Presenting the Results of *Relevance-Oriented Search* over XML Documents

Alda Lopes Gançarski
LIP6 University Paris 6, 8 Rue Capitaine Scott 75015
Paris, France
Tel. 33144273944
Alda.Lopes@lip6.fr

Pedro Rangel Henriques
University of Minho, Departement of Informatics, 4700
Braga, Portugal
Tel. 351253604469
prh@di.uminho.pt

ABSTRACT
In this paper, we discuss how to present the result of searching elements of any type from XML documents relevant to some information need (*relevance-oriented search*). As the resulting elements can contain each other, we show an intuitive way of organizing the resulting list of elements in several ranked lists at different levels, such that each element is presented only one time. Depending on the size of such ranked lists, its presentation is given by a *structure tree*, for small lists, or by a *sequence of pointers*, for large lists. In both cases, the textual content of the implied elements is given. We also analyse the size of ranked lists in a real collection of XML documents.

Categories and Subject Descriptors
H.2.3 [**Database Management**]: Languages – *Query languages*; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries – *User Issues*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Graphical user interfaces (GUI); Theory and Methods*; I.7.2 [**Document and Text Processing**]: Document Preparation – *Format and Notation*; *Markup Languages*.

General Terms
Languages, Human Factors, Design.

Keywords
User Interface for XML Retrieval.

1. INTRODUCTION
Traditional information retrieval consists of retrieving from a collection the relevant documents to a query expressed in natural language, while returning as few as possible of non-relevant documents. Moreover, the resulting documents should be ranked by their relevance to the query (i.e., by decreasing probability of being relevant). It is common to measure the relevance of a document to a query by the textual similarity between them. In general, the resulting ranked list of documents is presented to the user showing each document at a time, from the begining of the list. The user reads each document until he finds enough relevant information. If, at a certain point, he starts to find most of the documents non-relevant (the quantity is subjective), usually he gives up that list and refines the query to have a new list where he can find more relevant information. This is what happens when using Web search engines, like *Google*.

To take advantage from the structural information, query formats for structured documents retrieval were enriched to accede certain parts of documents based on content restrictions (comparisons against a value) and structural restrictions (referring to elements and their structural relations). The W3C Consortium is currently developing XQuery [1] to become the standard XML query language. XQuery is based on different query languages, such as XQL [8] and XML-QL [2]. These languages can be referred to as data (instead of information) retrieval query languages because the retrieval system searches for parts of documents that *satisfy* the query. Thus, there is no notion of relevance attached to the retrieval task. Recent extensions to these languages allow for textual similarity restrictions (e.g. [4]), yielding to a list of elements ranked by their relevance.

We also proposed an extension to XQL called IXDIRQL [5], for which we developed a system presented last year [6]. This time, we discuss the problem of how to show the results of a *relevance-oriented search* over XML documents, i.e. the results of searching elements of any type about some subject. In this case, a simple ranked list of elements can be difficult to explore because of the hierarchical relations between elements. For example, *ROSearch 'Compilation'* is a IXDIRQL query searching for parts of documents about *Compilation*. Supose that the respective resulting list of elements is: *<section 2, subsection 3.2, section 4, subsection 2.4, ..., subsection 2.1, ...>*. Analyzing this list, the user will get section 2 without knowing the way relevant information is distributed inside. Thus, he will read all its content, including its relevant and non relevant elements. Continuing to analyze the list, after some elements he will get each subsection of section 2, but he already analyzed them. This means that many information in the list is not used, since the user is not expected to read the same element more than one time. We propose to take advantage of relevance measures to allow the user to read as less as possible non relevant elements, thus accessing faster to the relevant information.

In the following sections we will explain how to organize ranked lists (section 2) and how to present them (section 3). We finish the article with a brief conclusion.

DocEng'04, October 28–30, 2004, Milwaukee, Wisconsin, USA.

2. BUILDING THE RANKED LIST

In our system, the ranked list is recursively build based on three requirements: (1) Elements are ranked by decreasing order of relevance. (2) Each non leaf element is included in the list being constructed inside a new element list (EList); the EList of an element is constructed appending the ranked list of its descendants and the element itself. (3) Each element is ranked once and only once. Let us first show how to construct a ranked list by an example and then explain its advantages. Suppose the user makes relevance-oriented search for the fictitious document represented in figure 1 by a tree. In this tree, nodes have element identifiers (from 1 to 16) and respective relevance measures (in [0..1]). Each EList is surrounded by a dashed curve. The first element of the document list is the 13 because is the one with the highest relevance (0.45). This respects point (1) in the requirements above. Among the remaining ones, element 4 is the most relevant one. So, respecting requirement (2), the document list is augmented with the EList of element 4 (note that 4 is a non leaf element). This EList is formed by the descendants list of element 4 (<12, 11>) followed by element 4 itself, becoming <12, 11, 4>. The descendants list of element 4 excludes element 13 because it was already ranked, as required by point (3). In fact, descendants with a higher relevance than the correspondent ascendant element are presented before, so they do not need to be part of the descendants list. At this stage, the list of the example document is <13, <12, 11, 4> >. Element 1 is the next most relevant one and its descendants list (<9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3> >) is constructed using the same criteria. The EList of element 1 (<9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3>, 1>) is obtained appending this element to its descendants list. Inserting the EList of element 1 into the document list in the stage we left it (<13, <12, 11, 4> >), we have the final list for the example document:

<13, <12, 11, 4>, <9, < <15, 14, 16, 5>, 6, 7, 2>, 10, <8, 3>, 1> >.

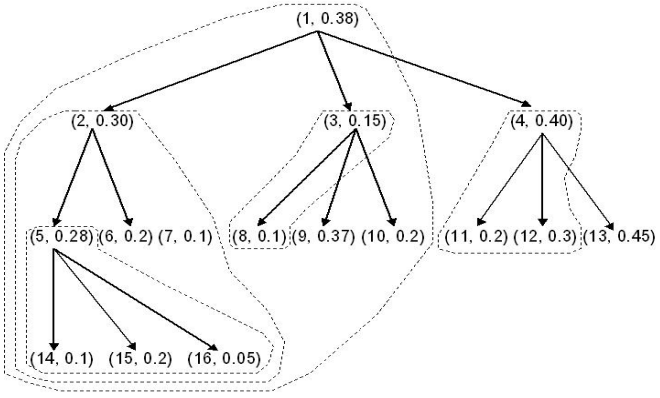

Figure 1 - An exemple of a document tree with element identifiers, relevance measures and ELists.

To better understand the advantages of such a list, we can represent it by a tree where: each node is an element; the root node means the set of elements to judge; ELists are split into the correspondent element (a node) and the descendants list (the sequence of its children). Figure 2 shows the tree representation for the above example list. This tree shows the list composed of several sub-lists at different levels, each one being the descendants list of some element. For example, *<15, 14, 16>* is the sub-list of element 5. The highest level sub-list is a special one belonging to the collection.

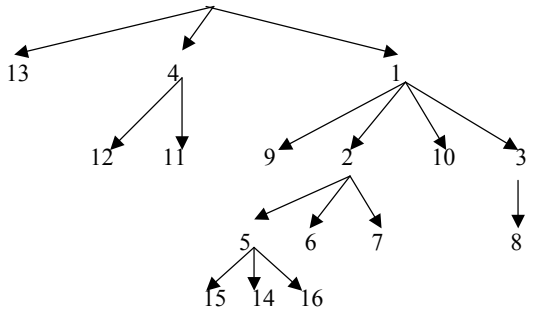

Figure 2 - The tree representation of the ranked list of the document of figure 1.

The tree in figure 2 has the same number of nodes representing elements to read than the document tree (figure 1). Their nodes are also read in a left-to-right, depth-first order. However, here, the algorithm is accelerated because, each time the user is analysing a sub-list, if, at a certain moment, he starts to find many non-relevant elements (as for classical IR, the quantity is subjective), he can decide to give up that sub-list and go up to the following element. This is because the remaining children have less and less probability of being relevant. For example, when the user judges elements 9 and 2 non relevant, he may also assume non relevant the remaining siblings, elements 10 and 3. This way, the user does not read those sibling elements, finishing the analysis of the father (element 1).

3. PRESENTING THE RANKED LIST

To present the ranked list to the user, the size of each sub-list (number of elements) is taken into account. A sub-list with a relatively small size can be efficiently presented in the following way. The structure tree of each element is presented (e.g. the tree in figure 1 for element 1), together with the corresponding textual view. Each node in the tree is linked to the respective textual content in the textual view. A visual variable is attached to the nodes of the tree to represent their relevance. This is similar to what they did in [7], where the visual variable is brightness (black for relevante, white for non-relevante and different shades of gray for intermediate values of relevance). Nodes in the tree that have higher relevance than the present one are not part of the sub-list and were already shown to the user (like node 4 w.r.t. node 1 in figure 1). We suggest that the user may tell the system if an element is relevant or not after analysing it. Thus, nodes corresponding to elements already shown are accompanied by an indication of *relevant* or *non-relevant*, if relevance judgement was given; otherwise, simply *already-shown*. Sub-lists directly descending from the present one are implicitly represented in the structure tree by the visual variable, if they are considered small (e.g. sub-list of element 2 w.r.t. the sub-list of element 1); otherwise, the corresponding node in the structure tree points to another view where the sub-list is shown to the user as follows (in both cases, is the same textual view).

The sub-list corresponding to the collection has no associated structure tree. Also, for large sub-lists, a structure tree with visual variable may be difficult to explore. Imagine, for instance, sub-lists constructed from documents with 4000 elements. In these cases, it is think it is better to proceed as in classical IR: to show one element at a time, starting from the begining of the sub-list. Supposing the sub-list of element 1 of the same example is large, figure 3 shows the respective view. On the top, the sub-list is visualised by the sequence of pointers corresponding to elements

9, 2, 10 and 3. Then, the textual view of element 1 is shown. A pointer of a textual element in the sub-list points to the respective textual content (e.g. elements 9 and 10). A pointer of an element with its own sub-list points to a view where that sub-list is adequately presented depending on its size (e.g. elements 2 and 3), using the same textual view. A visual variable can also be associated to the sequence of pointers to show the relative relevance measures. In the example, elements 4, 11, 12 and 13 are not part of the any descendant sub-list from element 1. To signal that, the corresponding textual content has an indication of *already-read*, *relevant* or *non-relevant* (in the figure, we use dashed horizontal lines).

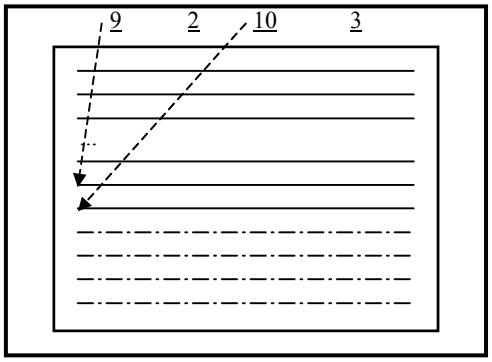

Figure 3 - Representation of a sub-list using lists of pointers.

Finally, to investigate the size of sub-lists, we made an experimental search over 100 documents from the INEX collection. INEX is, since 2002, an anual workshop for the evaluation of XML retrieval [3]. For that evaluation, INEX made available an infrastructure based on a big collection of documents and methods for attribution of scores. The collection is formed by 12107 articles from the *IEEE Computer Society* with 192 different types of elements and variable sizes. In average, an article has 1532 elements and 6.9 levels in the hierarchical structure.

The documents we used were taken in groups of 5 from 20 different directories of INEX. Their number of elements varies from 82 to 4639. Empty elements were excluded from the results, as well as some ones that do not have a meaning by themselves, like bold and italics. The size of the resulting sub-lists was, in average, 3.274414, varying from 2.785714 to 4.122951 per group. The histogram of figure 4 shows more precisely how many sub-lists (yy) had a certain size (xx). Sizes did not exceed 135 elements. If we consider large a sub-list with more than 8 elements, despites less frequent, they are about 10% of the total amount. Thus, the collection sub-list and those 10% of large sub-lists would be presented using lists of pointers, while the remaining ones with structure trees.

4. CONCLUSION

We discussed how to present the result of relevance-oriented search over XML documents. The result ranked list of elements can be presented to the user in two different ways: using the structure tree of elements and a visual variable to represent relevance, when lists are considered small; showing the elements one after the other, when lists are considered large.

As future work, we will, first, create a user-friendly interface to implement these ideas, trying to adequately positionate the views of sub-lists with respect to each other and using suggesting colours to signal different caracteristics. Then, we will analyse the impact of such interface with real users (as in [7]) concerning colours, the relative position of different views, the size of a small and a large sub-list, among other aspects.

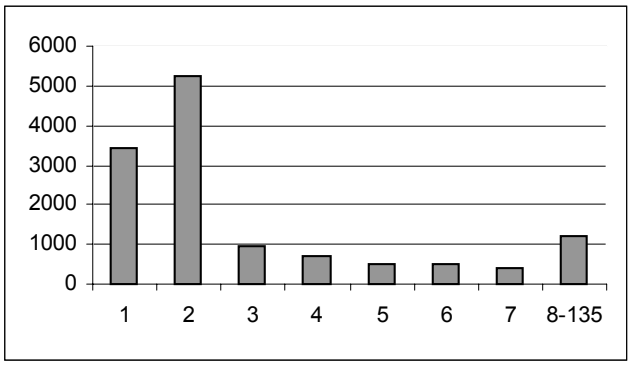

Figure 4 - Histogram of sub-lists size.

5. REFERENCES

[1] Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J., and Siméon, J., XQuery 1.0: An XML Query Language. W3C Working Draft. http://www.w3c.org/TR/xquery/ (November 2003).

[2] Deutsh, A., Fernandez, M., Levy, A., and Suciu, D. XML-QL: A query language for XML. W3C Note. http://www.w3c.org/TR/NOTE-xml-ql/ (August 1998).

[3] Fuhr, N., Govert, N., Kazai, G., and Lalmas, M., INEX: Initiative for the Evaluation of XML Retrieval, in Proceedings of INEX 2002 Workshop, (Schloss Dagstuhl, Germany, December 2002).

[4] Fuhr, N. and Grobjohann, K., XIRQL: An XML query language based on information retrieval concepts. In *ACM SIGIR 2002 Workshop on XML*, (August 2002).

[5] Gançarski, A., and Henriques, P., IXDIRQL: an Interactive XML Data and Information Retrieval Query Language. In Proceedings of ElPub'03, (Guimarães, Portugal, June 2003).

[6] Gançarski, A., and Henriques, P., Interactive Information Retrieval from XML Documents Represented by Attribute Grammars. In Proceedings of ACM DocEng'03 (Grenoble, France, November 2003).

[7] Grobjohann, K., Fuhr, N., Effing, D., and Kriewel, S., A User Interface for XML Document Retrieval. In Proceedings of GI Jahrestagung 2002 (Dortmund, Germany, 2002).

[8] Robie, J., XQL'99 Proposal, http://metalab.unc.edu/xql/xql-proposal-.xml, (1999).