A PRUNING ALGORITHM FOR IMPROVING WEB RESOURCES SEARCHING AND CLASSIFICATION IN KNOWLEDGE MANAGEMENT-BASED WEB SYSTEMS

Alfredo Cuzzocrea
DEIS Dept. – University of Calabria
via P. Bucci, 41C 87036 Rende, Italy

Carlo Mastroianni
ICAR - CNR
via P. Bucci, 41C 87036 Rende, Italy

ABSTRACT
Knowledge Management-based Web Systems (KM-bWS) are a novel class of Web Information Systems whose main goal is to adapt contents and presentations with respect to user needs and backgrounds through the execution of Knowledge Management processes. A KM-bWS can be seen as an “intelligent knowledge hub” because it makes distributed Web resources available by means of Knowledge Management techniques such as classification and clustering. In this paper we present a reference architecture for KM-bWS, and a pruning algorithm, called KM_search, that allows to improve Web resources searching and classification. This algorithm evaluates the relevance of Web resources for different classes of users, by matching search keywords with the content of Web pages. The Web page section where keywords are found as well as the position of the page inside the search tree both concur in the evaluation of Web resources relevance.

KEYWORDS
Algorithms, Web Engineering, Knowledge Management.

1. INTRODUCTION
Knowledge Management (KM) is an emerging area, which is gaining interest by both enterprises and academics. KM methodologies allow to create value from intangible assets of an enterprise and are an important issue for multinational corporations as well as for small business enterprises. The goal of a general purpose Knowledge Management System (KMS) is to provide the right knowledge to the right people at the right time and in the right format [1-4]. Knowledge Management-based Web systems (KM-bWS) are a very recent class of service-oriented Web Information Systems that is rapidly becoming popular. The major goal of these systems is to adapt their contents and presentations with respect to user needs and backgrounds through the execution of KM processes. The core layer of KM-bWS exploits complex methodologies such as knowledge representation, classification and clustering, reasoning, and, more recently, ontologies and semantic Web. In particular, knowledge representation has a leading role both in the KM process designing and in the developing of techniques and algorithms able to infer new and useful knowledge from explicit knowledge. As a consequence, the integration between novel KMS and traditional, well-known techniques such as Knowledge Discovery in Data Bases (KDD), Data Mining (DM) and Pattern Discovery in large data sets, is as an innovative and exciting research area.

Following this direction, we designed a methodology for developing KM-bWS that are able to semi-automatically retrieve, classify and publish Web resources belonging to a particular application domain. The Web publishing process is “adaptive”: each Web resource can be accessed by different user classes (classes are defined by the system author on the basis of his/her knowledge about the application domain): at run-
time, the system identifies the class of the current user and selects the “system view” associated to that class. A system view is a collection of Web resources available to a certain class of users. According to this approach, a user accesses only the knowledge that is significant for her/him and is not overwhelmed by not interesting data.

Though our methodology is “orthogonal”, i.e. it is applicable to different application domains, we have exploited it in an archaeological application domain. We have managed the knowledge concerning the Sibari Italian archaeological site and have developed the Sybaris Knowledge Portal (SKP), a knowledge Web portal able to provide information about the Sibari site and correlated themes. We have also designed and developed the correspondent authoring tool (the Sybaris Knowledge Portal Authoring Tool - SYKPAT) that completely supports the three fundamental phases (retrieval, classification and adaptive Web publishing) that compose the process of developing a KM-bWS. This experience has been conducted in the context of the project Valorizzazione del Patrimonio Archeologico e Monumentale della Regione Calabria – n° 50-pb/232 of the European development program Programma Operativo Plurifondo 94/99 – Misura 4.4, by the High Performance Computing and Networking Institute of the Italian National Research Council (ICAR-CNR).

As mentioned above, knowledge representation has a crucial role: in order to develop a KM-bWS, an author should deal with the knowledge representation issue in the context of the specific application domain. Another topical issue is the inference of novel knowledge from the explicit one, e.g. by means of a set of logical rules. For our application example, we have adopted the Consortium for the Computer Interchange of Museum Information Dublin Core (CIMI-DC) metadata format [9] for the knowledge representation of resources of archaeological interest. The Dublin Core (DC) format [10] defines a metadata standard that supports a broad range of purposes and business models, while the CIMI-DC format is the specialization of DC for representing museum information. Having chosen the CIMI-DC standard, we developed a methodology for the semi-automatically building of XML-based CIMI-DC metadata that represent Web resources.

In general, letting $A$ be the application domain of a KM-bWS, the most critical phase is the defining of a knowledge representation model suitable for the basic components of that domain (for example, in our case the basic components are the Web resources of archaeological interest). Formally, we can describe the representation model as a function $K$ that, for a basic component $c$ belonging to the application domain $A$, returns the (XML) metadata set $S_c$ that describes $c$ with respect to the representation model chosen for $A$.


The remainder of the paper is organized as follows: in Section 2 we briefly present a reference architecture for KM-bWS; in Section 3 we present our pruning algorithm and some experimental results that confirm its effectiveness in reducing the search space size; in Section 4 we give conclusions and hints about future work.

2. A REFERENCE ARCHITECTURE FOR KNOWLEDGE MANAGEMENT-BASED WEB SYSTEMS

In this Section we describe a reference multi-layer architecture for KM-bWS. The main functionalities provided by this architecture are the following (Figure 1):

- **Searching and retrieval of Web resources**: searching is performed by applying string pattern matching and rules that allow to achieve search space reduction (Knowledge Retrieval Layer);
- **Semi-automatic Web resource classification**: Web resources are selected by the system author and then are classified with respect to the current knowledge representation model of the system (Knowledge Engine Layer);
- **Providing of adaptive Web resources**: Web resources are specialized, and published, with respect to the system views associated to different classes of users (Knowledge Portal Layer).
As shown in Figure 1, the Knowledge Retrieval Layer is the software layer that deals with the searching and retrieval of Web resources and also with the semi-automatic building of HTML/XML wrappers. The software component that supports the searching of Web resources, not shown in Figure 1, is in charge to deliver the proper HTML pages to the Knowledge Engine Layer. The searching function is provided by a pool of concurrent search mobile agents that surf the Web to search HTML pages that contain the search keywords. We have used the Java Agent DEvelopment Framework (JADE) [16] as a software platform for supporting search agents. JADE, fully implemented in Java by the Telecom Italia Lab (TILAB), defines a multi-agent platform according to the Foundation for Intelligent Physical Agents (FIPA) specifications [17]. We have used the JADE API collection to customize the behaviour of the mobile agents in order to reduce the search space size.

For the Knowledge Engine Layer, we have chosen the XML language as a data representation and interchange format and Java for the application logic layer. Wrappers are needed to transform HTML pages into the XML language. In our implementation, the wrapper generator is LixTo [18], whose main feature is the ability to activate a former generated wrapper several times in order to retrieve relevant information from HTML pages that are “similar” to the original HTML page used for generating the wrapper. In such a way, it is possible to build HTML page clusters and improve the knowledge discovery process.

The Knowledge Engine Layer is the software layer that extracts novel knowledge starting by the known one. The knowledge discovery and the knowledge inferring models are strongly dependent on the application domain. In general, the more complex is the application domain, the more sophisticated is the knowledge inferring model.

3. SEARCHING POLICY AND REDUCTION OF THE SEARCH SPACE SIZE

In this Section we describe the searching policy that is exploited by search agents. We used a state abstraction-based technique to describe the dynamic behaviour of search agents: a search agent, being in state $S_i$, makes a state transaction towards another state $S_j$ on the basis of the current state $S_i$, and on the path followed from the root of the search tree to the current HTML page. The search space is represented as a HTML page tree: in fact we avoid the presence of cycles by discarding already visited HTML pages. The root of the search tree is the initial URL from which all search activities start, and the maximal value of the tree depth is a search parameter that can be set by system author. Each HTML visited during the search process is labelled with a Boolean value that indicates if the page is interesting or not for the search activity under consideration. So, if $\Phi$ is the set of visited HTML pages set and $\delta$ is the set of HTML pages of interest, we have $\Phi \supseteq \delta$. In order to reduce the size of the search space, we have designed and tested a cost function $f_C(\cdot)$: the domain of this function is the set of HTML pages that belong to the search space set, while the co-domain is the set of real numbers. The searching policy exploited by search agents is the following: letting $V$
be an "interest threshold", that is set by the system author, and \( P_A \) the current HTML page reached by agent \( A \). If \( f_c(P_A) > V \) then agent \( A \) updates the set of visited HTML pages set by adding \( P_A \), labels \( P_A \) with the TRUE value (so, \( P_A \in \Phi \) and \( P_A \in \emptyset \)) and continues the navigation from \( P_A \); otherwise, if \( f_c(P_A) \leq V \) agent \( A \) adds \( P_A \) to the visited HTML pages set, labels \( P_A \) with the FALSE value (so, \( P_A \in \Phi \) and \( P_A \in \emptyset \)) and continues the navigation from \( P_A \), where \( P_A \) is the last item inserted in the set \( \emptyset \).

The algorithm \( K M\_search \) implements the searching policy described above: it uses the procedure \( buildPageSet \) that builds the set of pages to be processed, starting by a current page \( p \): in turn, \( buildPageSet \) uses the procedure \( getLinkedPages \), that returns the set of pages linked to a given page, and the procedure \( appendChildrens \) that appends a list of nodes as children of a given parent node.

<table>
<thead>
<tr>
<th>Procedure: buildPageSet</th>
<th>Algorithm: KM_search</th>
</tr>
</thead>
<tbody>
<tr>
<td>input a page ( p ) and a threshold ( V )</td>
<td>input the initial page ( p ) and the max tree depth ( M )</td>
</tr>
<tr>
<td>output a pages to the set ( \Omega )</td>
<td>output the set of visited pages ( \Phi )</td>
</tr>
<tr>
<td>begin</td>
<td>begin</td>
</tr>
<tr>
<td>Vector ( \Omega = \text{null} );</td>
<td>Tree searchTree = new Tree();</td>
</tr>
<tr>
<td>( \Phi).add(p);</td>
<td>searchTree.setRoot(p);</td>
</tr>
<tr>
<td>if ( f_c(p) &gt; V ) {</td>
<td>Vector nodeVector = new Vector();</td>
</tr>
<tr>
<td>( p).isOfInterest = TRUE;</td>
<td>nodesToBeProcessedSet.add(p);</td>
</tr>
<tr>
<td>( \Omega = p).getLinkedPages(); }</td>
<td>Page currentNode = null;</td>
</tr>
<tr>
<td>else{</td>
<td>int ( j = 0 ); int ( i = 0 );</td>
</tr>
<tr>
<td>( p).isOfInterest = FALSE; }</td>
<td>Vector tempVector = null;</td>
</tr>
<tr>
<td>return ( \Omega );</td>
<td>while (( j \leq M &amp;&amp; \text{nodeVector.size()} &gt; 0 ){</td>
</tr>
<tr>
<td>end</td>
<td>currentNode = nodeVector.get(0);</td>
</tr>
</tbody>
</table>

Search operations are performed with a pattern matching between the search keywords and the current HTML page. In order to define the search function \( f_c(\bullet) \), we first introduce the function \( \delta(\bullet, \bullet) \): it is a two variable function that, for each pair \( \langle p, k \rangle \), where \( p \) is an HTML page and \( k \) is a search keyword, returns 1 if \( k \) is contained in \( p \), 0 otherwise. The relevance of a search keyword in an HTML page is determined by considering the position of the keyword inside the HTML page. For example, given a search keyword \( k \), a page \( P_i \) that contains \( k \) in its \( \text{HEAD} \) section is more interesting than a page \( P_j \) that contains \( k \) in its \( \text{BODY} \) section. To this aim, we have defined the function \( \text{pos}(\bullet, \bullet) \) that, for each pair \( \langle p, k \rangle \), where \( p \) is an HTML page and \( k \) is a search keyword, returns an integer number related to the position of \( k \) inside the HTML code of \( p \): in order to provide more flexibility, the co-domain of the \( \text{pos}(\bullet, \bullet) \) function is composed by a set of weights that correspond to different levels of interest for that page.

The formal definitions of these two functions are:

\[
\delta(p, k) = \begin{cases} 
1 & \text{if } k \in p \\
0 & \text{otherwise} 
\end{cases} \quad \text{(1)}
\]

\[
\text{pos}(p, k) = \begin{cases} 
w^k_0 & \text{if } k \in \text{TITLE}(p) \\
w^k_0 & \text{if } k \in \text{HEAD}(p) \\
w^k_0 & \text{if } k \in \text{BODY}(p) \\
w\_k & \text{if } k \text{ is a link} 
\end{cases} \quad \text{(2)}
\]

Values \( \{w^k_0, w^k_1, w^k_2, w^k_3\} \) are the weights of the co-domain of the function \( \text{pos}(\bullet, \bullet) \): we call them “relative” weights and associate the interest levels to the different HTML sections where a search keyword can be found.

The system author can also assign a weight \( w_k \) to each keyword, on the basis of the keyword relevance for the current search process. We call it an “absolute” weight. Moreover, we denote by \( \text{depth}(p) \) the depth of the
A PRUNING ALGORITHM FOR IMPROVING WEB RESOURCES SEARCHING AND CLASSIFICATION IN KNOWLEDGE MANAGEMENT-BASED WEB SYSTEMS

Finally, we provide the following formal definition of the cost function $f_C(\cdot)$:

$$f_C(p) = \sum_{i=0}^{||p||-1} \left( \log_{\text{post}(p,k)}(w_k) + \frac{w_k}{\text{depth}(p)} \right) \delta(p,k) \quad (3)$$

where $\Gamma$ is the set of keywords.

As shown in the above definition, the cost function $f_C(\cdot)$ is a linear combination of several terms. Note that if the number of search keywords is $W$, the number of terms is $2W$, because for each keyword $k$ we have two kinds of terms. In fact, we intended to capture two properties: the first (corresponding to the term on the left) is a “semantic” property, determined by the location of the keyword inside the HTML page; the second (the term on the right) is a “structural” property, determined by the distance between the current page $p$ and the search tree root. Each term is weighted with the relevance assigned by the system author to each keyword $w_k$. Note that the function $\delta(p,k)$ determines if the keyword $k$ gives a contribution or not, i.e. if the corresponding term of the linear combination is present or not.

With more details, the term $\log_{\text{post}(p,k)}(w_k)$ means that the keyword $k$ gives a contribution that increases with the keyword position $\text{post}(p,k)$. The term $\frac{w_k}{\text{depth}(p)}$ gives a contribution inversely proportional to the depth of the page $p$ that contains the keyword $k$. It means that we attribute a smaller relevance to pages that are more distant by the root of the search tree. Note that the two “effects” are somehow conflicting because, for example, if we consider a keyword $k$ with high values of $w_k$ and $\text{post}(p,k)$ and belonging to a page $p$ very distant from the root, the “reduction” effect induced by the term $\frac{w_k}{\text{depth}(p)}$ is opposed to the effect of the term $w_k \log_{\text{post}(p,k)}(w_k)$ and vice-versa. We have performed experiments that confirm the effectiveness of our cost function $f_C(\cdot)$ in reducing the search space.

Experiments were made as follows: first, we have chosen a set of keywords $G$ and an “absolute” weight for each keyword (see equation (3)); then, we have generated data distributions for mapping “relative” on different HTML sections (see equation (2)); finally, we have derived the number of pages by applying the search algorithm with respect to these data distributions and the max depth value $M$: the “absolute” weights $|G|$, the number of search agents $Q$, and the interesting threshold $V$ were fixed to constant values. We report our results in Figure 2. The depicted plots represent the number of pages found by considering as input three different increasing data distributions $f_j(w_0^k, w_1^k, w_2^k, w_3^k)$ such that $f_3$ produces higher values of the “relative” weights than $f_2$ and $f_2$ produces higher values than $f_1$. Other parameter values are: $|G| = 10$, $Q = 15$, $V = 55$, $M \in \{1, \ldots, 32\}$. As Figure 2 shown, we obtain that the number $N$ of retrieved pages decreases when $M$ increases because our cost function $f_C(\cdot)$ gradually restricts the search space. Moreover, when increasing data distribution values ($f_j$ with $j \in \{1, 2, 3\}$), we obtain higher values of $N$ for fixed values of $M$.

![Figure 2. N/M plots for increasing distributions of “relative” weights of the HTML sections and $|G| = 10$, $Q = 15$, $V = 55$, $M \in \{1, \ldots, 32\}$](image)
4. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a pruning algorithm that is able to successfully obtain reduction of the search space size in Knowledge Management-based Web Systems, i.e. Web Information Systems that adapt contents and presentations with respect to user needs and backgrounds, through the execution of Knowledge Management processes. Experimental results confirm the effectiveness of our algorithm. Future work is focused on the improvement of the proposed architecture and the algorithm by adding from a side more sophisticated techniques for the knowledge representation, as the novel paradigm of ontologies and semantic Web, and from the other side novel techniques of information retrieval and information access.

ACKNOWLEDGEMENT

This work has been partly carried out by a grant within the project Valorizzazione del Patrimonio Archeologico e Monumentale della Regione Calabria – n° 50-ph/232, program POP 94/99 – Misura 4.4. The authors wish to thank Mario Cannataro for his suggestions on an early version of the paper.

REFERENCES