A flexible object-oriented system for teaching and learning structured IR

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 y de Telecomunicación,
Universidad de Granada, C.P. 18071, Granada, Spain
{lci,jmfluna,jhg,aeromero}@decsai.ugr.es

Teaching (Structured) Information Retrieval in a postgraduate programme in Computer Science could be a difficult task, but the situation get worse when IR is considered as a mere application of the fundamentals in the programme core. In this case, the (usual) necessity of explaining the basic IR concepts joined with the necessity of doing some experimentation with the model play against the (available) time in which the student might acquires the knowledge. In this paper we present a platform that could help the student to understand about what issues are specific to efficient Structured Information Retrieval in a Probabilistic Graphical Model postgraduate programme.

Information Retrieval System, Structured Documents, XML, Probabilistic Graphical Models.

1. INTRODUCTION

Information Retrieval (IR) is a leading research field nowadays around the world, because the effective and efficient access to the great amount of information resources available in the Internet, among other sources, is a real problem. From an educational point of view, this fact has caused an increasing presence of IR courses in postgraduate studies. In general, we can say that the contents of these courses depend on the convenience of the research groups working on IR in each particular University. Although these facts could be interesting, since the students might be closer to their future research field, it also has some inconveniences. For instance, the fundamentals are spread out into different (non-related) disciplines, being hard to establish connections between researches, making difficult the student's mobility between different Universities.

Despite this fact, there are also some practical problems that appear when teaching IR. In this paper we are going to present the experiences that we have had in the postgraduate study: "Probabilistic Graphical Models (PGM) in Artificial Intelligence and Data Mining" (PGM, 2006). This is a two year programme (resulting from the collaboration during the last years between different Spanish research groups in the field of PGMs) for graduates who have already completed a computing or mathematics degree. The purpose is to train students with a solid base in PGMs able to introduce themselves successfully to the industry or university sector.

Inside this programme there can be found an (elective) module about *Automatic Models for Information Management* where the students learn topics on Information Retrieval, Recommender Systems and Filtering (e-mail), but mainly as an application of the PGM's formalisms where tasks as design, learning, inference, and so on are discussed. Within this module, special emphasis in Structured (XML) Information Retrieval (Kotsakis; 2002) (Y. Chiaramella; 2001) is done, since we consider it a new (and promising) research field[1].

There are two main problems when teaching in this course:

- The student training with no knowledge of IR concepts. In Spain, very limited aspects of the IR field, if any, has been taught in the undergraduate curricula in Computer Science. We guess than this problem is common to others countries because in the ACM/IEEE Computing Curricula for Computer Science (Coputer Curricula, 2001), IR fields only appear as elective units (the last four) included in the body of knowledge *Information Management (IM)*, where the main topics are about database systems. The situation is worst when we consider the students with a degree in Mathematics, since none of the previous concepts are known. Therefore, a great amount of the effort goes to solve the deficiencies that the students have in the field of IR. In our opinion, we expect that this situation will get worse when the European Space of Higher Education is completely established.

- The difficulty of making experiments in laboratory. The laboratory work will deal with both the study of the basic architecture of an (structured) IRS and the implementation of models or parts of them based on PGMs. This will allow the students to understand the general features of an IRS, and the specific ones imposed by the probabilistic framework and also will give the possibility of making basic research and developing new IR models. Therefore, on the one hand, it does not make sense that students have to implement the great amount of pre-processing modules, but on the other, the use of some general platforms, like LEMUR (http://www.lemurproject.org) or Terrier (http://ir.dcs.gla.ac.uk/terrier), or mechanisms concerning efficient storage and accessing of XML documents (see for example (Trotman, 2004) and (Luk, Leong, Dillon, Chan, Croft, Allan; 2002)), would demand to implement specific operations needed by the retrieval models based on

[1] Although SGML (the first standard for structured documents) was developed in the 1960, it was not until 1998, when XML 1.0 (the most popular standard for structured documents) became a W3C Recommendation.

PGMs, basically because they have not been designed to work directly with them. This would be a very hard and long work, beyond of the objectives of the module.

The objective in this paper is to present an object-oriented system that could help the student to solve the second problem because there does not exists a concrete software solution ready to implement PGM-based models. Thus, we have decided to implement our own platform for these models, called Garnata, making it extensible for a future use. So in Section 2 we present Garnata as a choice for these students, who want to dive into the XML IR world, and want to experiment and evaluate new probabilistic-based models. Section 3 describes the methodological advantages of the use of an object-oriented system in the postgraduate programme. Finally, some conclusions and future works are presented.

2. SYSTEM OVERVIEW

Garnata was born as an implementation completely adapted to the models based on PGM 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 an extensible and wide range of classes and a complete set of utility programs. At this moment, it implements our three models for structured documents: BNR-SD (based on Bayesian networks, evolved from the one presented in (de Campos, Fernández-Luna, Huete; 2003), and also SID and CID models, based on Influence Diagrams (de Campos, Fernández-Luna, Huete; 2004, 2005). In the following subsections we shall study its architecture from the indexing and querying points of view, presenting its main features.

2.1. INDEXING SUBSYSTEM

2.1.1.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 and create *virtual units* (see below). Moreover, as stated in (Baeza-Yates, Ribeiro-Neto; 2002), 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 pre-computed 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 (Figure 1 shows all these concepts).

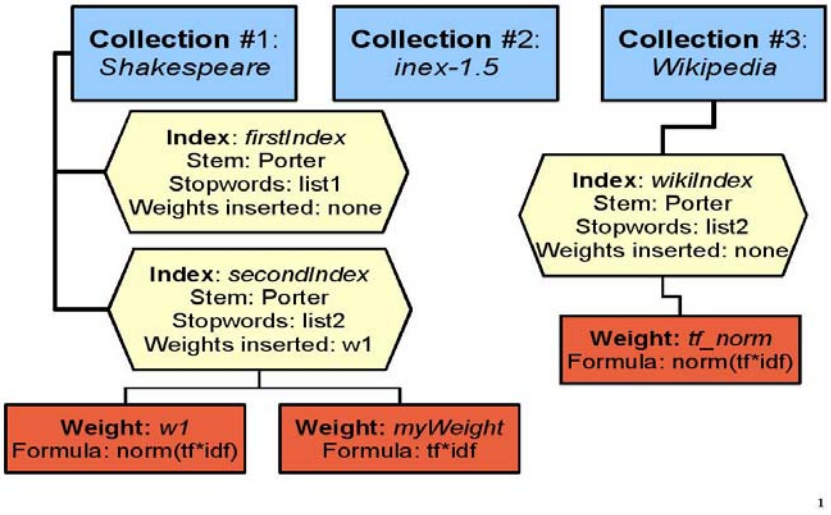

FIGURE 1: Concepts of indexing subsystem.

2.1.2.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 (Witten, Moffat, Bell; 1999). 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[2]. 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 2 for more details):

- Given a non-final unit, returning the list of identifiers of the units that it contains. For unit 1 in Figure 2, 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 2, it returns (2, 1).
- Given a final unit, returning the list of contained terms (known as the *direct index*). For unit 5 in Figure 2, it returns (1, 2).

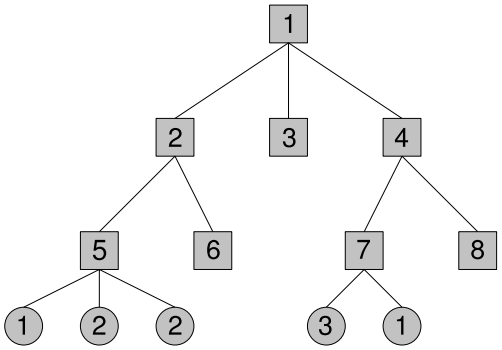

FIGURE 2: Relating terms (circles) and units (squares)

We have supposed that a structural unit contains text or other units, never both of them. In the cases we find units of this kind, we create a *virtual unit* with the text contained by this unit, and we leave the first only as a container. We present an example of the tree built with the following paragraph in Figure 3:

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

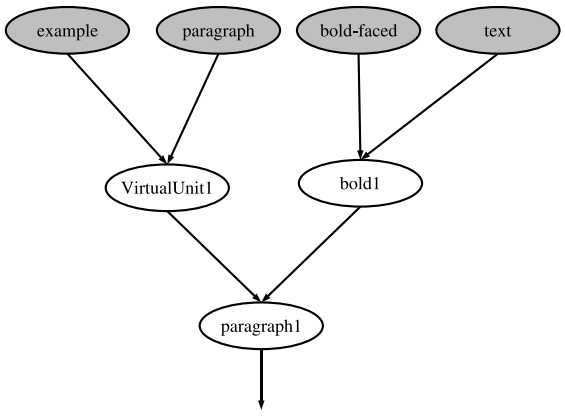

FIGURE 3: Relating terms (circles) and units (squares)

[2] 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.

Although the indexing subsystem of Garnata is not aimed at any particular IR model, all of the previous operations are specifically designed for the BNR-SD, SID and CID models (de Campos, Fernández-Luna, Huete; 2004).

We have abstracted the majority of I/O operations, creating several classes to read and write basic types (integer, float, …) from a file. A mechanism to perform compression is also included, with the possibility of including more algorithms. The format is very simple, but nevertheless transparent to the programmer; he does not need and should not care about the intrinsic of the compression algorithms, and I/O routines. We try to make the system more accessible (creating a layer of abstraction) and, on the other hand, more efficient (Fuhr, Govert; 2002) (Witten, Moffat, Bell; 1999).

2.2. QUERYING SUBSYSTEM

The 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):

- Query is parsed, and occurrences of the component terms are retrieved from disk.
- For each occurrence, implied final units are read into memory (if not already there).
- For each final unit, its descendants are read into memory (if not already there).
- Propagation is carried out (and utility computation, if we are using SID or CID models), units are sorted by its probability (or utility) 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[3], so for each cache slot, we will have several candidates (those identifiers being the hash inverse of the slot identifier).

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

3. METHODOLOGICAL ADVANTAGES

In this section, we discuss the advantages of the system, at a pedagogical point of view. Firstly, we study the system at a conceptual level, showing the advantages of an object-oriented system for learning purposes. Secondly, we explore all the system possibilities for the users to extend the system and use it as a platform for future developments.

3.1. CONCEPTUAL LEVEL

In Garnata, we provide a class for every entity recognizable in a structured IR system. This makes very easy to understand how an IR system works. While there are program-specific classes (i.e.: those aimed to the program themselves), we can find some typical classes in an IR system, and the relationships among them. The following list is a short description of the most important classes:

- *Term* represents a certain term in the system, with the string, its identifier, total frequency in the collection, and a pointer to its occurrence.
- *Occurrence* is the posting list for a term. It includes four lists: the list of documents (units) a term appears in, the frequency of each one, the weight of the term for each unit and the pointers to the list of positions inside the unit.
- *Lexicon*: a structure where terms are stored. We can search by identifier in O(1), and by term string in O(log n) (n being the number of terms).
- *Unit* (and its derivatives *UnitWriter* and *UnitReader*) represents a structural unit of the collection with its identifier, its tag identifier, the file, the position in its container, the number of items (term of units) contained, the weight of the unit in its container, and some metainformation about the kind (retrievable/not retrievable or final/not final).

[3] Identifier mod N, with N the size of the cache.

- *DTD*: list of different tags of the indexed collection. We associate one identifier to each tag, allowing different tags in different DTDs having the same string. A class *DTDList* manages the different DTDs, and it makes possible working with different DTDs on the collection.

- *Collection* represents a collection of XML files. It contains an identifier, the list of files, and a brief description of the collection.

- *Index* represents a certain index on one collection of the system. The class *IndexBuilder* makes the index, and *IndexReader* reads information from the index.

- *RetrievalModel* represents a certain information retrieval model. It contains generic operations to implement that model.

- *Query* represents a query to the system. At the moment, no standard language is supported.

- *XMLIndexer* manages all the SAX events, sending them to the *IndexBuilder*.

- *Weight*: abstract class that implements all the utilities to build a weighting scheme. Class *WeightBNR-SD* is built on top of this.

- *BNR-SD*, *SID* and *CID*classes for the so-called retrieval models. They have, as an ancestor, class *_BNR-SD*, an abstract class implementing probability propagation on a Bayesian network following our scheme.

3.2. IMPLEMENTING NEW MODELS

Developing new structured IR models with Garnata is easy. The system has been designed with a full degree of parameterization, so new developments are possible.

- At *indexing level*, we have the following options:
  - We can do *stemming* or not. At this point, only Porter's algorithm is included, but it is possible to add more in the future.
  - We can choose the list of *stopwords* among several we have introduced in the system. Having this, we can compare the performance of the system using different stopword lists.
  - A lot of information is gathered at this level, useful for future models:
- *At textual level*: term frequencies and term positions for each unit.
- At *structural level*: container unit and contained units, for each structural unit, in order to easily retrieve the whole tree of units. For each unit, *XPath* route is also stored.
- At *{ retrieval time*, we can choose the following items:
  - We can give the system a *maximum number of items* to retrieve, in order to obtain a medium-sized list of results.
  - We can select a *minimum value of probability* (RSV) for each unit to be retrieved. If a unit does not reach this value, it should not be retrieved.
  - At the moment, three are different retrieval models: BNR-SD, SID and CID. Easily way of adding new weighting schemes, maintaining several pre-computed weight files, outside a certain index, and insert one by one into the index, when necessary. This is done because:
  - Weight computation is a heavy task, and it should not be done more than once. Thus, weights are kept apart, and inserted into the index when needed.
  - Very often, while doing system evaluation, several weighting schemes are compared. Because of this, we think it is better to keep the weights outside, evaluate one by one, and leave inserted the best one.
- In the influence diagram-based models, we can choose among three methods to compute unit utilities (RSV) (de Campos, Fernández-Luna, Huete; 2004).

From a point of view of a student using this platform, there are two options: the first one would be to interact with Garnata with a text interface, giving some commands (in the style of Smart retrieval system). With such a dialogue, the user can change the values of the parameters depending on the experiments to be performed. The second possibility is to develop new classes implementing new models starting from the abstract classes provided. This is an easy task (only pervaded with the inherent difficulty of the programming itself) helped by a complete and comprehensive API documentation. Recompiling the whole software is therefore required.

4. CONCLUSIONS AND FUTURE WORKS

In this paper, we have presented Garnata, a flexible Information Retrieval System based on PGMs for structured documents, from a pedagogical point of view. This software is intended to teach the IR basics to students of a postgraduate degree on PGMs in a subject related to IR, but also the specific features that those models using these probabilistic frameworks present.

As a future work, we want to perform a detailed survey of the existing IRSs, included Garnata, from a teaching and learning perspective, determining their advantages and disadvantages from different points of view. We think this could be a very useful piece of work in order to make the decision of the software to use in each specific IR course.

Acknowledgements

This work has been jointly supported by the Spanish Ministerio de Educación y Ciencia under Project n. TIN2005-02516 and the Junta de Andalucía under Project. N. TIC276

REFERENCES

R. Baeza-Yates, B. Ribeiro-Neto (2002) Modern Information Retrieval, Addison Wesley.

Y. Chiaramella (2001). Information retrieval and structured documents. Lecture Notes on Computer Science, **1980**, 291–314.

Computer Curricula (2001). Final Report of the Joint ACM/IEEE-CS Task Force on Computing Curricula 2001 for Computer Science. http://acm.org/education/curric_vols/cc2001.pdf

L.M. de Campos, J.M Fernández-Luna, and J.F. Huete (2003) The BNR model: foundations and performance of a Bayesian network-based retrieval model. International Journal of Approximate Reasoning, **34**, 265–285.

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

L.M. de Campos, J.M Fernández-Luna, and J.F. Huete (2005) Improving the Context-based Influence Diagram Model for structured document retrieval: removing topological restrictions and adding new evaluation methods. Lecture Notes on Computer Science, **3408**, 215–229.

INEX. http://inex.is.informatik.uni-duisburg.de.

N. Fuhr, N. Govert (2002) Index Compression vs. retrieval time of inverted files for XML documents. Proceeding of the 11th Int. Conf. On Information and Knowledge Management, 662–664, 2002.

R. Luk, H. Leong, T. Dillon, A. Chan, W. Croft, J. Allan (2002) A Survey in indexing and searching XML documents. Journal of the American Society for Information Sciences and Technology , **53(6)**, 415–437, 2002.

C. Kanne, G. Moerkotte (2000) Efficient storage XML data. Proceedings of the 16th ICDE conference, 198.

E. Kotsakis (2002) Structured information retrieval in XML documents. Proceedings of the ACM SAC'02, 663– 667.

I. H. Witten, A. Moffat, T. C. Bell (1999) Managing Gigabytes, Morgan Kaufmann.

PGM (2006). Modelos Probabilísticos para la Inteligencia Artificial y Minería de Datos. http://www.ual.es/personal/asalmero/mpiamd.

A. Trotman (2004) Searching structured documents., Information Processing & Management, **40**, 619—632.