Computing and Informatics, Vol. 21, 2002, 509–528

INTELLIGENT SUPPORT FOR INFORMATION RETRIEVAL OF WEB DOCUMENTS

Robert Kovaľ, Pavol Návrat

Department of Computer Science and Engineering
Slovak University of Technology in Bratislava
Ilkovičova 3, 812 19 Bratislava, Slovakia
e-mail: `navrat@elf.stuba.sk`

Manuscript received 14 May 2002; revised 18 November 2002
Communicated by Jaroslav Pokorný

Abstract. The main goal of this research was to investigate the 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 visited by the users previously. 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 prototype 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.

Keywords: Intelligent information retrieval, suffix tree clustering algorithm, click-stream analysis, web tool, search agent

1 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 [10]. 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 a 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 [5]. 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. We focus our effort on the more advanced kinds of a 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 [33] or collaborative browsing [32, 13]. 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 a form of suggestion and he is able to evaluate their quality 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 more often. 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 rapidly 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 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.

The rest of the paper is structured as follows. In Section 2, we present our new approach, and, in particular, design of our new web tool and its architecture. In Section 3 suffix tree clustering process is described. Next, we deal with clickstream analysis. We also discuss caching strategies and collaboration as further techniques that can be incorporated in our design. Section 5 contains a detailed report on evaluation of our design. Evaluation methodology is introduced first. Then results of an empirical evaluation of speed and space requirements are presented. Section 6 concludes the paper, including some discussion on a possible future work.

2 DESIGN OF THE WEB TOOL ARCHITECTURE

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. *User tracking module* records his movement and content of the web pages visited. *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 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. It 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 co-operates. 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 and monitoring control.

Web proxy. In the proposed architecture, web 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.

While from a purely technical point of view, assuming that users access the internet via an internet service provider, the proxy can be located either at the ISP or on user's computer; the choice can make a lot of difference from legal point of view. In particular, there are many privacy issues that may need to be addressed.

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. It can be used in conditions where the usage of the proxy server is not possible. It is responsible for: recording of visited

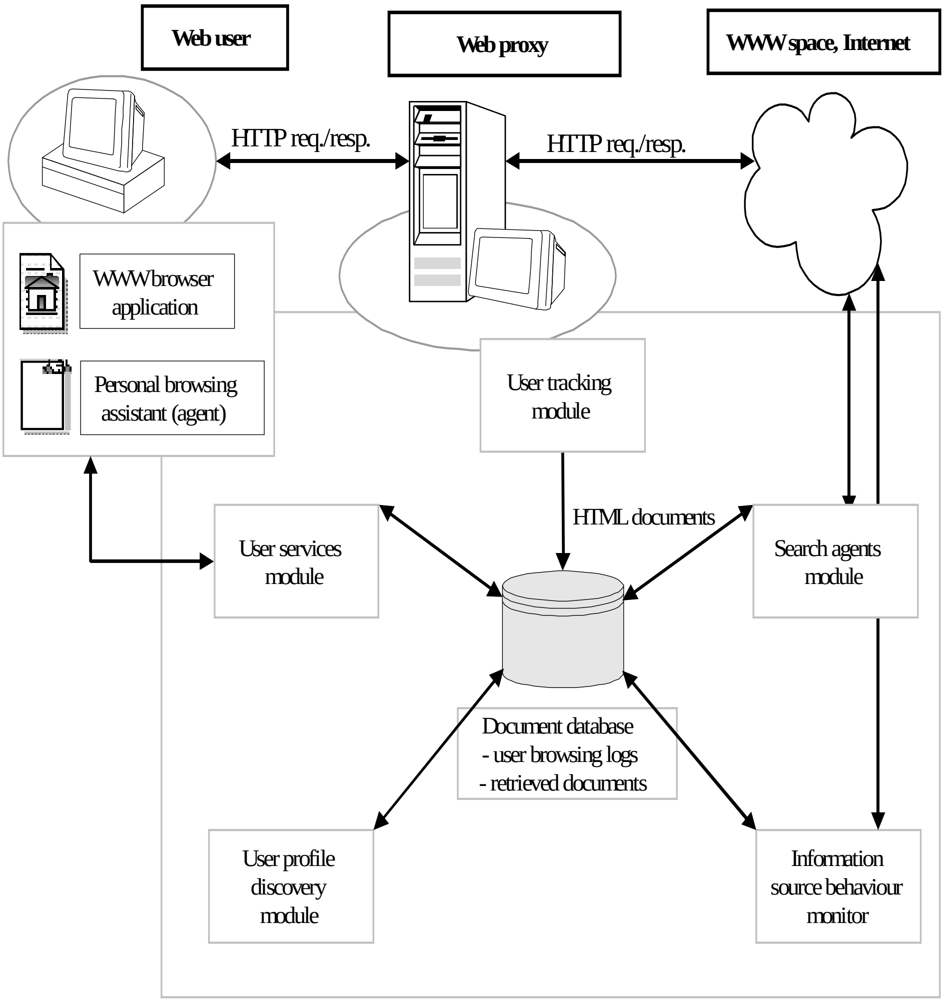

Fig. 1. Trillian architecture

pages and storing them locally in the Trillian document database, recording of 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 an 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 the received pages), *discovering clusters of user's interests* (this process is the core part of the whole Trillian system. It is very important to produce meaningful and correct clusters, which give the best description of user's profile and interests. The 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 results of the previous analysis. 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).

One important issue is a possibility that a user changes his interests. Indeed, most of research, including our one, does not sufficiently address this problem. It is clear that when user interests change, new clusters have to be formed and some existing ones may have to be deleted.

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.

The task of discovery of page update patterns is by far not trivial. Among the approaches that can be followed here, let us mention at least temporal web mining.

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

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

Here the underlying assumption is that it is quite easy to identify each distinct user from the log of the proxy server. It should be noted, however, that the IP addresses in the log may not be entirely adequate to distinguish individual users [17].

3 SUFFIX TREE CLUSTERING

The discovery of the user's profile is probably the most important part of Trillian's architecture. Its goal is to discover main document clusters using the analysis of visited documents (their contents) and an analysis of clickstream data. To discover 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 [26]. In our case, the query 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 [34] called the suffix tree clustering (STC). 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 the 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 [28] of all the words (strings) in a document set. Leaf nodes in this situation are marked not only with an 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 Ukonnen's algorithm because it uses suffix links for fast traverse of the tree [31].

3.1 Suffix Tree Clustering Process

Suffix tree clustering uses suffix tree data structure. Building the suffix tree is only one step in the whole document clustering process. 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.

A phrase cluster is a group of documents that all contain a common phrase; such a phrase designates the cluster and we shall call such a phrase a phrase cluster, too. A *maximal* phrase cluster is a phrase cluster whose phrase cannot be extended by any word in the language without changing (reducing) the group of documents that contain it.

These maximal phrase clusters are represented in the suffix tree by those internal nodes, whose leaves 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 the following formula:

$$s(m) = |m| \cdot f(|m_p|) \cdot \sum \mathit{tf*idf}(w_i),$$

where $|m|$ is the number of documents in a phrase cluster m, w_i are the words in a maximal phrase cluster and $\mathit{tf*idf}(w_i)$ is the 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. At this stage we can also apply scores of each word calculated by their position in the HTML document (e.g. in `<H1>` tag or `<B>`). A final score of each term can be obtained by multiplying its tfidf score and its HTML position score.

*tf*idf.* This measure (term frequency inverse document frequency) is a well known calculation, which evaluates the importance of a single word using an assumption that a very frequent word in the whole document collection has a lower importance than the one appearing less frequently among the documents.

After 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 [36].

Cluster merging. We need to identify those groups, which share the same phrase. By calculating binary similarity measure between each pair of maximal phrase clusters, we can create a graph where similar MPC's are connected by edges. Similarity between 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 various techniques:

- *Term tf*idf score. tf*idf* 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.

3.2 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.

4 CLICKSTREAM ANALYSIS

4.1 The Algorithm

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 at two points of the web communication. Web sites themselves 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 it tracks the user's activities within all web sites.

Data mining techniques can also be applied for clickstream analysis to achieve higher quality [20, 19]. Chen, Park and Yu have proposed some promising algorithms for mining for interesting patterns in clickstream data [4]. 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 shown on Figure 2.

```
while Clickstream not empty do
begin
  get Page;
  if Page.ClickData.http_Referrer is NULL
    then Page.TimeSpentOnPage = AverageTimeSpent
    else if Page.ClickData.http_Referrer is among previous records
      then if Page is member of frameset
        then assign Page to group of frameset pages and
          add the group to list of Documents
        else Page.TimeSpentOnPage = difference from previous page in the list
          add Page to list of Documents
      else Page.TimeSpentOnPage = AverageTimeSpent
end
```

Fig. 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 a 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. Cases of 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 becomes easier.

4.2 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 [33].

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 [22, 23] 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. [32, 24, 25, 13]. We can say that Trillian's architecture is ready to be extended for collaboration among users.

5 EVALUATION AND RESULTS

As a regular part of a software development process, we tested every module of the intelligent web tool architecture. Our primary interest was whether a specific module meets its requirements and whether its performance and results are satisfactory. However, from the scientific point of view, the attention should perhaps be focused more on modules where crucial or novel techniques are implemented. Therefore, in this section we focus on evaluation of the clustering quality.

Clustering effectiveness 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 information retrieval 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 judgements. We can then evaluate our clustering algorithm comparing its results to human categorisation 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 three testing collections were used: Syskill and Webert web page ratings, LISA collection, and 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.

5.1 Evaluation Methodology

For evaluation of clustering quality we used a common information retrieval 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 on the basis of a given cluster model, or, with the latter being 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 [36].

For numerical evaluation, we used two basic metrics: the precision factor and pairwise accuracy. We borrowed these metrics and methodology from [36], because we want to compare our results with this work.

Precision Factor. For each identified cluster we find the most frequent topic and consider it as "true" cluster topic. The precision is then calculated as the number of documents that belong to the "true" cluster divided by the total number of 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 documents that were clustered.

Pairwise 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 pairwise accuracy is calculated as follows, cf. also [36]. 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, respectively. Let $uncl(C)$ be the number of unclustered documents in C. We define the pairwise score of C, $PS(C)$, as:

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

where the summations are over all the 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 pairwise score of C, $maxPS(C)$, is:

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

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 pairwise accuracy quality measure of C, $PA(C)$ as:

$$PA(C) = (PS(C)/maxPS(C) + 1)/2.$$

5.2 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 [36] who compared six clustering algorithms including his version of the STC. They independently endorse the superiority of the STC which, together with the k-means algorithm, outperform the other algorithms.

5.3 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. The following figures 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

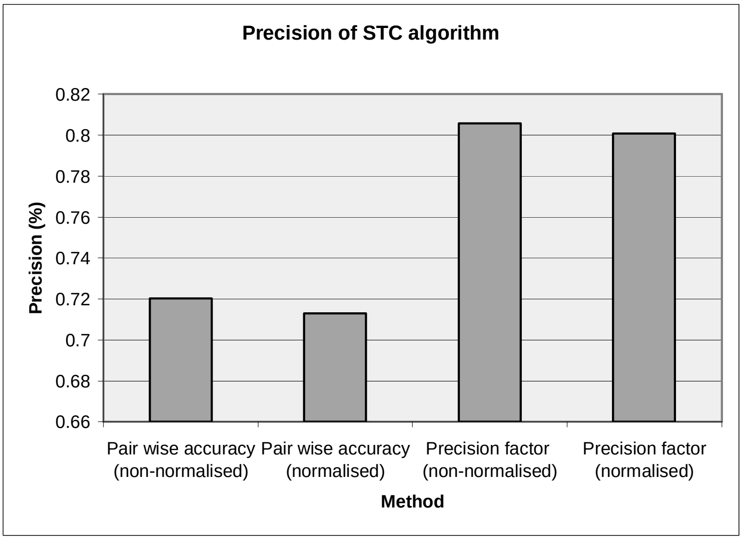

Fig. 3. Precision of Trillian STC algorithm

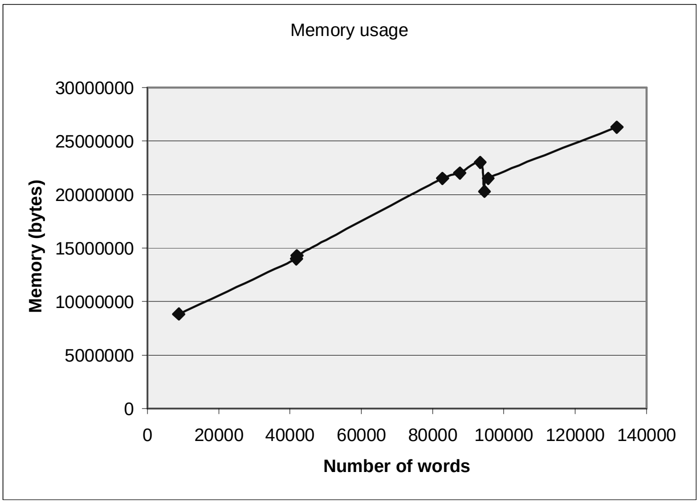

Fig. 4. Relation between number of words in document set and memory requirements

128 Mb RAM computer) without major problems. This number is not very high, when we imagine the number of documents, say, one week of browsing. It depends on the behaviour of the user, but in some cases it can be much higher than 800.

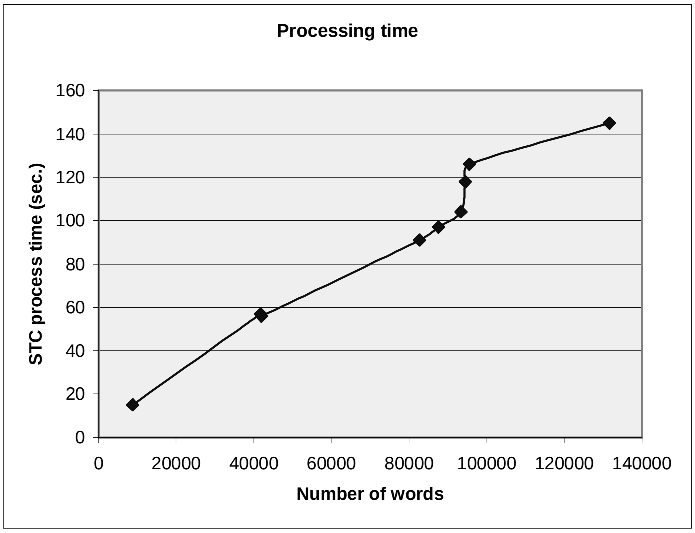

Fig. 5. Relation between number of words in document set and STC process time

Our implementation of the STC holds the whole suffix tree structure in the memory. Thus, when the document set is large, 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.

5.4 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 an experiment, we measured how many of the phrases contain more than one word. Figure 6 shows the percentage of single, double, triple and longer phrases in clusters on average.

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 information retrieval systems and tell us more about the context of the discovered topic.

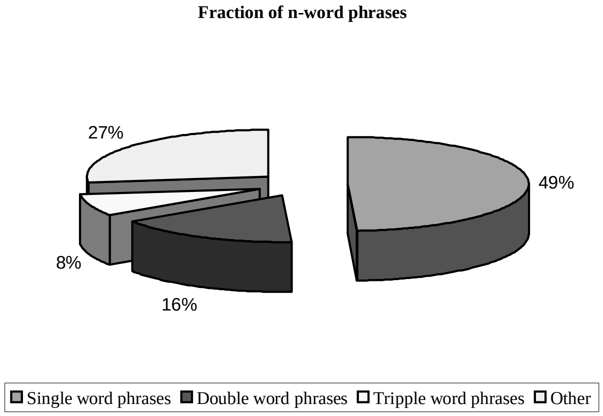

Fig. 6. Average fraction of the n-word phrases in clusters

6 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 from the web documents those potentially relevant to his interests.

The main question of this research was: "What is a feasible way of helping users 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 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. However, the presented prototype of Trillian may not qualify in its present design for the level 4 of the classification in [5], since to be "intelligent" probably requires more, e.g. to use ontologies, analysis of the contexts etc. On the other hand, the behaviour analysis is a step forward.

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. Another issue is whether STC can be built incrementally, and if yes, whether this can be done efficiently. For the first part, STC can indeed be built incrementally, since basically the idea of the suffix tree is to allow for incremental inclusion of phrases. However, its efficiency remains an open issue.

We have also identified the clickstream analysis as an important part of the whole 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 possible employment 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.

Acknowledgements

The paper is a revised and expanded version of a paper presented at the ADBIS 2002 Conference [12]. The work was partially supported by Slovak Science Grant Agency, grant No. G1/7611/20.

REFERENCES

[1] Armstrong, R.—Freitag, D.—Joachims, T.—Mitchell, T.: WebWatcher: A Learning Apprentice for the World Wide Web. In: Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous Environments, AAAI Press, 1995, pp 6–12.

[2] Bamshad, M.—Cooley, R.—Srivastava, J.: Web Mining: Information and Pattern Discovery on the World Wide Web, In: Proc. the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97), 1997.

[3] Catledge, L. D.—Pitkow J. E.: Characterizing Browsing Strategies in the World-Wide Web. Computer Networks and ISDN Systems, Vol. 27, 1995, No. 6, pp. 1065–1073.

[4] Chen, M. S.—Park, J. S.—Yu, P. S.: Efficient Data Mining for Path Traversal Patterns. IEEE Trans. on Knowledge and Data Engineering, Vol. 10, 1998, No. 2, pp. 209–221.

[5] Cheung, D. W.—Kao, B.—Lee, J.: Discovering User Access Patterns on the World Wide Web. Knowledge Based Systems, 1998, No. 10, pp. 463–470.

[6] Cormack, G.V.—Palmer, C. R.—Clarke, C. L. A.: Efficient Construction of Large Test Collections. In: Proc. the ACM Annual Conference on Research and Development in Information Retrieval (SIGIR 98), Melbourne, 1998, pp 282–289.

[7] Dorohocenau, B.—Nevill-Manning, C.: A Practical Suffix-Tree Implementation for String Searches. Dr. Dobb's Journal, 2000, No. 314, pp 133–140.

[8] Etzioni, O.—Weld, D. S.: Intelligent Agents on the Internet: Facts, Fiction and Forecast. IEEE Expert Vol. 10, 1995, No. 4, pp. 44–49.

[9] Hearst, M. A.: The Use of Categories and Clusters in Information Access Interfaces. In: T. Strzalkowski (ed.): Natural Language Information Retrieval, Kluwer Academic Publishers, Dordrecht, 1999, pp. 333–374.

[10] Kehoe, C.—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/, Accessed: November 24, 2000.

[11] Kovaľ, R.: Intelligent Support for Information Retrieval in WWW Environment. Master's thesis. Slovak University of Technology, Department of Computer Science and Engineering, 1999.

[12] R. Kovaľ—P. Návrat: Intelligent support for information retrieval in the WWW environment. In: Y. Manolopoulos, P. Návrat (eds.): Advances in Databases and Information Systems, Lecture Notes in Computer Science 2435, Springer Verlag, Berlin, 2002, pp. 51–64.

[13] Lashkari, Y.: Feature Guided Automated Collaborative Filtering. Master's thesis. MIT Department of Media Arts and Sciences 1995.

[14] Lawrence, S.—Giles, L.: Accessibility and Distribution of Information on the Web. Nature, 1999, No. 400, pp. 107–109.

[15] Lawrence, S.—Giles, L.: Inquirius, the NECI Meta Search Engine. Computer Networks, Vol. 30, 1998, Nos. 1–7, pp. 95–105.

[16] Lewis, D.: Learning in Intelligent Information Retrieval. In: Proc. 8th International Workshop on Machine Learning, 1991, pp. 235–239.

[17] Lou, W.—Liu, G.—Lu, H.—Yang, Q.: Cut-and-Pick Transactions for Proxy Log Mining. In: C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Bőhm, M. Jarke (eds.): Advances in Database Technology — EDBT 2002, 8th International Conference on Extending Database Technology. Lecture Notes in Computer Science 2287. Springer-Verlag, Berlin-Heidelberg-New York, 2002, pp. 88–105.

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

[19] Nanopoulos, A.—Manolopoulos, Y.: Finding Generalized Path Patterns for Web Log Data Mining. In: J. Štuller, J. Pokorný, B. Thalheim, Y. Masunaga (eds.): Current Issues in Databases and Information Systems, East-European Conference on Advances in Databases and Information Systems Held Jointly with International Conference on Database Systems for Advanced Applications (ADBIS-DASFAA 2000), Lecture Notes in Computer Science 1884, Springer-Verlag, Berlin-Heidelberg-New York, 2000, pp. 215–228.

[20] Pei, J.—Han, J.—Mortazavi-asl, B.—Zhu, H.: Mining Access Patterns Efficiently from Web Logs. In: T. Terano, H. Liu, A. L. P. Chen (eds.): Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, Lecture Notes in Computer Science 1805. Springer-Verlag, Berlin Heidelberg New York, 2000, pp. 396–407.

[21] Perkowitz, M.—Etzioni, O.: Adaptive Web Sites: an AI Challenge. In: Proc. 15th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, San Francisco, 1997, pp. 16–23.

[22] Polčicová, G.: Recommending HTML-documents Using Feature Guided Automated Collaborative Filtering. In: J. Eder, I. Rozman, T. Welzer: Advances in

Databases and Information Systems, Proc. 3rd East European ADBIS'99 Conference, Short Papers, Maribor, 1999, pp. 86–91.

[23] Polčicová, G.—Návrat, P.: Recommending WWW Information Sources Using Feature Guided Automated Collaborative Filtering. In: Z. Shi, B. Faltings, M. Musen: Proc. Conference on Intelligent Information Processing at the 16th IFIP World Computer Congress, Beijing, 2000, pp. 115–118.

[24] Polčicová, G.—Slovák, R.—Návrat, P.: Combining Content-based and Collaborative Filtering. In: Y. Masunaga, J. Pokorný, J. Štuller, B. Thalheim: Proceedings of Challenges, East-European Conference on Advances in Databases and Information Systems Held Jointly with International Conference on Database Systems for Advanced Applications (ADBIS-DASFAA 2000), Prague, 2000, pp. 118–127.

[25] Polčicová, G.—Návrat, P.: Semantic Similarity in Content-Based Filtering. In: Y. Manolopoulos, P. Návrat (eds.): Advances in Databases and Information Systems, Lecture Notes in Computer Science 2435, Springer Verlag, Berlin, 2002, pp. 80–85.

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

[27] Salton, G.—Buckley, C.: Term-Weighting Approaches in Automatic Text Retrieval. Information Processing and Management, Vol. 24, 1988, No. 5, pp. 513–523.

[28] Sedgewick, R.: Algorithms. Addison-Wesley, Reading 1988.

[29] Selberg, E.—Etzioni, O.: The MetaCrawler Architecture for Resource Aggregation on the Web. IEEE Expert, 1997, pp. 11–14.

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

[31] Ukkonen, E.: On-line Construction of Suffix Trees. Algorithmica, Vol. 14, 1995, pp. 249–260.

[32] Ungar, L.—Foster, D.: Clustering Methods for Collaborative Filtering. AAAI Workshop on Recommendation Systems, 1998.

[33] Yu, H.—Breslau, L.—Shenker, S.: A Scalable Web Cache Consistency Architecture. In Proc. ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '99), Vol. 29, 1999, No. 4, No. 163–174.

[34] Zamir, O.—Etzioni, O.: Web Document Clustering: A Feasibility Demonstration. In: Proc. 19th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98), 1998, pp. 46–54.

[35] Zamir, O.—Etzioni, O.—Madani, O.—Karp, R. M.: Fast and Intuitive Clustering of Web Documents. In: Proc. 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97), 1997, pp. 287–290.

[36] Zamir, O.: Clustering Web Documents: A Phrase Based Method for Grouping Search Engine Results, University of Washington, 1999.

Robert Kovaľ received his Bc. and Ing. (Master) degrees in computer science and engineering from Slovak University of Technology, Bratislava in 1999 and 2001, respectively. He currently works for Interactive Ltd., Dubai in its Internet division as software development team leader. His research and professional interests include Internet protocols, data mining, adaptive web sources, web analysis, software development processes and configuration and versioning issues.

Pavol Návrat received his Ing. (Master) summa cum laude in 1975, and his PhD. degree in computing machinery in 1984 both from from Slovak University of Technology. He is currently a professor of informatics at the Slovak University of Technology. During his career, he was also with other universities overseas. His research interests include related areas from software engineering, artificial intelligence, and information systems. He published numerous research articles and the following books: Programming in Lisp, Microcomputers, Programs, People, and textbooks Functional and Logic Programming, and Artificial Intelligence. He co-edited and co-authored monographs: Knowledge-Based Software Engineering (Amsterdam, IOS-Press, 1998) and Advances in Databases and Information Systems (Berlin, LNCS Springer, 2002). He was editor of a special issue on Knowledge-Based Software Engineering of the journal Informatica in 2001. Prof. Návrat is a Fellow of the IEE and a Senior Member of the IEEE and its Computer Society. He is also a member of the ACM, American Association for Artificial Intelligence, Association for Advancement of Computers in Education, Slovak Society for Computer Science and Slovak Artificial Intelligence Society.