A Model and a Visual Query Language for Structured Text

Ricardo Baeza-Yates
Gonzalo Navarro
Depto. de Ciencias de la Computación,
Universidad de Chile
{rbaeza,gnavarro}@dcc.uchile.cl

Jesús Vegas
Pablo de la Fuente
Depto. de Informática,
Universidad de Valladolid, Spain
{jvegas,pfuente}@infor.uva.es

Abstract

We present a new model to query document databases by content and structure. The main merits of the model are: it allows rich structure in the documents; the query algebra is intuitive (moreover, complemented by a visual query language) and powerful; it is efficiently implementable; it can be built on top of a traditional indexing system or even with no index at all; it is strongly oriented to user-definable relevance ranking instead of boolean logic; and it allows flexible visualization of results in terms of structure, contents and highlighting of user-defined important parts in the query.

1. Introduction

In the past five years the amount of information electronically available has experimented an impressive growth. As more and more textual information becomes available, the need for better search engines gains importance. One of the recent trends is to take advantage of the structure of documents to formulate more precise queries. Structure allows to give some context to the words being searched, and so the semantics of the query is better. Also, many times we remember visual structure that can help to locate a document. This is especially important because standards to represent structured texts are finally consolidating (e.g. SGML [8] and its instance HTML).

A number of models to structure and query textual databases by content and structure already exist. Some work regarding the relationship between the power to structure and query a textual database and the resulting system efficiency has been pursued [3]. However, much less work has been devoted to consider the interest of those powerful query language for the users. In traditional information retrieval systems, however, this issue is a central one, despite the lack of ability to handle structure.

Our main goal is to fill that gap. We define a model to query document databases by content and structure which is intuitive and expressive. The model can be implemented in many ways, answering a reasonable subset of queries in linear time on the size of the intermediate results. Our query language is not based in standard boolean logic but in weights, to help in ranking. All this is complemented with a visual query language.

There are a number of different alternatives to index structured text and answer queries by structure and content. Most of the indices are ad-hoc. We instead propose the use of a traditional inverted index with little enhancements to support queries by structure. These indices have been studied in literature and their behavior is well known [17, 7]. Moreover, they are the most common indices among current text databases. This makes our model simpler to add to an existing installed text database.

The organization of the paper follows. First, we present previous work on the topic, pointing out the differences between a normal IR system and the user–query–answer process in a structure/content environment. Second, we present our model, which in some sense simplifies our previous proposal [14], including the query language and the visual metaphor associated to it. Third, we present the software architecture of a first prototype. Finally, we mention on-going and future work.

2. Previous Work

Traditionally, textual databases have allowed to search their contents (words, phrases, etc.) or their structure (e.g. by navigating through a table of contents), but not both at the same time. Recently, many models have appeared that allow mixing both types of queries.

Mixing contents and structure in queries allows to pose very powerful queries, being much more expressive than each mechanism by itself. By using a query language that integrates both types of queries, the retrieval quality of textual databases can be potentiated.

However, despite all the work done on efficient and/or expressive query languages for structured text [14, 1, 16, 4, 12, 9], the issue of the user's viewpoint has not been studied in detail.

In a classic information retrieval system the query process is well defined. The user makes a query, the system answers it and the user evaluates the results. If the user needs are satisfied, the process is finished. Otherwise, the user can submit another query, filtering the previous answer or starting again (for example, see [5, 6]).

There are some aspects in the IR content based systems that change in the IR structure–content systems. In content-based systems, users do not need to know where the query must be, therefore the structure knowledge is not important. Also, a document is more or less relevant if the query appears more or less times in it (although this depends on the exact ranking algorithm used). Now, when we put structure into the query, the user knowledge on structure becomes important, and the document relevance is related with the place where we can find the information. There is little previous work on relevance ranking for structured text [15].

So, it is important that the user should know the document structure if we want to use it in the query. Therefore, the interface must show the document structure and assist the query process in this way. However, almost all proposals for query languages that include structure [3, 11] are too complex for a final user. That is one of the main reasons because structure is not used. One solution to this problem is a visual query language. A visual language is closer to the user than the traditional ones and makes it easier to express relations between objects. On the other hand, a visual languages limits the expressiveness of the language, because the main goals are simplicity and easiness of use. There are very few languages of this kind [10].

Another problem is how to show the structure in the answer. One helpful solution is to use a visual metaphor to represent every occurrence of the query. If possible, this metaphor should be similar to the one used in the visual query language. This interface should also help the user to rank the retrieved documents and evaluate the precision of the answer [15].

Although there is work on the database community and on commercial systems on these ideas, the approach and the goals of this paper are different. Also, several systems can handle SGML (for example, Search'97 of Verity) but they only provide partial support of content and structural queries (mainly, only structural inclusion).

3. Proposed Model

In traditional IR models, documents are based on elements that reference only their content. In our case we have content elements, c, and structure elements, s. In addition, content elements are not only text-based, as they include any type of data as images or audio. Therefore, we have to define how documents will be described in terms of content and structure.

3.1. Metadata

In traditional IR systems, the structure is very simple (none at all or a sequence of fields) and it is not difficult to specify. Text with rich structure can be specified by ad-hoc techniques or by proposed standards as SGML. We have chosen SGML to define our structure definition language. We call this *metadata*. This information is also useful to:

- guide the user in the querying process suggesting content and structural elements that can be included;
- optimize the queries;
- link the query with the searching engine; and
- verify the syntax of the database and the query.

For each structural or content element of the database we define a metadata element. They are defined using an instance of SGML [8], using a simple format that specifies the name of the element (which will be the tag name for it), its type (using a set of predefined types), a list of elements that can be included inside it, the set of its initial and final tag markers in the text, and a description of the element.

This metadata should describe one hierarchy of information elements (and just one). This is a restriction that we impose, but in practice is very rare to find documents with more than a hierarchy and one hierarchy is simpler for the user to understand.

Also, implicitly, the fact that elements can only be included inside other elements implies that either an element is contained completely in another one or their intersection is empty. Therefore, we use a collection of segments to represent the whole content and structure of the database. Each segment is defined by its initial and final position and segments do not overlap unless one is inside another. We do not make distinctions at the segment level about content or structure, simplifying the indexing and searching process. This plus the fact that we have only one hierarchy, implies that each document can be described by a tree that represents segment inclusion as in [14].

3.2. Query Language

The visual query language must be based on a formal query language. Following the work in [14], we define an intermediate language that will be used between the user interface and the search engine.

The specific language is divided in two parts. At the top level, we have an algebra that returns *ranked documents*. It is formed by a union of one or more *subqueries*. The documents retrieved are those containing segments that are retrieved by at least one non-negative subquery. Formally, we have that a query Q is

$$Q = (s_1 \times q_1, \cdots, s_m \times q_m)$$

where Q is the union of the subqueries q_i. Each subquery q_i can be a *presence* expression or an *inclusion* expression (we later describe these two). The coefficients s_i are the weights of the subqueries. Positive weights increase the relevance of a subquery, while negative weights decrease the relevance and can be considered as the negation operator. The default value for the weights is 1. Later we explain one possible ranking mechanism by using these weights.

The language for the subqueries is an algebra on *segments*. It retrieves a set of segments with an associated weight. The presence operator has the form:

$$(e_1, \cdots, e_m)$$

where each element e_i can be a basic element (either content or structure). This means that at least one of the elements must be present. For example, `(a,b,c)` implies the documents that have `a` or `b` or `c`. If we want to force more elements to be present, we concatenate these expressions. For example, `(a)(b)` means that both `a` and `b` must be present.

The inclusion operator is used to indicate that an expression has to be included in a structure element. It has the following general form:

$$(S(e_1, \cdots, e_m))$$

where S is a structural element, and at least one of the elements e_1 to e_m is contained in it. Again, the e_i can be content or structure elements. This inclusion is transitive (in fact, we cannot query for only direct inclusion). For example, `section(a,b)` returns the documents that have `a` or `b` inside a section. As for presence, may want to force two or more elements to be included. For that, we use the same syntax as before. For example, `section(a)(b)` implies that both `a` and `b` must be included in the section.

This language is an implicit boolean language with inclusion. The weights are used basically for union and subtraction (or and butnot), and concatenation is set intersection (and).

3.3. A Visual Representation

Our proposed metaphor is very simple. The main motivation is that the algebra just defined cannot be easily understood by a normal user, nor even an expert used. What the user usually sees is a page of text. So our visual language will be a page where the structure is composed from a set of predefined objects and the content is written where we want to find it. Each structure element will be a rectangle with its name in the top (using the special name Text if it is a content element). Each content element is placed inside the rectangle. Weights are specified in a scroll bar at the right side of each rectangle. For example, a content element is given in Figure 1 where we use the name "Text" to indicate a content rectangle for the word "hello".

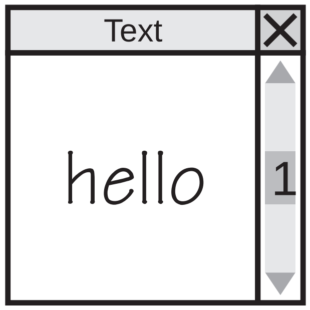

Figure 1. Visual Query for `("hello")`.

Union of queries are obtained by putting rectangles besides each other. Inclusion is obtained by placing rectangles inside rectangles. Figure 2 shows the query `(3` $\times$ `Chapter(a), -c)` where `a` must be inside a chapter and we assign negative relevance to documents having the content element `c`.

The answer is a list of documents, and we can use the same visual metaphor to show how the query matches each document by using its specific content and structure. In this way, the same language is used for both tasks. Nevertheless, one visualization is not enough,

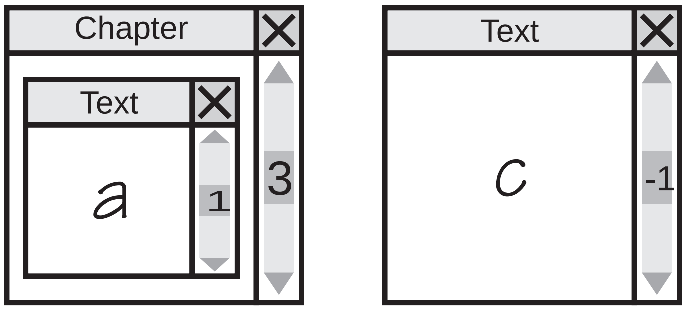

Figure 2. Visual Query for `(3× Chapter(a),-c)`.

and other views for the answer should be provided [2]. An ambitious goal can be to allow the user to edit an answer, such that can be submitted as a new query.

3.4. Ranking

The ranking notion that we have chosen can be formalized in the following way. Let D be the set of documents in the database. To each document $d_j \in D$ we associate a relevance $R(d_j)$ with respect to the query Q. Initially $R_0(d_j) = 0$. Let $\nu(d_j, q_i)$ be the number of times that q_i appears in d_j (given by the search engine), which are always occurrences of content elements (structural elements are not counted).

The final relevance is given by a 2-tuple, the first for the positive weights and the second for the negative weights. That is, $\forall d_j \in D$,

$$\begin{aligned} R(d_j) &= (x, y) \\ x &= \sum_{i,\ s_i>0} s_i \times \nu(d_j, q_i) \\ y &= -\sum_{i,\ s_i<0} s_i \times \nu(d_j, q_i)) \end{aligned}$$

The need for computing separately the positive and negative weights is for ranking purposes as explained next.

An initial algorithm that we are using to rank the documents is as follows:

- Only consider documents with some positive weight (that is, $x > 0$).
- Sort the remaining documents using $x - y$.
- Including or not documents with negative weight might be a user-defined option, because even if the final score is negative, it had some positive weight.

Clearly, other ranking schemes can be devised, but this is a simple one. An evaluation study of this and other strategies is planned.

4. Prototype

We have developed a first prototype (the one shown in the examples), which is based on a three level architecture, like the ANSI relational database standard, to achieve logical and physical independence. The three levels in our case are:

- external: user interface for querying and viewing the results.
- conceptual: receives the query in some language and establishes the search strategy (optimization). When the search engines return the answers, these are combined, filtered and weighted before passing it to the external level to be visualized.
- internal: several search engines, each one can search about one different kind of data: structure, some kind of content (text, images, sound, video), etc. The elements of this level communicate the search results to the conceptual level using the segment model (with a specific format).

All these levels share the data definition (metadata), where the structure and contents allowed in the documents are defined. The complete architecture is shown in Figure 3.

4.1. The External Level

The external level is where the queries are formed and the results are visualized. The objectives of this level are:

- To present to the user an interface that allows to make queries and to view the results.
- To obtain a representation of the query in an algebra expression that will be solved by conceptual level.
- To show the results delivered by the conceptual level using the different visualizations associated to the answer, as well as correctly displaying the different media present in contents.

4.2. The Conceptual Level

This level is responsible for the implementation of the language. It receives from the external level a query

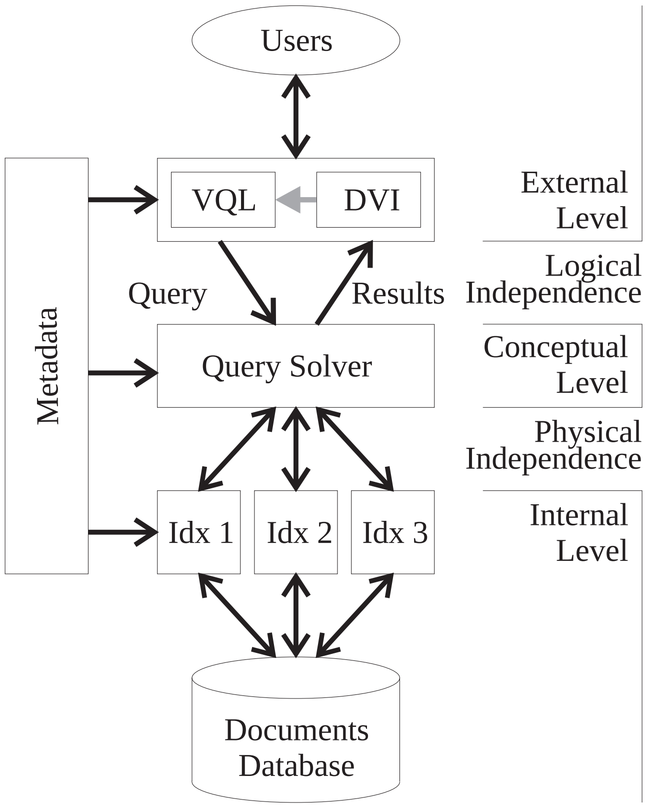

Figure 3. System Architecture.

expressed in the algebra and delivers back the set of documents that match the query, together with their ranks and segments to highlight.

It also interacts with the internal level, from where it retrieves the set of segments corresponding to basic constructor names and queries on content.

The query syntax tree is solved in four steps

Step 1 - Query optimization: The query is optimized and an access plan is devised.

Step 2 - Obtaining candidate documents: The set of documents that match the query is computed.

Step 3 - Ranking: For each matching document, its rank is computed.

Step 4 - Highlighting: For each document to be visualized, its structure and areas to highlight are obtained.

In order to give the user early feedback on the result of the query, Step 2 is not monolithic. Instead, the documents can be obtained one at a time. Moreover, Step 3 can be performed at any time and in any order of document. This allows the external level to implement a smooth displaying of results that allows the user to perform an early evaluation of his query.

In Step 1, the first optimization which is performed is to convert the query tree into a DAG (directed acyclic graph), factoring common subexpressions. This is especially important because the external level expands conditions which are preferential into two expressions (with and without the condition) that share possibly complex operands.

Some simplifications are performed on the expression used in Step 2, which are not valid for Step 3. For instance, all the information on preferences is discarded, as well as negated operations (since they will not affect the result set, only its rank). This may also increment the degree of operand sharing in the DAG (i.e. two operands may be equal except for preferences).

Further optimizations include exchanging the order of some operations, for instance the one which corresponds to the identity $A(B)(C) = A(C)(B)$. It is more convenient to solve first the smaller between B and C, since hopefully this will leave less segments to operate against the other one. This cannot be done beforehand, since it is necessary to obtain B and C before deciding the best execution order.

The operations on segments can be performed in full or lazy form. In the first form, the arguments of an operation are completely computed before starting to solve the operation. In the second form, we ask a first result to the root of the expression tree. The root, in order to deliver a first result, asks more results to its children, and so on. The advantage of lazy evaluation is that some parts of the results are never computed, because it is possible to determine beforehand that they will not be needed.

Step 2 obtains the documents which match the query, with no weight information. This requires much less work than obtaining the full data. In Step 3 we do not search for matching documents, but given a document that matches the query we evaluate its weight. Similarly, in Step 4 we obtain the interesting structural components and the list of segments that must be highlighted. Therefore, we work on Step 3 only on matching documents, which avoids the heavy work on determining highlighting on a document that will not classify. In the same spirit, we work on Step 4 (which is even harder) only on documents that the user really wants to display.

Steps 2, 3 and 4 may look similar (since they use the same query) but they are very different. First, as explained, more optimization can be performed for Step 2. Second, Step 2 needs to locate matching documents without traversing the whole database, while Steps 3 and 4 do not need to locate the documents. Third, Steps 2 and 3 do not need the exact positions (in characters) of the segments associated to words and structure elements. Any numbering scheme (such as the word-count instead of character-count) would work as long as the left-to-right relationships are preserved. On the other hand, Step 4 needs exact positions to determine what to highlight. This makes Steps 2 and 3 very suitable to be solved with the use of a classical inverted index, while Step 4 is more likely to be solved by an on-line processing on the matched document.

With regard to costs, Step 2 can be implemented in linear time with respect to the total size of the intermediate results (see [14], where Step 2 is the main operation). Steps 3 and 4 are applied only to individual documents (which are to be ranked or displayed, respectively), and are implemented in time proportional to the total size of the intermediate results of that document only.

4.3. The Internal Level

This level is responsible for retrieving the areas corresponding to each structure element and to queries on content. Since there may be different media in the content of a document (text, audio, etc.), we can have a separate index for each content medium.

It is also possible to have a separate specialized index for the structure. However, we use a different approach here, which makes this model very easy to integrate with already installed information retrieval systems. The idea is to solve queries on structure by using the index on textual content. If a language has markup information to describe the structure (such as SGML), we only need that the tags that mark the beginning and end of structures are considered as words in their own and indexed as any other word. That is, we need that the index be able to efficiently retrieve all the positions where the beginning or end tags are present in the text. It is not hard to deduce the hierarchy in linear time given the information on positions (moreover, this is the same information stored in an ad-hoc index proposed in [14]). This requirement is not very hard to accomplish given an already present inverted index for the textual content.

It is also possible that there is no explicit markup information, but it is derived, by a parsing process on the text. In that case, we can modify the filters of the information retrieval system. All those systems access the text via an interface that eliminates format information (for example, to seamlessly index documents generated with different word processors), maps characters, eliminate stopwords, replace synonyms, etc. We should make that filter to generate bogus tags whenever it decides that a structural element has begun or ended.

The idea of using the content index to answer queries on structure has been previously used in another model [16]. However, they do not allow nesting in the structures, while we allow it with no additional overhead. Moreover, since the most efficient way to obtain the structure given the set of tags positions is to process them in ascending order, an inverted list is more suitable than the suffix arrays [13] used in that model, which deliver the positions in suffix order rather than in positional order.

Finally, it is worth to mention that the model can be implemented with no index at all, if we can pay for the time overhead (that is, an additional search time which is proportional to the text size). Moreover, some steps of the search are better performed on-line.

5. Conclusions and Future Work

We are currently testing and extending the prototype of the model based in the software architecture described here. After that, we plan to do an usability test of our visual language. Also, ranking can always be improved. Further study on how our proposal can help user guided and automatic ranking is needed.

Another future work should compare the expressiveness of the proposed model with previous proposals as in [3]. In particular, we are limiting the structure to only one hierarchy of non-overlapped elements. Nevertheless, most documents only have one, namely the logical structure, which usually does not overlap (e.g. SGML like). Our language also does not allow direct relations of inclusion, that is, to find parent-child relationships on the hierarchical tree. All inclusion relations are transitive. In most cases, this limitation is not important. On the other hand, languages without this restriction are more complex, need special indices and not always are efficient [3].

References

[1] R. Baeza-Yates. An hybrid query model for full text retrieval systems. Technical Report DCC-1994-2, Dept. of Computer Science, Univ. of Chile, 1994. `ftp://sunsite.dcc.uchile.cl/pub/users/-rbaeza/hybridmodel.ps.gz`.

[2] R. Baeza-Yates. Visualizing large answers in text databases. In *Int. Workshop on Advanced User Interfaces (AVI'96)*, pages 101–107, Gubbio, Italy, May 1996. ACM Press.

[3] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *ACM SIGMOD Record*, 25(1):67–79, Mar. 1996.

[4] C. Clarke, G. Cormack, and F. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 1995.

[5] W. Croft, R. Krovetz, and H. Turtle. Interactive retrieval of complex documents. *Information Processing and Management*, 26(5):593–613, 1990.

[6] W. Croft, L. Smith, and H. Turtle. A loosely coupled integration of a text retrieval system and an object-oriented database system. In *Proceedings of SIGIR 92*, pages 223–232, 1992.

[7] W. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.

[8] International Standards Organization. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, 1986. ISO 8879-1986.

[9] P. Kilpeläinen and H. Mannila. Retrieval from hierarchical texts by partial patterns. In *Proc. ACM SIGIR'93*, pages 214–222, 1993.

[10] E. Kuikka and A. Salminen. Two-dimensional filters for structured text. *Information Processing and Management*, 1995.

[11] A. Loeffen. Text databases: A survey of text models and systems. *ACM SIGMOD Conference. ACM SIGMOD RECORD*, 23(1):97–106, Mar. 1994.

[12] I. MacLeod. A query language for retrieving information from hierarchic text structures. *The Computer Journal*, 34(3):254–264, 1991.

[13] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. In *Proc. ACM-SIAM'90*, pages 319–327, 1990.

[14] G. Navarro and R. Baeza-Yates. Proximal Nodes: a model to query document databases by content and structure. *ACM TOIS*, 15(4):401–435, Oct 1997.

[15] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel. Database systems for structured documents. In *Proc. ADTI'94*, pages 272–283, 1994.

[16] A. Salminen and F. Tompa. PAT expressions: an algebra for text search. In *COMPLEX'92*, pages 309–332, 1992.

[17] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.