XOR - XML Oriented Retrieval Language

Shlomo Geva
Queensland University of Technology (QUT)
2 George St, Brisbane
Q 4001, Australia
s.geva@qut.edu.au

Marcus Hassler
Universität Klagenfurt
Universitätsstrae 65-67 9020
Klagenfurt
Austria
Marcus.Hassler@hekkas.com

Xavier Tannier
Ecole Nationale Supérieure des Mines
158 Cours Fauriel
42023 Saint-Etienne, France
tannier@emse.fr

ABSTRACT

The wide acceptance and rapidly growing use of XML as a standard storage and retrieval data format blurs the historical divide that exists between Information Retrieval and Database Retrieval. On the structured database retrieval side it is now possible to support highly structured access to documents using XML specific tools such as XPath, XQuery, XQL and more. On the information retrieval side it is possible to support access to the XML documents using XML specific retrieval query languages such as NEXI. None of the above are intended for end-users, but rather as enabling back-end technologies. In this paper we introduce XOR - a new XML Oriented Retrieval language that is designed to facilitate query specification with a strong IR flavour. XOR is backwards compatible with NEXI, but significantly extends its functionality overcoming many of its restrictions and limitations. While XOR itself is not an end-user tool, it is designed with the explicit goal of supporting IR, and more specifically, user oriented interfaces such as Natural Language Queries (NLQ) or interactive user interfaces. XOR provides the missing functionality that none of the existing XML retrieval tools support, and which advanced IR requires.

1. INTRODUCTION

A historical divide exists between Information Retrieval and Database Retrieval. The former is mostly concerned with text documents or web pages with minimal structure, while the later is primarily concerned with highly and strictly structured documents. XML supports the representation of all types of documents, catering for the full spectrum - from unstructured to highly structured documents. The gap between IR and traditional Database approaches is closing. Indeed, many approaches to XML-IR rely on database technology as a back-end system, rather than rely on IR specific file structures. XML retrieval tools from W3C, such as XPath [1] or XQuery [2] are highly sophisticated query languages. For information retrieval applications, XPath and XQuery are arguably completly over the top. Information retrieval queries are often loosely defined, vague, or even ambiguous. NEXI [3] is an alternative query language that was designed to drastically cut-down XPath, while being extended with explicit IR flavoured functionality and with implicit IR flavoured interpretation.

NEXI (Narrowed Extended XPath I) is a language for Information Retrieval over XML document collections (XML-IR), proposed and used by INEX - the INitiative for the Evaluation of XML Retrieval - since 2004[1]. NEXI offers a good compromise between the need to formally express structural and textual constraints on the one hand, and the ability to write IR flavoured queries on the other hand.

At the same time, since 2004, INEX ran a Natural Language Processing task. Rather than requiring participants to implement NLP based XML search engines, the main task of the NLP track is the automatic translation of an expressed natural language information need into a formal NEXI query. The automatically generated formal queries are then evaluated by a standard XML search engine that is provided by the track organizers.

The results from the first two years are very encouraging, but it appears now that NEXI specifications constrain further research and development of this approach. Indeed, an analysis of natural language queries can lead to identification of interesting features or relations between terms (or elements). But the need to use NEXI as a pivot language prevents the use of this knowledge when formulating queries. It should be noted that XPath does not support the necessary functionality either - it is not the simplification from XPath to NEXI that is the cause of the problem.

Here we introduce the XML-Oriented Retrieval (XOR) formal query language, an extension of NEXI that supports new features. The extensions are not a re-introduction of XPath features that were removed when NEXI was designed; rather, the extensions are geared towards more expressive, albeit more complex, queries; however it is primarily intended for use by automatic query generators (such as natural language interfaces or interactive user interfaces that are guided by explicit user feedback in response to clarification requests from the system). This is an important trait to notice, because XOR is a formal XML-IR language that is NOT designed for direct use by people - not even XML experts who are the users of XPath and XQuery. The

SIGIR 2006 XML Element Retrieval Methodology workshop August 10, 2006, Seattle, Washington, USA. Copyright of this article remains with the authors.

[1] http://inex.is.informatik.uni-duisburg.de/2006/

language is designed purely as an intermediary in IR application, and in our case specifically with NLP in mind to facilitate explicit support of natural language queries.

Nevertheless, the language can be used by end-users or expert-users if desired. It is still much simpler than XPath, XQuery, or XQL, for instance. the *XOR* language is extensible and backwards compatible with NEXI, meaning that any query written in NEXI can be successfully parsed and processed by the *XOR* parser. This is very important because it allows researchers and developers to test systems that are based on *XOR* with the datasets and assessment data and tools that were developed by the INEX initiative over the past 5 years, as well as into the future.

XOR is designed from the outset as an open ended extensible language. The support of specific functionality in *XOR* is left to the implementation of a search engine and is not part of the *XOR* parser. However, *XOR* provides a concise and simple syntax for extension. This is another feature that distinguishes *XOR* from most other XML query language specifications. For instance, if the query specifies constraints over part-of-speech (POS) tags, then it is a search engine implementation issue with respect to which POS tagger it supports; furthermore, it is even possible that the search engine will choose to ignore the POS constraints altogether.

The *XOR* parser implementation is also required to perform an additional transformation that is not commonly found when a language syntax is defined. The parser converts the query from infix notation to postfix notation (or Reverse Polish Notation). This transformation is designed to assist the development of the back-end search engines that support *XOR* and to lower the threshold to participation in eveluation forums like INEX.

In what follows we describe the *XOR* language and provide examples of the syntax and of queries. We then describe the query transformation to Reverse Polish Notation (RPN), and provide some early results that were obtained with *XOR* over the INEX repository of XML documents, topics, and relevance assessments. The appendix provides the BNF diagrams of *XOR*.

2. NEXI

NEXI is a formal query language that is based on XPath. It has been designed to allow a simple but efficient representation of information needs for XML information retrieval.

The syntax of NEXI is similar to XPath, however, it only uses the descendant axis step, and extends XPath by incorporating an *"about"* clause to provide IR flavour to queries. NEXI's syntax is:

```
//A[about(.//B,C)]
```

where A is the context path, B is the relative path and C is the content requirement.

It is possible for a single NEXI query to contain more than one information request.[2]. Therefore the query *"Return paragraphs about watermarking in article containing a paragraph about data embedding"* can be represented as follows:

```
//article[about(.//p, "data embedding")]
       //p[about(.,watermarking)]
```

The query contains two information requests (or sub-topics):

```
//article//p[about(.,watermarking)]
```

And:

```
//article[about(.//p, "data embedding")]
```

In NEXI each information request is specified by an *'about'* clause. However, elements matching the rightmost *'about'* clause, here the first request, are returned to the user. INEX refers to these requests and elements as "target requests" and "target elements". Elements that match other *"about"* clauses, here the second request, are used to support the return elements in ranking. We refer to these requests and elements as "support requests" and "support elements". In order to be valid, each NEXI query must have at least one target request, along with any number of support requests.

While NEXI does support the specification of more complex queries using parenthesis and the boolean operators AND and OR, the interpretation of such features is not strict. In standard IR query terms are regarded as retrieval hints, and therefore query expansion is allowed (even expected). In the same manner, in the interpretation of NEXI, all structural specifications are also taken merely as hints. The NEXI expression is not regarded as deterministic and it is left to the search engine to interpret it. For instance, the AND operator is commonly evaluated with OR semantics [4] [5] [6] [7] by search engines. For sure, any system that implements a simple keyword search, such as represented by the title element of an INEX topic, effectively performs an implicit OR (because the title element contains all keywords that appear in the castitle element, but the structure and boolean conditions are lost.) A common approach to the implementation of AND and OR is to use the fuzzy-like operators whereby scores are multiplied (AND) or added (OR). The AND operator is no longer interpreted striclty and takes on an OR flavour.

3. LIMITATIONS OF NEXI

Translation of natural language queries into a formal language like NEXI presents some limitations, mainly due to the fact that the natural language preprocessor cannot specify certain constraints to the retrieval system. The formal language, if not specifically designed with this aim, is pivotal in preventing helpful "communication" between both systems.

[2]NEXI does support multiple path specification whereby a query can be be return multiple elemtn types. It does not however provide explicit support to multiple distinct search requests

For example, it is not possible to consider the following features within single NEXI queries[3]:

- NEXI allows only single queries (with only one target element). This is very limiting when trying to express the same information need in several ways. For example, suppose that we are seeking information concerning Einstein's 1905 article about electrodynamics. We may directly look for this article:

```
//article[.//year = 1905
         AND about(.//author, Einstein)
         AND about(.//*, electrodynamics)]
```

  But we could also want to see some of the many articles that explain or discuss this article...

```
//article[about(., Einstein article 1905)
         AND about(., electrodynamics)
```

  ... or 1900's articles on this subject to have an idea of the state of the art at this period:

```
//article[about(.//year, 1900)
         AND about(., electrodynamics)
```

  Thus a single information need (*"I want to understand everything about this famous Einstein's 1905 article about electrodynamics"*) may be represented in at least three different complementary queries.

- NEXI handles only the `'about'` predicate, while others could be of interest for the search process. For example, with NEXI, whether the terms should be matched strictly or with potential syntactic, semantic variations like stemming or term expansion, is up to the back-end system. An NLP system cannot intervene and specify the desired interpretation even if it is available.

- NEXI does not allow the user to refer to more than one article. Many requests in INEX concern bibliographic references, but search engines are not explicitly asked to look at referred articles (if available) and, to date, all implementations of NEXI are restricted to search the references section titles - a very narrow window to referenced articles indeed.

  e.g.: *"Find bibliographic references that are about text categorisation where Support Vector Machines (SVM) categoriser is used."* (Topic 136), which is translated in NEXI as:

```
//bib[about(., text categorisation)
  AND about(., "Support Vector Machines" SVM)]
```

[3]In addition to this list, NEXI is not designed to deal with many database-oriented constraints, particularly when dealing with strongly typed elements, but we are not concerned with this here

- Finally, NEXI lacks a way to express any additional desired features concerning the tags or the search terms[4]. Among these features, we can cite the minimum / maximum size of the element, the type of interpretation (strict or vague), part-of-speech, word case, language, or any other useful feature imagined. *XOR* is designed to support an open ended set of selection qualifiers.

NEXI has been designed as a reduction of XPath to handle only information retrieval oriented features. We think that it is now time to extend NEXI with more powerful IR oriented features.

4. XOR LANGUAGE SPECIFICATION

The *XOR* language is almost entirely compatible with the previously defined NEXI specification [3]. The extensions mainly concern the (automatic) query formulation capabilities of combining several queries into a single query, more elaborate specifications of paths and terms, and a larger set of matching predicates for specific information needs.

4.1 Negation operator

The sign `'-'` of NEXI is supported, but it is sufficient to clearly express that a term must not appear in returned elements. We propose the negation of the *about* clause, semantically more adequate, and syntactically more powerful.

For example, a query like *I am not looking for devices for computer-based training.*[5] does not mean that terms *"devices"* and *"computer-based training"* must not appear in elements (as would NEXI by `-devices -"computer-based training"`), but that they must not be found *together*[6]. An expression like the following better suits the information need:

```
NOT about(., devices "computer-based training")
```

4.2 Logical operators for queries

We justified in the introduction the utility of allowing multiple expressions in the same query. XOR enables the specification of a set of CAS queries combined with boolean operators.

This step includes strict bracketing to avoid ambiguities and may contain negations (using `NOT`). An examples of syntactically correct *XOR* query, that is not valid NEXI is

```
(//A[about(.,B)] AND //A[about(.,D)]) OR
(//A[about(.,B)] AND NOT (//A[about(.,C)] OR
                  //A[about(.,D)]))
```

[4]See Sigurbjornsson and Trotman "Queries: INEX 2003 working group report" where they state "There already exist two data types, numeric and string. This is anticipated to expand in the future to include names, units of measure, and even geographic locations. The language must be extensible to include these at a future date."

[5]From Topic 196, INEX 2004

[6]The complete query is about education problems raised by computer-based training, and then the term *"computer-based training"* is found in a relevant element.

Thus, the example of the 1905 Einstein article can be simply translated into

```
//article[about(.//year,1905)
         AND about(.//author, Einstein)
         AND about(.//*, electrodynamics)]
  AND
//article[about(., Einstein article 1905)
         AND about(., electrodynamics)
  AND
//article[about(.//year, 1900)
         AND about(., electrodynamics)
```

In practice, as search engines return a list of elements that are independent from each other, and not groups of elements, the operators `'AND'` et `'OR'` usually have exactly the same semantics[7]. It remains as a challenge for the search engine to merge (or fuse) the results of the distinct queries which may possess completely different statistics.

4.3 Path extensions

XOR supports the specification of additional path constraints. This information is optional and expressed within curly brackets as a set of {`key:value`} pairs. Thus, this information can easily be extended to include further types of matching. Currently, the following `key:value` pairs are supported in our back-end, but more are possible. Consider the following examples:

`match:strict|vague` Specifies the kind of structural requirement matching, influencing the result set and ranking. For instance:

```
//article[about(.//year{match:strict},1905)
         AND about(.//author, Einstein)
         AND about(.//*, electrodynamics)]
```

Here insisting that "1905" must be found in an element tagged as "year". Or:

```
//section{match:vague}[about(.,cars)]
```

Here indicating that a section-like element is required. The default is implementation defined, but vague seems most appropriate in the IR context.

Besides additional types of paths, wildcards are allowed to specify node names. The following patterns are valid:

`//*` e.g., all node names, //xyz, //jim

`//node*` e.g., //node6, //nodename

`//*node` e.g., //mynode, //this_test_node

`//*node*` e.g., //mynodeextension, //the_node_quantifier

[7]For example, asking for *sections and paragraphs* or *sections or paragraphs* will lead in both cases to a ranked list of sections and paragraphs.

The intended use of this feature is in situation where the DTD is not available, or too complex to enumerate possible matches. Far from being the exception, this may well be the norm, particularly with private or dynamic collections. The use of wildcards is nevertheless resting on the assumption that meaningful tag names are used in the collection (in a natural language). The Wikipedia XML collection that is used by INEX in 2006 is a good example of precisely this situation.

4.4 Term extensions

For the purpose of more exact query matching *XOR* enables the addition of further information to a given term (in the same manner as to the path). Again, the additional information is optional and expressed within curly brackets as a set of `key:value` pairs. Consider the following examples:

`POS: part-of-speech` Specifies the kind of Part-Of-Speech (POS) tag.

```
//abstract{match:vague}[about(.,Go{POS:Noun})]
```

Here we are looking for the game "GO" not the verb.

`CASE: upper|lower` Specifies the case of the text - useful for acronyms for instance.

```
//section{match:vague}[about(.,AJAR{CASE:upper})]
```

Here we are looking for the acronym for "Acronyms, Jargon, Abbreviations and Rubbish", not ajar meaning slightly open.

4.5 Logical operator qualifiers

It is possible with XOR to qualify the logical operators. The interpretation of the qualifiers is again left to the search engine. For instance,

```
//article[about(.,Germany)
  AND{mode:strict} about(.,football)]//sec[about(.,Europe)]
```

Here we insist on strict interpretation of the logical AND operator.

4.6 Additional predicates

In the context of heterogeneous information needs and highly sophisticated search techniques, a single `about` predicate for matching seems too restrictive. For this reason *XOR* implements several additional predicates, having similar format to the *about*() function:

LinkTo((XLink—XPointer),keywords) matches documents that are linked to by the context element. For instance, the implementation could check that the linked-to element is *about*() the keywords.

LinkFrom() matches elements which link to the context element. For instance, the implementation could check that the linked-from element is *about*() the keywords.

Contains() This is the same as the XPath function and has a strict interpretation.

lt(), eq(),gt() for less than, equal, greater than respectively. Necessary for numeric element comparisons, or fixed format fields because XML files do very often contain both free text and strictly typed elements. The motivation for using functions rather than the traditional symbols is twofold - it keeps the language much simpler and all operators are treated uniformly.

The XOR specifications do not require the parser to check that functions actually exist. This is left to the backend search engine. So any implementation of XOR can create new functions and the parser does not get involved as long as the syntax of the function call is valid. The about() and eq() functions, for instance, are both treated identically in the syntax and both are left to the search engine to implement.

5. REVERSE POLISH NOTATION

In order to support the implementation of back-end processors, the actual XOR parser checks the validity of XOR expressions and returns a vector (of text lines) containing the translation of the expression from infix to postfix notation, or as it is often known *Reverse Polish Notation* (RPN) [8]. Each line in the RPN is a simple NEXI expression. The following example illustrates this transformation.

***XOR* Query:**

```
//article[about(.,Germany)
  AND NOT about(.,football)]//sec[about(.,Europe)]
AND
//article[about(., European union enlargement)
  AND about(.//*,German point of view)]
```

***RPN*:**

```
//article[about(.//*, German point of view)]
//article[about(., European union enlargement)]
AND
//article//sec[about(., Europe)]
//article[about(., football)]
//article[about(., Germany)]
ANDNOT
SUPPORT
AND
```

This notation should be read with the following binding:

```
//article[about(.//*, German point of view)]
//article[about(., European union
enlargement)]
AND
//article//sec[about(., Europe)]
  //article[about(., football)]
  //article[about(., Germany)]
  ANDNOT
SUPPORT
AND
```

The binary operator $SUPPORT(X, Y)$ means that the second argument (here, an article about Germany but not football) is used as a support to the selection of the target element, which is the first argument (here, a section about Europe). Implementation of AND, ANDNOT, OR, SUPPORT are up to the search engine. This is where the developers of the search engine have the freedom of interpretation - this is akin to the IR flavoured about() function in contrast to the XPath strict contains() function.
Inverted lists are generated as a product of the atomic function calls within the XOR filters, like about() or contains() etc. These atomic units are presented as separate lines (search requests) in the RPN representation. The advantage of the RPN is that it allows for unlimited nesting of parenthesis and any path expression depth (one of the limitations of NEXI). Furthermore, the RPN format lends itself to simple implementation of search algorithms by systems that are based on the processing of inverted lists, a stack, and binary or unary list operators. We were able to easily incorporate XOR into GPX, a search engine that supports NEXI queries, and which is based on inverted list processing.

5.1 Important differences

Important to stress are the following oddities to XPath, etc.

- `//*` in NEXI means any descendant node, in XOR it means the context node or any descendant node. We often find that nodes contain both direct text and descendant nodes that contain more text. So selecting //sec//*[...] means select sections or descentedants of sections that satisfy the condition. To select only descendents of section in XOR we use //sec/*[...]

- `=` in XOR the function eq() is used instead. Similarly for other comparison operators. Instead of the NEXI query:

  ```
  //section[.//year = 1905]
  ```

  in XOR we would write:

  ```
  //section[eq(.//year,1905)]
  ```

  or perhaps:

  ```
  //section[eq(.//year{match:vague},1905{match:strict})]
  ```

[8] http://en.wikipedia.org/wiki/Reverse_Polish_Notation

6. IMPLEMENTATION EXAMPLE: GPX-XOR

We have used the GPX search engine as a back end system to test the *XOR* parser. The prupose of this experiment is no more than a sanity check and an example of how *XOR* might be implemented with existing search engines, that can already process simple NEXI expressions. GPX is an XML search engine that was used at INEX in 2004 and 2005. GPX is based on inverted lists - a detailed description can be found in [4], but suffice to say that the retrieval and score calculation for elements in each search request in the *XOR* RPN expression is largely unchanged. Each of the elements in the lists is scored with a TF-IDF variant by the standard GPX algorithm. The *XOR* operators AND, OR, ANDNOT and SUPPORT were implemented as described in the following sections.

6.1 OR(X,Y)

The OR operator is a union of two inverted lists, X and Y. Items in the lists identify XML result elements by file-id, full XPath expression, and relevance score. The OR operator performs a set union whereby elements that appear in both lists are merged and their scores combined. Other elements keep their original score.

6.2 AND(X,Y)

The AND operator was optionally implemented in one of three different ways. The default option is to simply implement it as OR(X,Y). This seems to work quite well in most instances, and also on average. However, in some queries the user *really* means AND. The second option is to implement it as a strict set intersect. Only XML elements that appear in both X and Y are kept, and their scores combined. This option is too restrictive because sometimes the lists contain overlapping elements and then the relationship with respect to AND is unclear. By insisting on a strict match many relevant results are lost. The third implementation keeps overlapping nodes, combines the scores, but keeps only the largest node (deepest common ancestor). In the experiments that we report in the next section, we used the first (default) option.

6.3 ANDNOT(X,Y)

The ANDNOT operator is implemented in a straight forward manner, and we adopted the the strictest interpretation - elimination. Any node in X that has an exact match in Y is eliminated. We assume that when users just want to discourage some keyword from appearing they will use the milder *"-keyword"* form of query specification. The list X is then returned. The *XOR* parser only allows the use of the NOT operator only in conjunction with AND, that is - X AND NOT Y - hence ANDNOT.

6.4 SUPPORT(X,Y)

The SUPPORT operator takes a list of nodes in Y that provide support to the selection of nodes from list X. For instance, when we look for paragraphs about Americium in articles with abstract about the Periodic Table, the result elements are paragraphs, and paragraphs are supported by abstracts about the Periodic Table. Both the support and result elements must have a common ancestor within the document tree, so the supporting abstract must appear in the same document as the supported paragraphs. The support operator identifies for each result element in X, all the support elements in Y, and combines the scores. All the elements from X are returned but those with support have an increased score.

6.5 Preliminary Results

The conversion of our existing search engine to support *XOR* took about one day (although it took a bit longer to iron out some bugs.) We were able to test *XOR* interactively with numerous queries with very pleasing results. We also tested GPX-XOR, and the RPN approach, against the INEX 2005 tasks with the Context and Strucure (CO+S) topics. These topics were all specified in NEXI - a subset of *XOR*. Figures 1 to 3 depicts the performance of the GPX-XOR system with the three best performing official submissions in INEX 2005 in the COS.Thorough task. Each of the baseline submissions (TWENETE, QUT, IBM) produced the best result in either the Strict, Generalised, or GenLifted quantization respectively, as measured by the MAep value. The results that we obtained with *XOR* are very promising, and the performance exceeded that of the baseline submissions in all 3 cases. The uppermost curve in all figures belongs to GPX-XOR. Of course this result can only be taken as a sanity check. This is not a definitive evaluation since there is a risk of overfitting the results to assessments when experimenting (and debugging) with known qrels. Similarly good results were obtained with all the INEX tasks, when compared with the GPX (NEXI) baseline system. We will be able to test *XOR* more rigorously with unseen qrels at INEX 2006. The point that we wish to make is not the specific performance of GPX-XOR, but rather the simplicity of converting an existing NEXI search engine to *XOR* without any loss in performance.

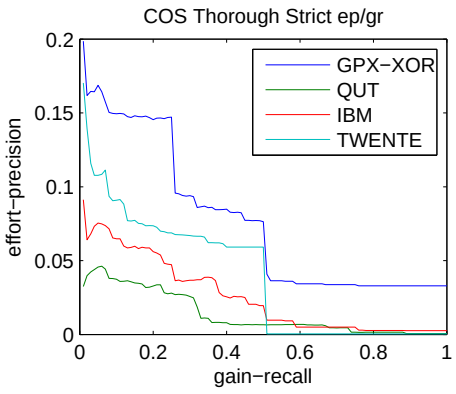

Figure 1: Strict quantization

7. CONCLUSIONS

We have presented *XOR*, a language explicitly designed to support IR in XML collections. More specifically, *XOR* was designed with the experience gained in the INEX natural language queries task, to support more elaborate search options than would be possible with NEXI. Yet, *XOR* is not extended with XPath like functionality, but rather with functionality that is IR oriented and that is not supported by existing XML search languages. More specifically, *XOR* extends selection specification of search terms, allowing for

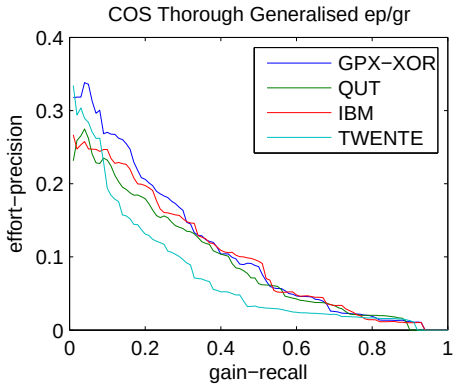

Figure 2: Generalised quantization

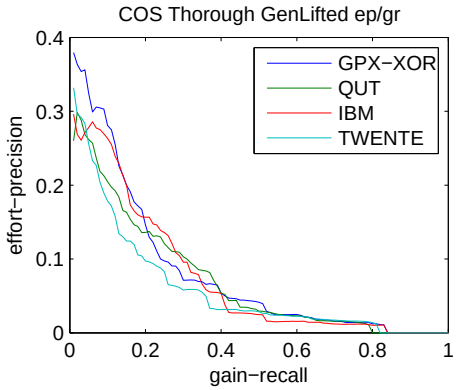

Figure 3: GenLifted quantization

more refined control of query content expansion. It also extends the selection specification of XPath expressions with wildcards, extends the allowable overall query complexity, and specifies a transformation from infix to postfix notation for easier integration into existing search engines. *XOR* is open ended and thus future work will concentrate on providing more functionality in *XOR* and on open source search engine implementation. *XOR* can support easier integration of advanced XML IR techniques. Support for *XOR* will reduce the need to develop complete search engines to implement powerful user interfaces to XML IR systems, such as natural language query interfaces.

8. REFERENCES

[1] XML Path Language (XPath) 2.0, W3C Candidate Recommendation 8 June 2006. http://www.w3.org/TR/xpath20/

[2] W3C XML Query (XQuery), XML Query is currently a W3C Candidate Recommendation. http://www.w3.org/XML/Query/

[3] A. Trotman and B. Sigurbjrnsson, Narrowed Extended XPath I (NEXI). In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlàvik, editors, *Advances in XML Information Retrieval. Third Workshop of the Initiative for the Evaluation of XML retrieval (INEX)*, volume 3493 of *Lecture Notes in Computer Science*, pages 16–40, Schloss Dagstuhl, Germany, December 6-8, 2004, 2005. Springer-Verlag, New York City, NY, USA.

[4] S. Geva, GPX - Gardens Point XML IR at INEX 2005. Proceedings of INEX 2005, Schloss Dagstuhl, Germany, November 2005, in Springer, Lecture Notes in Computer Science LNCS 2006. To Appear. Pre-proceedings - http://inex.is.informatik.uni-duisburg.de/2005/pdf/inex-2005-preproceedings.pdf

[5] V. Mihajlovi?, G. Ramrez, T. Westerveld, D. Hiemstra, H. Ernst Blok and A. P. de Vries, TIJAH Scratches INEX 2005 Vague Element Selection, Overlap, Image Search, Relevance Feedback, and Users . Proceedings of INEX 2005, Schloss Dagstuhl, Germany, November 2005, in Springer, Lecture Notes in Computer Science LNCS 2006. To Appear. Pre-proceedings - http://inex.is.informatik.uni-duisburg.de/2005/pdf/inex-2005-preproceedings.pdf

[6] P. Arvola, J. Keklinen and M. Junkkari, TRIX Experiments at INEX 2005 . Proceedings of INEX 2005, Schloss Dagstuhl, Germany, November 2005, in Springer, Lecture Notes in Computer Science LNCS 2006. To Appear. Pre-proceedings - http://inex.is.informatik.uni-duisburg.de/2005/pdf/inex-2005-preproceedings.pdf

[7] R. Schenkel and M. Theobald, Relevance Feedback for Structural Query Expansion. Proceedings of INEX 2005, Schloss Dagstuhl, Germany, November 2005, in Springer, Lecture Notes in Computer Science LNCS 2006. To Appear. Pre-proceedings - http://inex.is.informatik.uni-duisburg.de/2005/pdf/inex-2005-preproceedings.pdf

APPENDIX

A. XOR SPECIFICATIONS

```
SKIP ::= " " | "\t" | "\n" | "\r" | "\f"

OR          ::= "or" | "OR"
AND         ::= "and" | "AND"
NOT         ::= "not" | "NOT"

ALPHANUMERIC ::= ["a"-"z","A"-"Z","_"]
NUMERIC      ::= ["0"-"9"]

STUFF        ::= "&" | "'" | "~" | "" | "#" | "`" | "_" | "" | "^" |
       "" | "" | "$" | "" | "" | "%" | "" | "?" | "!" | "" | ""
TERMRESTRICTION ::= "+" | "-"
SLASH           ::= "/" >
ASTERISK        ::= "*" >
ATTR            ::= "@" >
PIPE            ::= "|" >
LPAR            ::= "(" >
RPAR            ::= ")" >
LBRACK          ::= "[" >
RBRACK          ::= "]" >
LBRACE          ::= "{" >
RBRACE          ::= "}" >
COMMA           ::= "," >
COLON           ::= ":" >
DOT             ::= "." >
ARITHMETIC      ::= "<" | ">" | "=" | "<=" | ">="
//////////
// NON terminals
//////////
Start        ::= Query
Query        ::= (Cas | "(" Query ")") Query2
Query2       ::= ((AND | OR) Query Query2 | "")
Cas          ::= AbsolutePath

AbsolutePath     ::= ( "/" ["/"] (Node | Attribute) [PathConstraints] [Filter] )+
RelativePath     ::= "." [AbsolutePath]
Node             ::= ["*"] Word ["*"] | "*" | "(" Node ("|" Node)+ ")"
Attribute        ::= "@" Word
PathConstraints  ::= "{" PathConstraint ("," PathConstraint)* "}"
PathConstraint   ::= Word ":" Word

Word             ::= (NUMERIC | ALPHANUMERIC)+

Filter           ::= "[" FilteredClause "]"
FilteredClause   ::= SimpleFilter | "(" FilteredClause ")
FilteredClause2  ::= ((AND | OR) FilteredClause FilteredClause2 | "")
SimpleFilter     ::= PredicateClause | ArithmeticClause
PredicateClause  ::= Predicate "(" RelativePath "," Keywords ")"
ArithmeticClause ::= RelativePath ("<" | ">" | "=" | "<=" | ">=") Word

Keywords           ::= (Keyword | Keyphrase | RestrictedKey)+
RestrictedKey      ::= ("+"|"-") (Keyword | Keyphrase)
Keyword            ::= Word [KeywordConstraints]
Keyphrase          ::= "\"" (Keyword)+ "\"" [KeywordConstraints]
KeywordConstraints ::= "{" KeywordConstraint ("," KeywordConstraint)* "}"
KeywordConstraint  ::= Word ":" Word
```