# Garnata: An Information Retrieval System for Structured Documents based on Probabilistic Graphical Models

**Luis M. de Campos, Juan M. Fernández-Luna, Juan F. Huete, Alfonso E. Romero**
Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S.I. Informática, Universidad de Granada, 18071 Granada, Spain.
{lci, jmfluna, jhg, aeromero}@decsai.ugr.es

**Abstract**

In this paper, Garnata, an information retrieval system for XML documents is presented. This system is specifically designed for implementing Bayesian network-based models for structured documents. We show its architecture and performance from the indexing and the retrieval points of view, coming to the conclusion that the system is flexible and fast.

**Keywords:** Information Retrieval Systems, Structured documents, Bayesian networks

## 1 Introduction

In the beginning, Information Retrieval (IR) only dealt with documents whose internal organization was not taken into account. Thus, they were considered as atomic units, and therefore, a query result was a set of those documents sorted by their relevance status value (RSV). Nowadays, due to the great acceptance of markup languages as SGML and XML, the field of IR has evolved developing new techniques to work with structured documents [2]. This research has shown two important points: the internal organization of the documents is relevant for retrieval purposes (determining the appropriate level of component granularity to be returned) and the structure can be useful to compute the RSV of different parts of the document, making the ranking more accurate.

This evolution implies that new models have been developed from the 'classic' IR to this new 'structured' field, but the scope of these new models is not limited to retrieval: indexing is also a challenging task. So, new Information Retrieval Systems (IRS) have been built in order to implement these theoretical models. Here, our aim is double: on the one hand, to build an experimental software being able to test the effectiveness and efficiency of the models (as it is done in the *INitiative for the Evaluation of XML Retrieval* [6], a forum to compare structured IRSs under the same conditions), and, on the other hand, to have an operational environment where the user can interact with them.

An example of this evolution is the *Bayesian Network Retrieval Model* [3], which was initially designed to work with flat documents, and later extended to deal with structured documents. At present, our current model, is the *Context-based Influence Diagram for Structured Documents* [4, 5], a Probabilistic Graphical Model (PGM) that takes into account the context of the components in the documents, as well as user preferences.

There are available lots of IRSs, most of them for flat document retrieval (really few of them for structured documents), but these are too general for the specific requirements that PGMs need in order to perform efficient and effective retrieval. The cost of updating or extending them to support our PGM-based IR models is much greater than the cost of building our own IRS from scratch, totally adapted to our models. Therefore, the *Gar-*

*nata* IRS has been specifically designed to implement models based on PGMs. In its development, the underlying model to deal with structured text was carefully studied, extracting the most critical operations in terms of efficiency. Having them in mind, the system has been implemented with the aim of optimizing them, using an efficient combination of data structures that assure a good response time for a query. Although at its present form the system only can answer 'content' queries, it is completely ready to implement 'content and structure' queries using the same data structures. It is basically an experimental platform, where different PGMs can be implemented and tested using standard test collections, but also used in operational environments without too much effort.

The aim of this paper is to describe the architecture of Garnata, studying the system performance but not the model effectiveness. In order to expose our ideas, this paper is divided into the following sections: the first one will briefly show the context of our work. Section 3 will describe the Bayesian network model for structured documents implemented by Garnata. Section 4 presents the architecture and the main features of this software. With the objective of testing its performance in terms of indexing and retrieval time, in Section 5 some experiments are presented, concluding in Section 6 with some remarks and future research.

## 2 Related Work

There are several approaches for tackling the problem of indexing and retrieval of structured documents, mainly marked up in XML. One of these is based on storing documents in tables of relational databases and using the tools given by database management systems to retrieve documents, for instance, the SQL language. Examples of this approach are *Agora* [11] or *NATIX* [9]. A group composed of classic IR techniques is the other main perspective to the indexing and retrieval of structured documents. Basically, they store the tree representing the document collection, as for example [10], or use the classic approach of inverted file containing the occurrences of the terms in XML tags, and any kind of data structure to represent the internal organization (hierarchy) of the documents [13]. This is the philosophy that Garnata implements, differing the data structured used. In [8] the reader can find a detailed revision of the state-of-the-art in XML indexing and retrieval.

## 3 The Bayesian Network Model for Structured Documents

A collection of structured documents will be composed of $M$ *documents*, $\mathcal{D} = \{D_1, \ldots, D_M\}$ and the set of the *terms* used to index these documents. Each document $D_i$ is organized hierarchically, representing structural associations of elements in $D_i$, which will be called *structural units*. Each one is composed of other smaller structural units, except some 'terminal' or 'final' units which are indivisible, but composed of terms. Conversely, each structural unit, except the one corresponding to the complete document, is included in only one structural unit.

The Bayesian network modeling a collection of structured documents will contain two kinds of nodes, representing the terms and the structural units. The former will be represented by the set $\mathcal{T} = \{T_1, T_2, \ldots, T_l\}$. There are two types of structural units: *basic* or *final structural units*, containing only terms, and *complex structural units*, composed of other final or complex units, $\mathcal{U}_b = \{B_1, B_2, \ldots, B_m\}$ and $\mathcal{U}_c = \{S_1, S_2, \ldots, S_n\}$, respectively. Therefore, the set of all structural units is $\mathcal{U} = \mathcal{U}_b \cup \mathcal{U}_c$. Each node $T$, $B$, $S$ or generally $U$ has associated a binary random variable, which can take its values from the sets $\{t^-, t^+\}$, $\{b^-, b^+\}$, $\{s^-, s^+\}$ or $\{u^-, u^+\}$ (the term/unit is not relevant or is relevant), respectively. A unit is relevant for a given query if it satisfies the user's information need expressed by this query. A term is relevant in the sense that the user believes that it will appear in relevant units/documents.

There is an arc from a given node (either term or structural unit) to the particular structural

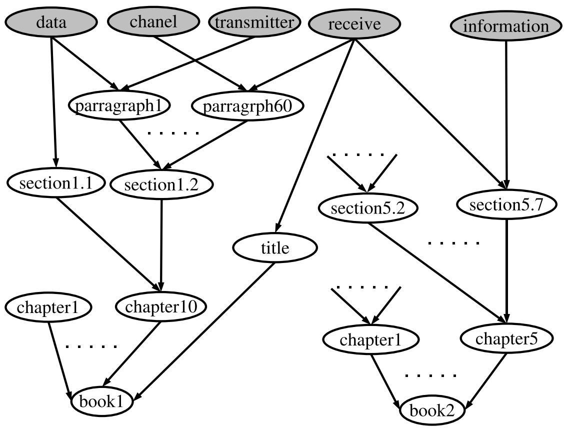

Figure 1: Bayesian network representing a structured document collection.

unit node it belongs to. It should be noticed that the hierarchical structure of the model determines that each $U \in \mathcal{U}$ has only one structural unit as its child –the unique structural unit containing $U$– (except for the leaf nodes, i.e. the complete documents, which have no child). We shall denote indistinctly by $Hi(U)$ or $U_{hi(U)}$ the single child node associated with node $U$ (with $Hi(U) = null$ if $U$ is a leaf node). Figure 1 displays an example of the proposed network topology.

The conditional probabilities have also to be assessed: $p(t^+)$, $p(b^+|pa(B))$, $p(s^+|pa(S))$, for every node in $\mathcal{T}$, $\mathcal{U}_b$ and $\mathcal{U}_c$, respectively, and every configuration of the corresponding parent sets ($pa(X)$). The conditional probabilities for the basic and complex structural units are defined as follows:

$$\forall B \in \mathcal{U}_b,\; p(b^+|pa(B)) = \sum_{T \in R(pa(B))} w(T,B)\,, \tag{1}$$

$$\forall S \in \mathcal{U}_c,\; p(s^+|pa(S)) = \sum_{U \in R(pa(S))} w(U,S)\,. \tag{2}$$

where $w(T,B)$ is a weight associated to each term $T$ belonging to the basic unit $B$ and $w(U,S)$ is a weight measuring the importance of the unit $U$ within $S$. In any case $R(pa(U))$ is the subset of parents of $U$ relevant in the configuration $pa(U)$, i.e., $R(pa(B)) = \{T \in Pa(B)\,|\,t^+ \in pa(B)\}$ and $R(pa(S)) = \{U \in Pa(S)\,|\,u^+ \in pa(S)\}$. These weights can be defined in any way, constrained to $w(T,B) \geq 0$, $w(U,S) \geq 0$, $\sum_{T \in Pa(B)} w(T,B) \leq 1$, and $\sum_{U \in Pa(S)} w(U,S) \leq 1$. For example, they can be defined using a normalized tf-idf scheme (that is, they are divided by the sum of all weights in the final unit), as in [4]. With respect to the prior probabilities of relevance of the terms, $p(t^+)$, they can also be defined in any reasonable way, for example an identical probability for all the terms, $p(t^+) = p_0$, $\forall T \in \mathcal{T}$, as proposed in [4], although any other approach could be used.

To provide the user an ordered list of structural units given a query $Q$, expressed as a list of index terms, and taking into account the specific characteristics of the canonical model used to define the conditional probabilities, the posterior probabilities of relevance $p(b^+|q), \forall B \in \mathcal{U}_b$ and $p(s^+|q), \forall S \in \mathcal{U}_c$, are efficiently computed using eqs. (3) and (4):

$$\begin{aligned} p(b^+|q) = {} & \textstyle\sum_{T \in Pa(B) \setminus Q} w(T,B)\, p(t^+) + \\ & \textstyle\sum_{T \in Pa(B) \cap R(q)} w(T,B)\,. \end{aligned} \tag{3}$$

$$p(s^+|q) = \textstyle\sum_{U \in Pa(S)} w(U,S)\, p(u^+|q)\,. \tag{4}$$

where $R(q)$ denotes the set of terms in the query $Q$ and $p(u^+|q)$ is the posterior probability of any unit parent of $S$. Figure 2 shows an algorithm that efficiently computes these data, derived from eqs. (3) and (4) (see [5] for a more detailed explanation), traversing only the nodes in the graph that will require updating. It is assumed that the prior probabilities of all the nodes are stored in prior[X]; the algorithm uses variables prob[U] which, at the end of the process, will store the corresponding posterior probabilities. Essentially, the algorithm starts from the items in $\mathcal{Q}$ and carries out a width graph traversal until it reaches the basic units that require updating, computing $p(b^+|q)$. Then, starting from these modified basic units, it carries out a depth graph traversal to compute $p(s^+|q)$, only for those complex units that require updating.

The algorithm that initializes the process by computing the prior probabilities prior[U] (as the items $T \in \mathcal{T}$ are root nodes, the prior probabilities prior[T] do not need to be calculated, they are stored directly in the structure) is quite similar to the previous one, but it needs to traverse the graph starting from all the items in $\mathcal{T}$.

```
for each term T in Q
    for each unit B child of T
      if (prob[B] exists)
        prob[B] += w(T,B)*(1-prior[T]);
      else { create prob[B];
             prob[B] = prior[B]+w(T,B)*
                       (1-prior[T]); }
for each basic unit B s.t. prob[B] exists {
  U = B; prod = prob[B]–prior[B];
  while (Hi(U) is not NULL) {
    S = Hi(U);
    prod *= w(U,S);
    if (prob[S] exists)
      prob[S] += prod;
    else { create prob[S];
           prob[S] = prior[S]+prod; }
    U = S; }
}
```

*Figure 2: Computing $p(b^+|q)$ and $p(s^+|q)$.*

### 3.1 Virtual nodes

At the beginning of this section, we mentioned that each structural unit is composed of other smaller structural units, except for the 'final' units, which do not contain any other unit but text. However, it is quite common to find collections where a unit can contain text and other units at the same time; e.g. a paragraph which includes a bold-faced sentence:

```
<paragraph> This is an example of a
paragraph with a <bold> bold-faced
text <\bold>. <\paragraph>
```

To deal with this situation, and in order to continue having a clear distinction between complex and final units, so the propagation algorithm could work correctly, the model includes some special nodes, called *virtual units*, which are fictitious nodes that are parents of the unit where this situation has been detected (in the previous example, the 'paragraph' unit) together with the units contained in it (in this case, only 'bold'). Therefore, these virtual units will be final units containing only terms. Figure 3 shows this situation.

These virtual units cannot be retrieved and their corresponding weights are computed according to the general weighting scheme used, taking into account their parent terms.

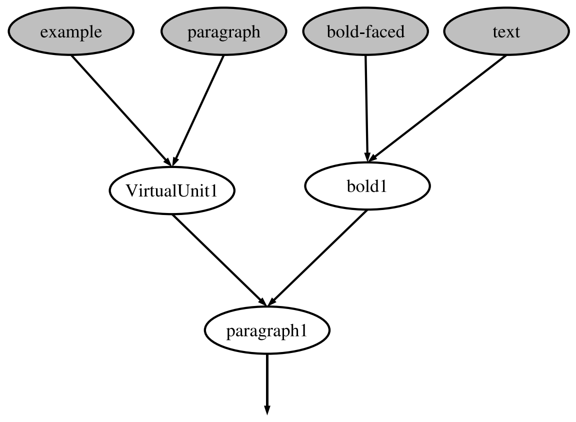

Figure 3: Fragment of Bayesian network containing a virtual unit.

### 3.2 Some important requirements

From the algorithm in Figure 2, we can detect some operations that any IRS implementing this model must perform efficiently: 1) given a unit, obtaining its child (the unit that contains it); 2) given a unit and its child unit, obtaining the corresponding weight; 3) given a term, obtaining the list of units where it occurs; and 4) given a term and a unit where it occurs, obtaining the corresponding weight.

Beside these operations, the IRS has also to allow a fast access to the data used by the algorithm to compute the posterior probabilities of the corresponding units (for instance, prior probabilities and intermediate storage of posterior probabilities).

## 4 The Garnata Information Retrieval System for XML documents

Garnata was born as an implementation completely adapted to the models based on PGMs to retrieve structured documents, although other models following the same philosophy could be easily implemented in it. Written in C++, following the object-oriented paradigm, it offers a wide range of classes and a complete set of utility programs. In its first stage it implements the model described in section 3, but it is ready to implement the CID model [5].

In the following sections we shall study its architecture from the indexing and querying points of view, presenting its main features.

## 4.1 Indexing Subsystem

### 4.1.1 Application Level

Garnata is able to manage different collections, and different indexes over the same collection. Thus, a program (`makeIndex`) is provided to do this task. It can choose among different stopword lists (previously inserted into the system) and use (if desired) Porter's stemming algorithm. Also, it is able to detect those situations explained in section 3.1 and create the corresponding virtual units.

Moreover, as stated by Baeza and Ribeiro [1], an IR system is characterized by its *weighting function*, $w_{ij}$, which assigns a weight to the term $i$ in the document $j$. In our model, several valid weighting schemes could exist because of its experimental nature. As a consequence, in Garnata, indexing does not compute the weights (setting all of them to be zero). Instead of that, we have added the possibility to calculate weights (following a certain weighting scheme) for previously built indexes without inserting into them, and store them in files –the so-called *weight files*– . So, records of that precomputed weight files are kept in order to provide a fast way to insert one into the index itself in order to retrieve with it. To achieve these two tasks two separated executables (`makeWeightFile` and `insertWeightFile`) are provided.

### 4.1.2 Physical Level and Data Structures

Because indexes are only built few times, there is no need for using a very fast indexing algorithm. In Garnata, it is emphasized querying time over indexing time, even storing some redundant information. Nevertheless, we will also describe the structures associated to indexing.

To store textual information (terms and identifiers of the final units where they appear), we use inverted indexes [12]. While the lexicon is kept entirely in memory (both while indexing and querying), the list of occurrences is read from disk. We use another file to write the list of relative positions of each term inside a unit in order to answer queries containing proximity operators or phrases (although in the current stage of Garnata, they are not used to formulate a query).

To maintain information about the structural units, we use one direct access file, except for the XPath routes, which are stored separately. Other files keep relations among units, being accessible with two disk reads only[1]. So a large file contains data of each unit itself (identifier, tag, container, position, ...) and besides, we can easily manage the following relationships with two disk accesses (see Figure 4 for more details):

- Given a non-final unit, returning the list of identifiers of the units that it contains. For unit 1 in Figure 4, it returns $(2, 3, 4)$.
- Given a final unit, returning the container unit and, recursively, all the containers until a root unit is found. For unit 5 in Figure 4, it returns $(2, 1)$.
- Given a final unit, returning the list of contained terms (known as the *direct index*). For unit 5 in figure 4, it returns $(1, 2)$.

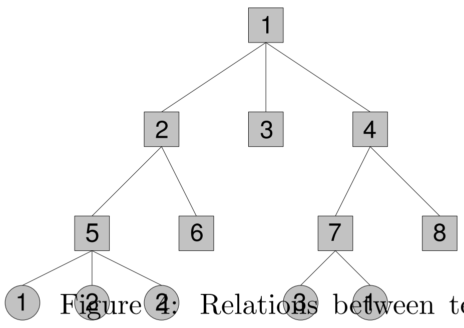

Figure 4: Relations between terms (circles) and units (squares).

Although the indexing subsystem of Garnata is not aimed to any particular IR model, all

[1]An access to this information requires two disk reads: the first in the direct access file (to retrieve the address) and the second in the proper data file (using retrieved address). This approach is similar to how a pointer works in programming languages.

of the previous operations are specifically designed for the BNR-SD model and its derivatives (SID and CID models [4]). Nowadays, this subsystem does not implement any kind of file compression.

### 4.2 Querying Subsystem

Querying subsystem is the most critical part of an IR system. In our case, we have built structures at indexing time to reduce at maximum the amount of disk accesses while processing a query, in order to save time and give a short response time. The algorithm for achieving this task comprises the following steps (not necessarily in this order):

1. Query is parsed, and occurrences of the component terms are retrieved from disk.
2. For each occurrence, implied final units are read into memory (if not already there).
3. For each final unit, its descendants are read into memory (if not already there).
4. Propagation is carried out, units are sorted by its probability of relevance, and the result is returned.

The first big bottleneck to be minimized is due to the reading from disk of the unit objects (containing information about each unit). We will keep two unit caches in memory: the first one, containing final units, and the second one, containing complex units. Both will be *static caches*, meaning that they will not change the unit stored in each cache slot. Cache is accessed doing a hash function-like scheme[2], so for each cache slot, we will have several candidates (those identifiers being the hash inverse of the slot identifier).

For the final units cache, in each slot, we will store the unit containing greater number of terms (among the candidates). For the complex units cache, in each slot, we will store the unit containing more final units. These two heuristics has shown very good performance in our experiments.

[2]Identifier *mod* $N$, with $N$ the size of the cache.

## 5 Experimentation

The experimentation presented in this paper tries to determine the efficiency of Garnata in the tasks of indexing and retrieving, in terms of disk space required by the indexes (uncompressed), and time to index the collection and to retrieve, following an approach similar to [7], although not comparable because of the use of different collections. In this paper we are not concerned about the effectiveness of the model, fact that will be treated in a further study.

Two XML collections have been used to test the system : Shakespeare[3], which is composed of Shakespeare's plays, and INEX [6], composed of articles from journals and proceedings of the IEEE Computer Society about computer science. We have used version 1.4 (from 1995 until 2001).

In order to test how the indexing time and index size increases when the size of the collection also increases, we have obtained several sub-collections from INEX 1.4 of different sizes, as observed in Table 1, where we also show the size of the original XML collection in Megabytes, the number of total structural units, and the number of final units of each collection. The partitions created in INEX 1.4 correspond to the closest values to 5%, 20%, 50% and 75% of the total size, without splitting journals, i.e., integrating complete journals in a partition.

| Collection | Size | Units | Final Units |
|---|---|---|---|
| Shakespeare | 7 MB | 179,689 | 179,626 |
| INEX (6%) | 36 MB | 412,596 | 326,326 |
| INEX (15%) | 81 MB | 886,270 | 711,650 |
| INEX (40%) | 214 MB | 2,500,624 | 1,994,248 |
| INEX (78%) | 418 MB | 6,280,164 | 5,044,672 |
| INEX (100%) | 536 MB | 8,239,997 | 6,600,015 |

Table 1: Information about test collections.

In Table 2, we show the size of the indexes created by Garnata and the indexing times (in minutes), and in Figures 5 and 6 the graphs representing the original sizes of the collections vs the sizes of the indexes and the indexing times, respectively. These experi-

[3]http://qmir.dcs.qmw.ac.uk/Focus/collbuilding.htm

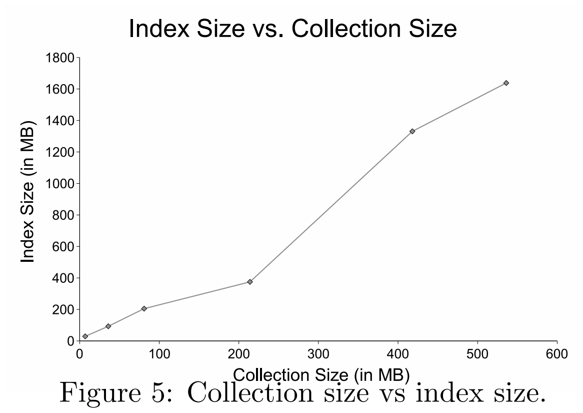

Figure 5: Collection size vs index size.

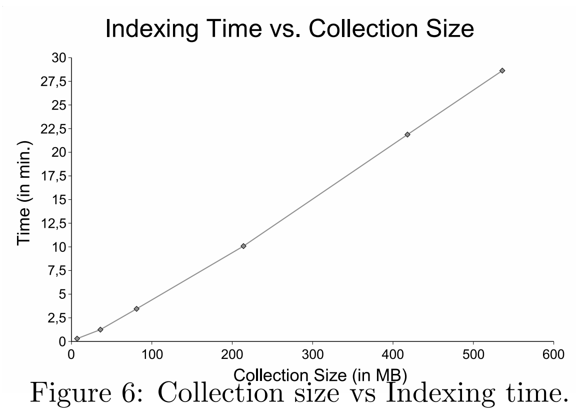

Figure 6: Collection size vs Indexing time.

ments have been carried out in a Pentium IV 2.8GHz, and 3GB of RAM.

| Collection | Index Size | Index Time |
|---|---|---|
| Shakespeare | 29 MB | 0.3 min. |
| INEX (6%) | 93 MB | 1.25 min. |
| INEX (15%) | 205 MB | 3.45 min. |
| INEX (40%) | 375 MB | 10.08 min. |
| INEX (78%) | 1,331 MB | 21.87 min. |
| INEX (100%) | 1,638 MB | 28.63 min. |

Table 2: Index sizes and indexing times.

In order to measure the retrieval time, we have ran a set of standard queries in each collection, normally used for evaluation purposes. In Shakespeare, this set is composed of 43 queries, while in INEX, we have selected the set of content queries used in 2004, containing 74 queries. Table 3 shows the average retrieval time in seconds and the standard deviation, for each considered collection. Figure 7 displays the original size of the collection vs. average retrieval time.

| Collection | Mean | Deviation |
|---|---|---|
| Shakespeare | 0.05 s. | 0.22 s. |
| INEX (6%) | 0.05 s. | 0.22 s. |
| INEX (15%) | 0.12 s. | 0.33 s. |
| INEX (40%) | 0.51 s. | 0.78 s. |
| INEX (78%) | 0.93 s. | 0.52 s. |
| INEX (100%) | 1.39 s. | 0.74 s. |

Table 3: Retrieval times: averages and standard deviations.

Observing the results of the experiments, it is easy to notice how the size of the indexes is around three times the size of the set of XML files composing each collection. So, the disk

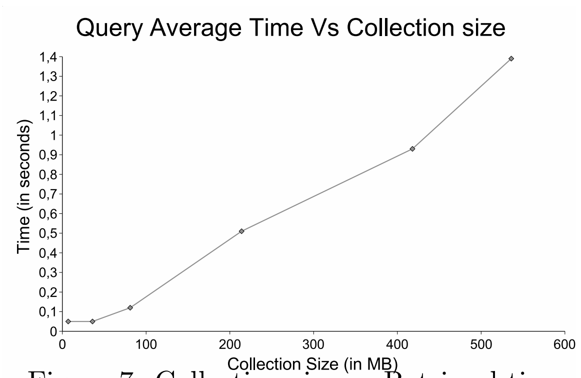

Figure 7: Collection size vs Retrieval time.

space occupied for these and other similar experimental datasets is reasonably affordable. Taking into account the size of the current hard disks, even for really very large collections this is not a serious problem.

We can also conclude that indexing time is quite good, although possibly it could be improved. As it is proportional to the amount of information to be written in disk, its corresponding ratio is quite similar to the original collection size vs. index size ratio. Having in mind that indexing is a task that it is not very often performed and that this system is mainly a testing environment, these times are clearly acceptable, specially if we assume that INEX is considered as a relatively large structured collection.

With respect to the retrieval time, the main conclusion that we can draw is that the response time is quite acceptable for an experimental system, although for an operational system it should be reduced.

## 6 Conclusions and Future Research

In this paper, Garnata has been presented. This is an information retrieval system specifically designed to implement PGMs to retrieve XML documents, although it could be easily extended to deal with passage retrieval. We have described the initial underlying retrieval model as well as its architecture. The experiments about indexing time and space, and retrieval time show an acceptable performance, useful for environments mainly experimental, but also for operational ones.

In order to reduce the size of the indexes, we are planning to implement an efficient method to perform compression and studying, on the one hand, the decrease of size of the indexes, and on the other hand, the expected increase of the indexing and retrieval times. Only empirically we can determine how much the performance of the system could change and make the decision of using compression or not. Clearly, we are in favour of a faster search, although more disk space is needed.

We also expect to measure the system effectiveness evaluating the output with the relevance judgments provided by INEX. In this line, the CID model will be implemented and its retrieval behaviour checked.

**Acknowledgements**

This work has been jointly supported by the Spanish Ministerio de Educación y Ciencia and Junta de Andalucía, under Projects TIN2005-02516 and TIC276, respectively.

## References

[1] R. Baeza-Yates, B. Ribeiro-Neto. *Modern Information Retrieval*, Addison Wesley Longman, 2002.

[2] Y. Chiaramella. Information retrieval and structured documents. *LNCS*, 1980:291–314, 2001

[3] L.M. de Campos, J.M Fernández-Luna, and J.F. Huete. The BNR model: foundations and performance of a Bayesian network-based retrieval model. *IJAR*, 34:265–285, 2003.

[4] L. M. de Campos, J. M. Fernández-Luna, J. F. Huete. Using context information in structured document retrieval: an approach based on influence diagrams. *IP&M*, 40(5): 829–847, 2004.

[5] L.M. de Campos, J.M Fernández-Luna, and J.F. Huete. Improving the Context-based Influence Diagram Model for structured document retrieval: removing topological restrictions and adding new evaluation methods. *LNCS*, 3408:215–229, 2005.

[6] http://inex.is.informatik.uni-duisburg.de

[7] N. Fuhr, N. Govert. Index Compression vs. retrieval time of inverted files for XML documents. In Proc. of the *11th Int. Conf. On Information and Knowledge Management*, 662 –664, 2002.

[8] R. Luk, H. Leong, T. Dillon, A. Chan, W. Croft, J. Allan. A Survey in indexing and searching XML documents. *JASIST*, 53(6):415–437, 2002.

[9] C. Kanne, G. Moerkotte. Efficient storage XML data. In Proc. of the *16th ICDE*, 198, 2000.

[10] E. Kotsakis. Structured information retrieval in XML documents. In Proc. of the *ACM SAC'02*, 663 – 667, 2002.

[11] I. Manolescu, D. Florescu, D. Kossmann, F. Xhumari, D. Olteanu. Agora: Living with XML and relational database. In Proc. *VLDB*, 623 – 626, 2000.

[12] I. H. Witten, A. Moffat, T. C. Bell *Managing Gigabytes*, Morgan Kaufmann, 1999.

[13] A. Trotman. Searching structured documents. *IP&M* 40:619–632, 2004.