A Theme-based Search Technique

Nida Al-Chalabi & Khalil Shihab
Department of Computer Science, Sultan Qaboos University, Box 36, Al-Khod, Oman

Abstract: - This work presents an intelligent search engine, called ORCA, that returns the most relevant documents for user's queries. This search engine analyses the queries and builds themes (models) to be used when the engine is confronted with similar queries. The intelligent component is used for constructing a model of the user behavior and using that model to fetch and even pre-fetch information and documents considered of interest to the user. It uses both latent semantic analysis and web page feature selection for clustering web pages. Latent semantic analysis is used to find the semantic relations between keywords, and between documents.

Keywords: Information Filtering, User's Model, Search Engines, Latent Search Analysis.

1 Introduction

At the current explosive growth rate of the Web, Internet users are finding it more and more frustrating to effectively retrieve the desired information[1, 2, 3]. Current search engines usually flood the users with the number of pages and URLs for a given search query. These engines are based on first identifying keywords from the user queries and, secondly, searching the web to find the pages according to the frequency of these keywords in the HTML pages. The disadvantage of this approach is that the results of these search engines usually contain a lot of unwanted information, which does not relate to the users actual requirements. In other words, they have lots of "noise". With this type of blind search and lack of automatic filtering capability, traditional search engines make it difficult for the users to find new information about the same topic.

This work addresses this very problem and aims at coming up with a design for a filtering system for the Web searching which is based on the interests of individual users, and has the capability for making intelligent decisions in regard to what the user desires to access. In summary, the filtering system should give the users only the information they want, discarding irrelevant and duplicate pages; thus saving the users time and effort while surfing the Web. In other words, search engines should not bother the users with unwanted knowledge.

Moreover, the search engines should incorporate some sort of intelligent capability to help the users in locating information related to their interests. This implies that an engine should possess learning capability for building a model of the user behavior while surfing the Web. This model can be constructed from the user access log files, and can be used to access. In the final section of this paper, we identify some key components of such an intelligent component.

Recently, several additional search engines which are described to be intelligent have been introduced. These engines mainly depend on automatically discovering the users access patterns while accessing the Web and use that knowledge for future retrieval of information relevant to the interest of the users. Examples of such engines include WebWatcher [3] and Letizia [4]. They aim at automatically performing some of the Web exploration on behalf of the user to pre-fetch what is considered of interest to the user.

WebWatcher is basically a tool which aims at helping the user by interactively giving him advice during the navigation. While the user is browsing through the pages, WebWatcher assesses the links contained in the pages, and then recommends those links which are considered more promising in meeting the requirements of the search criteria, i.e. the goal. Letizia is an intelligent agent which works with conventional browsers such as Netscape. It tracks the user behavior during browsing and attempts to figure out his goals and hence build a model of his interests. It then uses that model to search for pages considered relevant to the goals embodied in the user model. The goal here is to conduct some exploration of the Web on the behalf of the user. Obviously, in such automatic exploration, too many noises are expected. For example, some pages are visited by the user not because they are of any interest, but simply either they have some intermediate links which need to be followed to get to the desired page or they have been mistakenly visited. Unfortunately, both of the above engines lack efficient techniques to deal with this last issue.

In short, intelligence claimed to be embodied in these engines lie in their ability to model the user's access behavior and use that model to anticipate and predict future access patterns. It is worth noting that these intelligent search tools are still in their experimental stages. It will be interesting to see how effective these tools are after some performance evaluations are reported. It goes without saying that any attempt to include some sort of intelligence, as the one described above, would require incorporating complicated machine learning and automatic user modeling techniques in the design of search engines.

2 Themes – User's Behavior Models

In this approach, a search engine should come up with the models that represent best of the user access patterns and various areas of interest. We use *explicit planning mechanism* to model the user behavior on the Web. In this mechanism, the user is required to explicitly spell out his/her interests that which externalize and represent his access patterns. In this way, the information about the user overall access behavior on the Web is analyzed by the search engine. This approach greatly simplifies the problems associated with designing systems for intelligent or automatic deduction of the user access patterns.

2.1 The Architecture

Here we would like to lay down an architecture of the User' Model (Theme) component of the system, which is placed on top of the existing search engines. The system is written in Visual Basic and is designed to help the users in avoiding seeing undesired information when carrying out the search.

At the architectural level (shown in Figure 1), the system consists of two main components, namely the **Theme Manager** and the **Processing Manager**.

The Theme Manager serves several purposes besides being the interface which communicates with the users. As an interface, the Theme Manager allows the user to establish his/her personal profile, which includes such items as (1) set of keywords (search criteria) which captures the topic or concept of interest to the user, (2) the desired search engine and (3) the options for live update. The Theme Manager maintains individual user models in the Theme Database. The Manager is also responsible for submitting the queries to the search engines and accessing the Theme Database where the results of the search are stored, organized and managed by the Processing Manager.

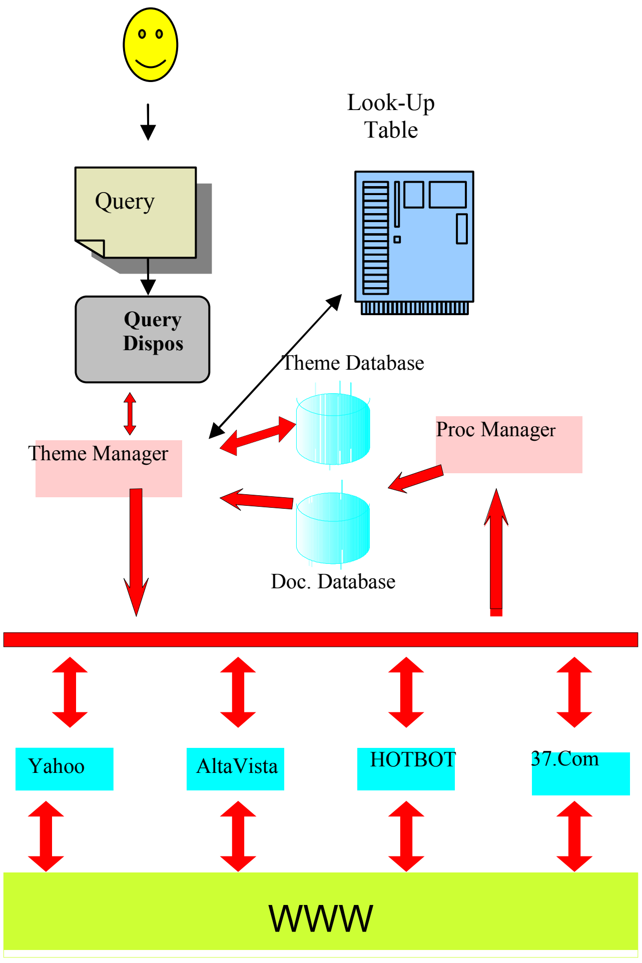


Figure 1: System Architecture

2.2 The Operations

After retrieving or creating themes for the current user, the system displays the Current Themes and Search Criteria window on the screen and allows the user to add, modify, or delete his search preferences. For example, the user can enter four search criteria, but decide to submit only the first and the last ones for search. The search engines selected can be Yahoo for the first criteria and 37.Com for the last criteria.

After the user submits the searches, the system processes the user's query using the noise-disposal procedure in order to produce a query consisting of keywords only. It, then, uses these keywords to look up for a theme from a table called *look-up table*, which contains all the themes in the database. Using a partial matching process, the system tries to match the first keyword in the user's query with the first column in the table. If a match is found, it checks the next field that contains the themes or tables related to the first keyword and once again it tries to find another match. This process will continue until the system finds the theme that contains all the URLs. The process continues in the same manner with the other keywords in the user's query.

The system also deals with empty queries, the case of having only one word and this word is final sub theme, and the case of none of the given keywords is a final sub-theme and the case when the query doesn't exist in the database. For the last case, the system creates a new table with its description as the query keywords and sends the queries to the search engines selected and the results produced are fed into the Processing Manager. The Manager compiles the lists of URLs from the engines, merges them, deletes duplicates, and finally produces a list of URLs which satisfy the queries submitted. As documents arrive, the Processing Manager classifies URLs and stores these results into the Document Database.

The Process Manager uses two types of features extraction, latent semantic analysis and web page features selection, for web page clustering using our conceptual clustering technique (see section 5). The latent semantic analysis (see section 4) extracts common semantic relations between keywords and a document. The web page feature selection extracts the text features from a given web page for inputting to our clustering technique. Latent semantic analysis is used to find the semantic relations between keywords, and between documents. The latent semantic analysis method projects terms and a document into a vector space to find latent information in the document. At the same time, we also extract text features from web page content.

```
Var
 x, i  : integer
word, query :  string
find   : Boolean
begin
  x = 0
  Size = 0 //size of the array of keywords of query.
  query = bring the user's query from the HTML
            interface "project.asp"
  while (query is not "  ")
  begin
    x = find the position of the first
          white space " " in query
    word=starting from left of query
      take the sub string from the first index till (x-1)
    find=check if word is a noise "false if it's a noise"
   if   find   then
     begin
         keywordArray (Size) = word
         increment Size by 1
     end       // if
   query=update query to start from
            x till length of query
 end           //while
end.
```

3 System Implementation

3.1 Query Dispose

At this stage, the query is modified by removing all the noise and all the unnecessary words. The system, then, converts the query into individual keywords. What we mean by noise is the words that are helpless in the searching process such as find, about and without. For example, if the user query is "training courses of diploma in computer science" so the pure query is "training courses diploma computer science". These keywords are stored in a vector called terms-vector.

This sub routine is written using Visual Basic Script and ASP.

3.2 Look-Up Table

This table is a two-dimensional array in which the first column consists of all the tables (or themes) available in the system's database. The cells in each row contain the tables (themes) related to the first entry in each row. The last filled entry in each row is indicated by "*" to show that this is the last entry. This is due to the various numbers of sub themes related to each theme. We have used "$" to indicate that this theme is a final sub theme.

For an experiment, we created a database containing three main themes: e*ducation, business and computer*. The database is created in Microsoft Access. There are 59 tables in total for all the three themes.

Now consider the *business* theme along with its only three related sub themes, which are *economics, finance* and *marketing*. If we trace the *finance* sub theme, we get the following sub themes*: banking, accounting, planning, investment services, etc*. If we trace the theme *investment service*, the related sub themes beneath it are: *brokerages* and *socially responsible investment.* If we continue with *brokerages,* we get *bonds* and *money* as a final sub theme. This technique is used for all the tables to get to the final sub themes. The following shows the scheme that was stated above.

Business theme

Lookup (9, 1) = "business"
Lookup (9, 2) = "economics"
Lookup (9, 3) = "finance_and_investment"
Lookup (9, 4) = "marketing_and_advertising"
.

```
.
.
Lookup (9, 23) = "brokerages"
Lookup (9, 24) = "bonds"
Lookup (9, 25) = "money"
Lookup (9, 26) = "*"
```

Finance Sub Theme

```
Lookup (29, 1) = "finance_and_investment"
Lookup (29, 2) = "banking"
Lookup (29, 3) = "accounting"
Lookup (29, 4) = "planning"
Lookup (29, 5) = "investment_service"
.
.
.
Lookup (29, 13) = "*"
```

Investment Service Sub-Theme

```
Lookup (12, 1) = "investment_service"
Lookup (12, 2) = "socially_responsible_investment"
.
.
.
Lookup (12, 6) = "*"
```

Brokerages Sub Theme

```
Lookup (8, 1) = "brokerages"
Lookup (8, 2) = "bonds"
Lookup (8, 3) = "money"
Lookup (8, 4) = "*"
```

Bonds Final Sub Theme

```
Lookup (7, 1) = "bonds"
Lookup (7, 2) = "$"
```

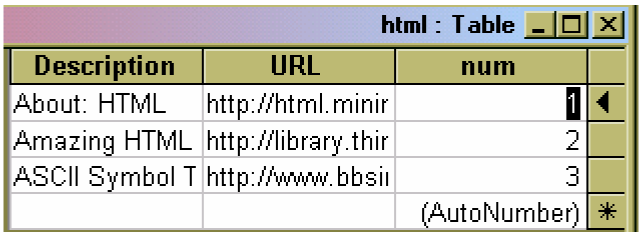
Money Final Sub Theme

```
Lookup (15, 1) = "money"
Lookup (15, 2) = "$"
```

3.3 Database

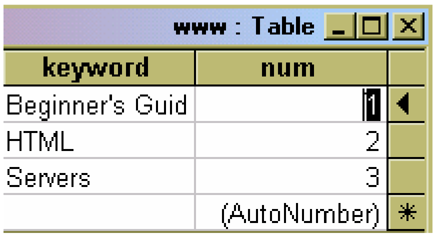
The database is created in Microsoft Access and it initially contains 59 tables. There are two types of tables: final sub theme tables and other theme tables. The final sub theme tables are at the bottom level of each theme (or the most sub theme).

Table 1: html table, which is an example of the final sub-theme.


html : Table

Description	URL	num
About: HTML	http://html.minir	1
Amazing HTML	http://library.thir	2
ASCII Symbol T	http://www.bbsii	3
		(AutoNumber)

Table 2: www table, which is an example of a theme table.


www : Table

keyword	num
Beginner's Guid	1
HTML	2
Servers	3
	(AutoNumber)

To retrieve the fields in the final sub theme tables, we use the following algorithm:

```
dim objrs
  set objrs = Server.CreateObject("ADODB.Recordset")
  connectme = "DSN=dsnname"
  sqlstmt = store here the query to retrieve the fields from
a specified table
  objrs.open sqlstmt,connectme
  Do While not objrs.EOF
       response.write objrs("URL")
       response.write  objrs("Description")
       objrs.movenext
   loop
```

3.4 Theme Search

In order to complete this stage we need four major subroutines: first_col (), leaf_node (), search_row () and findtheme ().

The first_col () subroutine gets a string and tries to find a match in the lookup table by searching the first column, which contains all the themes and sub-themes available in the database, and returns an integer number, which represents the row where the theme can be found. It searches the lookup table using a partial matching technique.

The leaf_node () subroutine takes an integer, which represents the row number. It should check whether the string is in first column and the given row is a final sub-theme. It returns a Boolean value true if it is a final sub theme and false otherwise.

```
x = first_col (current keyword)
 leaf_node (x)
'if the first keyword is not a final sub theme
  begin
    While (count <= array size) And (final sub theme is not
      found) do
      begin
         if (current keyword is not available) then
             create a new table in the database
         Else
           begin
             leaf_node( a given row)
             if the given keyword is final sub theme
               begin
```

```
                tablename = lookup (count, 1)
                 sqlstmt = select required fields from this
                 final sub theme
                 retrieve the urls from database.
            End
             Else
              begin
               Bool=search_row(a given row, key (count+1))
               if bool then
                   increment count
               End
        End
    End
   Else if the first keyword is final sub theme then
           retrieve the urls from the database.
```

The search_row () sub routine accepts an integer representing the row number and a string. It returns a Boolean value true if the string is found in the indicated row, otherwise it returns false.

```
While (lookup (row number, count) != "*") &&
          (string not found)
 begin
  if some letters in lookup (row number, count)
   match some letters from string (partial matching) then
         return true
  Else
         Increment count
 end
```

Finally, the findtheme () is a search algorithm takes each keyword from the vector of keywords one at a time. It searches the lookup table by passing each keyword to first_col (). The findtheme() returns the row number. If it is a final sub-theme, the system retrieves the required fields (Description and URL). Otherwise, the system should carry out a further check. If the keyword cannot be found in the database, the system should expand the database using initially this keyword as a title. If the keyword exists in the database, the system will check if the next keyword in the vector of keywords in the row of the previous keyword. If the keyword is there, the process is repeated using the remaining keywords. But if the next keyword doesn't exist in that row, then the system has to create a new theme.

3.5 Expanding the Database
The database should be expanded when the query entered by the user has not found in the local database. The system should, therefore, adds a new table (theme) with query as an initial name of the new table. The system
starts with empty new theme (no records). The records will be added manually. The following code is responsible for this issue:

```
DIM strSQL
   strSQL = "CREATE TABLE" & " " & tablename   " &
& ")" & "_
              " num Counter Primary Key,"  &_
              "URL  Text(50), "  &_
              "Description  Text(50) "  &_
             ") ";
DIM Conn
 SET Conn= Server.CreateObject("ADODB.Connection")
   Conn.Open "sun","",""
  Conn.Execute (strSQL)
  Conn.close
  set conn=Nothing
```

4 Latent Semantic Analysis (LSA)
Latent semantic analysis applies a vector space concept [5]. All keywords and documents form a two-dimension term-document matrix; singular value decomposition is used to decompose the term-document matrix to obtain the semantic features. Suppose that X is a term-document matrix, which is which is an $i{\times}j$ matrix; where i is the number of keywords and j the number of documents. Each element $X[i,j]$ is the number of occurrences of keyword i in document j. The SVD computes the matrix singular value decomposition of X. It uses the eigenvalue and eigenvector to reduce the dimensions of the original data, filtering irrelevant information. The original matrix has a high dimension. An SVD can reduce the original high term-document matrix dimensions to a low term-document matrix. The Matlab command [U,S,V] = svd(X) produces a diagonal matrix S of the same dimension as X, with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V'.

5 Clustering Technique
Our clustering technique is computationally efficient and has high classification accuracy [6]. The basic idea of the method is simple yet effective. It uses a control-generate-test strategy, which involves the generation of a set of dynamic filters (concepts) and then testing these concepts on given data vectors (components) for detecting clusters of these components.

The clustering technique is based on using the identity matrix (I), i.e. a matrix that contains only the base vectors, rather than using a random matrix (RM). These vectors are, therefore, considered as the seeds for producing the required filtering concepts. Using this method, the algorithm operates in O (N) time; where N is the number of base vectors. It provides great speed and efficiency improvement over the first method and over the methods currently used for clustering.

Concepts construction:

Prepare a suitable identity matrix **I** of base vectors and the data matrix that is required to be divided into groups. The number of the elements (patterns) of the data matrix should constitute the dimensions of the identity matrix **I**. Thus, if the data matrix has ***n*** elements, then the dimensions of the identity matrix **I** should be $n \times n$.

1. Use the data matrix to produce a category matrix or pattern matrix **PM**$=[pv_i]^T$ of pattern vectors (pv_i). This requires a transformation of data or smoothing. We have used $\log_{10}$ for the transformation. This step is not necessary but it provides further performance enhancement and, as stated before, it can only be carried out if we know in advance the categories of each attribute in the data matrix.

2. If the number of columns of the pattern matrix **(PM)** is less than the number of columns of the identity matrix I, then fill the missing data of PM with averages to produce a new matrix, namely IM. This step is required in order to have the same dimensionality for both **PM** and **I**.

3. Use the identity matrix **I** (or **IM**), and the pattern matrix (**PM**), and go throughout the following steps to generate a concept matrix **CM**$=[cv_i]^T$ of concept vectors along with its associated distance vector **DV**$= [dv_i]^T$.

```
For i =0 to the last pattern in PM do
 Begin
  For j=0 to the last random vector in IM do
   Begin
    a. Find the closest base vector $bv_j$ in IM to $pv_i$ in PM.
    b. Calculate the distance before alignment (bdi):
       $bdi = ||bv_j - pv_i||$, and save the Result in $BD=[bd_i]^T$
    c. Align $bv_j$ and $pv_i$ as follows:
          $bv_j = bvj + \alpha ||bv_j - pvi||$; $0<\alpha<1$.
    d. Replace $bv_i$ with the result of the alignment to
       produce a candidate concept.
  End;
 End;
```

4. Remove the base vectors that are not used from **IM** and save the result as a concept matrix (**CM**) along with the distance vector (**BD**).

5. Eliminate the redundant concepts; concepts are considered redundant if the same clustering result can be reached with using only the remaining concepts. The elimination should be carried out as follows:

a. Let $ad_i= ||pv_k - cv_i||$; k=0, 1, 2, …, n be the distance after the alignment.

b. For all i and j such that $i \neq j$ if $||cv_i - cv_{ji}|| \leq ad_i$, then eliminate cv_{ji}

Clusters generation

6. At the start of iteration i, select a concept vector (cv_i) and its associated distance (bd_i) at a random.

7. Find the pattern vectors that satisfy the inequality, given below, and place them in the concept group i (**CG$_i$**)

Let $ad_i= ||pv_k - cv_i||$; k=0, 1, 2, …, n be the distance after the alignment and bd_i is the distance before the alignment. For a pattern vector k to be placed in group i, it must satisfy the following inequality:

$$||bd_i - ad_i||<T_i \text{ where } T_i=ad_i*\propto; \quad 0<\propto<1.$$

8. Stop when you reach an empty pattern matrix (PM). Otherwise, increase $\propto$ slightly.

9. Reset **PM** and **CG$_i$** for all i, and repeat from step 1.

6 Conclusion and Further Work

This paper has presented a prototype system to improve the efficiency of the Web search engines. The main goal of the search tool described here is to overcome the drawbacks found in the current search engines, which are related mainly to their being one-time, unintelligent search engines. The novelty of the design presented lies in its ability to support intelligent (automatic) approaches. The retrieved documents are grouped into clusters using our clustering technique that accepts as input the web features and the semantic features produced by LSA. Through this approach, the users are always presented with updated information and are never flooded with duplicate information every time a query is submitted.

References:

[1] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/

[2] http://www.netscape.com/

[3] Armstrong et al. WebWatcher: A Learning Apprentice for the World Wide Web.

[4] H. Lieberman. Letizia: An Agent that Assista Web Browsing , *IJCAI'95*, 1995.

[5] H. Ramadhan and K. Shihab. Improving the Engineering of Search Engines for the Web, Proceedings of the Intl. Conference on Internet Computing, 2002, pp 188-193, USA.

[6] Landauer, T. K., Foltz, P. W., & Laham, D. Introduction to Latent Semantic Analysis, *Discourse Processes*, 25, 1998, pp. 259-284.

[7] Shihab, K. Improving Clustering Performance by Using Feature Selection and Extraction Techniques, Journal of Intelligent Systems, Vol. 13, No. 3, 2004, pp 249-273, 2004.