99*, Sept. 1999.

[Hearst, 98] Hearst, M. A. The use of categories and clusters in information access interfaces. In T. Strzalkowski (ed.), *Natural Language Information Retrieval*, Kluwer Academic Publishers, 1998.

[Kehoe and Pitkow, 99] Kehoe, C. and Pitkow, J. Tenth WWW Survey Report, The Graphics, Visualization & Usability Center, Georgia Institute of Technology, 1999. [Available online]: [http://www.gvu.gatech.edu/user\\_surveys/survey-1998-10/](http://www.gvu.gatech.edu/user_surveys/survey-1998-10/), 1999. [Accessed: November 24 2000]

[Koval, 99] Koval, R. Intelligent support for information retrieval in WWW environment, Slovak University of Technology, Master thesis project. 1999.

[Lashkari 95] Yezdi Lashkari Feature guided automated collaborative filtering. Master's thesis, MIT Department of Media Arts and Sciences, July 1995.

[Lawrence and Giles, 99] Lawrence, S. and Giles, L. Accessibility and Distribution of Information on the Web. *Nature*, 400, 107-109, July 1999.

[Lawrence and Giles, 98] Lawrence, S. and Giles, L. Inquirius, the NECI meta Search Engine, in *Seventh International World Wide Web Conference*, (Brisbane, Australia), 95-105, 1998.

[Lewis 91] Lewis, D. Learning in intelligent information retrieval. In *proceedings of Eighth International Workshop on Machine Learning*, p. 235 – 239, 1991.

[Mogul et al., 97] Mogul, J. Frystyk, H. Berners-Lee, T. Hypertext Transfer Protocol - HTTP/1.1(RFC 2068), World Wide Web Consortium, HTTP Working Group, January 1997

[Pei et al., 00] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, Mining Access Pattern efficiently from Web logs, Proc. 2000 Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'00), Kyoto, Japan, April 2000.

[Perkowitz and Etzioni, 97] M. Perkowitz and O. Etzioni. Adaptive Web Sites: an AI challenge. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1997.

[Polčicová, 99] Polčicová, G.: Recommending HTML-documents using Feature Guided Automated Collaborative Filtering. In: Eder, J., Rozman, I. and Welzer, T.: Advances in Databases and Information Systems, Proceedings of the 3rd East European [ADBIS'99](#) Conference, Short Papers, 86--91, 1999.

[Polčicová, Návrat, 00] Polčicová, G., and Návrat, P.: Recommending WWW information sources using Feature Guided Automated Collaborative Filtering, In: Shi, Z., Faltings, B. and Musen, M.: Proceedings of Conference on Intelligent Information Processing at 16th [World Computer Congress](#), 115--118, 2000.

[Polčicová, Slovák, Návrat, 00] Polčicová, G., Slovák, R., and Návrat, P.: Combining Content-based and Collaborative Filtering. In: Masunaga, Y., Pokorný, J., Štuller, J., and Thalheim, B.: Proceedings of Challenges, [2000 ADBIS-DASFAA](#) Symposium on Advances in Databases and Information Systems, 118--127, 2000.

[van Rijsbergen, 79] van Rijsbergen, C. J Information Retrieval, Butterworths, London, 1979.

[Salton and Buckley, 88] Salton, G., and Buckley, C. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513-523, 1988.

[Selberg and Etzioni, 97] E. Selberg, O. Etzioni. The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert*, (January–February):11–14, 1997.

[Thacker, 82] C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, "Alto: A Personal Computer," pp. 549-572 in *Computer Structures: Principles and Examples*, ed. D. P. Siewiorek, C. G. Bell and A. Newell, McGraw-Hill (1982).

[Ukkonen, 95] Ukkonen, E. On-line construction of suffix trees. *Algorithmica*, 14:249-260, 1995.

[Ungar and Foster, 98] L. Ungar and D. Foster. Clustering methods for collaborative filtering. *AAAI Workshop on Recommendation Systems*, 1998.s

[Zamir and Etzioni, 98] Zamir, O. and Etzioni, O. Web Document Clustering: A feasibility demonstration. In: *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, 46-54, 1998.

[Zamir et al., 97] Zamir, O., Etzioni, O., Madani O. and Karp, R. M. Fast and intuitive clustering of Web documents. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97)*, 287-290, 1997.

[Zamir, 99] Oren Eli Zamir, Clustering Web documents: A phrase based method for grouping Search Engine results, University of Washington, 1999