Retrieval Using Structure for Question Answering

Paul Ogilvie
Language Technologies Institute
School of Computer Science
Carnegie Mellon University
pto@lti.cs.cmu.edu

ABSTRACT

This paper examines the use of XML for modern extraction-based question answering (QA). We feel that the XML community has taken too narrow a view of structured retrieval, and that examining specific applications such as QA can give the XML retrieval community a broader view of the problems and challenges that structured retrieval faces. In our examination of QA, we argue that the next steps for retrieval supporting QA should be the use of a structured database for retrieval of passages. We believe retrieving passages could provide higher precision result lists without sacrificing the recall. This could greatly reduce the processing time for question extraction or allow the examination of more passages. The application of XML search technologies to QA is a realistic scenario and presents several interesting challenges to XML database systems. These challenges come in many forms: representation of structured documents in XML, indexing documents for efficient querying, and representation of queries. This paper explores these issues by considering techniques for question answering and also considers future directions for QA. Specific queries are provided as motivating examples.

1. INTRODUCTION

Much of the existing research on XML retrieval systems has focused on designing expressive query languages and developing efficient structures for the support of retrieval. Many of the examples in literature are motivated through use cases. Use cases are a good way to examine the requirements for some specific problems, and through their examination useful generalizations may be found. Work in this area has produced powerful query languages for XML such as XPath [1], XQuery [2], and NEXI [18]. However, this research has provided little guidance on knowing which tasks are more important to the users of an XML retrieval system. Other works have adapted existing database query languages such as SQL to XML [3][5]. There are no guarantees that the questions asked of traditional database systems will be the same as those of XML retrieval systems. These issues imply the query languages may not be able to express typical user needs and the structures may not be optimized for the most common search tasks.

The work in the INEX community has helped to provide some guidance by establishing a forum for the creation of real information needs [7]. The topics created for evaluation use either flat text queries or structured queries. All of the information needs evaluated in INEX require some notion of looking for text relevant to the information need. Systems are evaluated based on how well they produce rankings of document components that satisfy the user's information need.

One important recognition of INEX is the need to express the notion of what text is about in queries. Recognizing the role of approximate matching in XML search presents the need for ranking results by degree of match or relevance to the user's information need. Another important recognition of the INEX community is that the user may not know what types of document components may contain information relevant to their need.

However, INEX has its limitations. Many researchers have found it difficult to create realistic information needs that require structured queries. This may be a limitation of the nature of the corpus: a collection of journal articles about computer science with some document structure markup. The narrow domain may limit the kinds of information needs. Alternatively, the simple structural markup of the document may be the limiting factor. Regardless, we feel that it is useful to look beyond INEX to find additional realistic scenarios for XML retrieval.

Asking the "how" question of how XML can be useful for a broader range of users and problems raises interesting research questions that will advance the state of the art within XML. These users need not be human users - they may be systems that process XML data. Not only will this activity advance the start of the art for XML, but it will make XML research applicable to a wider audience.

With that in mind, this paper explores the use of structured retrieval for question answering (QA) systems. There are two general approaches to QA: knowledge-based and extraction-based. Knowledge-based QA uses a database of typically domain specific knowledge. Natural language questions are analyzed and converted into database queries that return the answers. More recent QA has been extraction-based. Questions are asked on a open-domain text corpus and answers are extracted from the text. This process roughly consists of three steps: question analysis, passage retrieval, and answer extraction. This paper will focus on extraction-based QA approaches, but with the goal of moving some of the processing usually done by the extraction component into the retrieval component. We view this as taking a small step from open domain extraction-based QA toward open domain knowledge-based QA (which many systems are currently working toward).

The process of examining what XML can and cannot do for QA will in turn point out challenges and issues for XML retrieval. These issues come in three forms: representation of queries, representation of structured text, and indexing of structured text for efficient retrieval. We argue that ranking is important, that query term weighting and proximity operations are needed, and that the structures of real documents may be overlapping. Overlapping structure will require new

TDM'04, the first Twente Data Management Workshop on XML Databases and Information Retrieval, Enschede, The Netherlands

mechanisms for indexing and retrieval.

The next section reviews QA and the role of search in QA. Section 3 provides specific motivating examples of how XML search could be used by QA systems. Section 4 draws on the examples to elaborate some of the challenges and limitations for XML. Section 5 describes some potential solutions to these problems, and Section 6 concludes the paper.

2. OVERVIEW OF QUESTION ANSWERING

In this section we describe approaches to QA. Techniques generally fall into two classes: knowledge-based QA and extraction-based QA. However, the distinction between the two is sometimes unclear, as some extraction-based QA approaches use knowledge bases and some knowledge-based QA systems use extraction technologies.

2.1 Knowledge-Based QA

Knowledge-based QA has a longer history than extraction-based QA. We will only briefly discuss the use of knowledge bases here as our focus is extraction-based methods. In this setting, knowledge or facts are represented within a database. The knowledge represented by these systems is typically domain specific, as construction of these knowledge bases is typically manual labor intensive. Natural language questions are analyzed and converted into database queries that will return the answer.

One early example of a knowledge-based QA system is Lunar [20]. Lunar parsed natural language questions into a formal query language using syntactic and semantic rules. The knowledge base contained chemical data about moon that was gathered on Apollo missions.

A more modern example of a QA system that uses knowledge bases is Start [9]. Start used natural language annotations to represent knowledge. Queries were compared to the annotations and matching annotations were returned. These annotations took the form of triples representing a relationship between a subject and an object. The annotations were extracted automatically from text. The knowledge represented by the simple annotations was limited, but could answer a surprising variety of questions.

2.2 Extraction-Based QA

Much of modern research in QA is extraction-based. Answers are sought in a large, open-domain text corpus and extracted from the text. The typical phases of an extraction question answering system are question analysis, document retrieval, passage extraction, and answer extraction.

The question analysis phase examines the question and often classifies it into a set of categories based on the expected answer type for the question. For example, "Where is the Taj Mahal" would be classified as a location question. Techniques for question analysis range from simple pattern matching of hand-written rules [19] to classification using parse trees [17].

There are a few basic approaches to document retrieval for QA and passage extraction. Some systems perform both steps at once by doing passage retrieval. Passages are typically taken as overlapping windows of terms or bytes or a sequence of sentences. Ranking algorithms for documents or passages are either based on traditional information retrieval scores or measures of term proximity [4]. It is also common to index text identifiers of text matching answer types so that they can be included as important components of the query [16]. Queries to the retrieval engine tend to be simple text queries [4] or Boolean queries [14]. Passage extraction is usually based around the tightest cluster of query terms [4].

Answer extraction takes passages and the output of the question analysis as input and produces a ranking of answers (or one single best answer). Answer extraction tends to be the most knowledge intensive component of QA systems. The level of knowledge encoded may vary from system to system; some systems use extract answers using simple templates, while others attempt to prove the answer is correct using a knowledge base and linguistic analysis of the passage [14].

2.3 Modern Hybrid QA

Among the more knowledge intensive extraction-based techniques the works of LCC, Keselj, and van Durme. LCC [14] uses a theorem prover that leverages linguistic knowledge to verify extracted answers. Keselj demonstrated the use of unification of head-driven phrase structure grammars of processed text and queries [10]. Van Durme [19] proposed the use of light semantic processing, fuzzy unification, and the computation similarity score for the quality of the unification.

State-of-the art QA systems use a hybrid of extraction-based and knowledge-based methods. Multiple techniques are used with varying levels of knowledge and then the extracted answers from the different approaches are combined to give a single ranking. These approaches may use existing knowledge bases or build them using information extraction techniques. For example, Jijkoun et al. [8] combine a knowledge base with extracted factual information with template based approaches, n-gram matches, and a full question answering system. The knowledge base consists of tables of factual information extracted from the corpus targeted to answering specific question types. Lin and Katz [11] provide access to knowledge bases on the web by providing wrappers so that object-property-value database queries can be issued. Echihabi et al. [6] combine knowledge-based approaches with pattern-based and statistical methods.

Of particular note is the work of Litkowski [13][12]. Litkowski has been developing an XML-based QA system. The markup employed is both syntactic and semantic in nature. Litkowski first parses passages to produce parse trees which are then processed to annotate them with semantic role and relationship information. Question answering is performed by creating an appropriate XPath query to search the passages. The XPath queries are created using a rule-based mechanism, which can be manually intensive. In Litkowski's evaluations, it is observed that while many answers are found for questions, they are not always ranked highly. This stresses the need for structured retrieval systems to rank results and gracefully back off when all desired structural constraints are not satisfied. We argue that this is a marked contrast to the query-relaxation approach which provide sets of unranked (or poorly ranked) results, which is typical for the retrieval component of many QA systems.

While this paper's focus is on extraction-based question answering tasks, it is important to note that modern QA research is moving beyond this task. Current research in QA also includes work on definition questions which require selection of valuable snippets. This work is more complex as it may require novelty detection and summarization approaches to produce good answers. This work is beyond the scope of this paper, and would have further additional requirements of a structured retrieval system. This paper's purpose is to start the process of examining additional requirements of structured retrieval systems, and is by no means exhaustive.

3. RICHER QUERIES FOR RICHER DOCUMENTS

In most current extraction-based QA, there is little interaction between answer extraction components and retrieval components. With a richer index that stores the output of some system preprocessing, a greater degree of interaction between extraction and retrieval would be enabled. We view this as trying to treat the text retrieval system more like a knowledge base, even though it may be a small step in that direction. However, the goal is still extraction-based QA.

In order to enable a greater interaction between answer extraction and retrieval, we may need to index syntactic and semantic structure contained within the documents. Typically, knowledge bases are assumed perfect, but the automatic document markup used in a QA system cannot be assumed perfect. Knowledge encoded in text may not be accurate, and the natural language processing methods used will not provide perfect results. This has two primary implications to this approach:

- It is appropriate to provide rankings of results, rather than a set. The order of the ranking should reflect confidence of the passage matching the query. Effective retrieval should rank the most exactly matching results highest, but gracefully back off to approximate matches.
- Additional processing or reasoning over text or using external knowledge bases will still be appropriate. A system may use an external resource containing knowledge about the world or language to verify answers. It may also be too computationally expensive to do all desired forms of annotation of the text prior to indexing and retrieval.

The main idea is that by increasing interaction between the extraction and retrieval components, more specific result lists can be passed to the extraction component. Returning passages that are more likely to satisfy the answer extraction component can enable some benefits:

- The extraction component then has fewer results to process, therefore decreasing processing time during the extraction phase.
- The extraction component can process more results, possibly resulting in higher recall.

In order to gain these benefits, the retrieval system must index documents with structure. Some of the options for structural markup include:

- *document structure markup - paragraphs, sections, etc.* These forms of structure are common in XML documents and can be added at low cost to flat text documents using machine learning algorithms.
- *named entities.* Named entities such as people names, companies, locations, and dates are extensively used in QA and there are established approaches for extracting them.
- *part-of-speech.* Part-of-speech taggers are somewhat expensive to run but are also commonly used by answer extraction components.
- *parse trees.* Parse trees are very expensive, but provide rich information about the relationships of words in a sentence. Parse trees are used by some QA systems.
- *semantic role labels.* Semantic roles are arguably more useful than parse trees, as they can concisely represent the relationship between objects in sentences. This means semantic roles can apply to more general situations than parse trees.
- *extracted knowledge.* This knowledge is similar to semantic role labels in that it represents knowledge about the semantics of the sentence. However, it is much more specific to specific kinds of knowledge and may be tailored to answering specific question types.

The structure indexed in the retrieval system can have large effects on the types of queries that may be appropriate, as we will see in the example below.

3.1 Example Query

In this section we examine the different types of queries that could be asked of an XML search system for the question:

```
Who killed Abraham Lincoln?
```

In this example we will start with simple queries typical for retrieval components of QA systems and build up to more complex queries that could be issued to more structured retrieval systems. The example queries presented here do not conform to any specific query language in order to keep the notation simple. Later in the paper we will examine how these queries may be represented in an XML retrieval system and how corresponding text in documents that match the queries could be represented. Note that we never assume the system must match the structural requirements in the query exactly, but should instead rank highly those that match the query best.

Most basic information retrieval systems for QA would search for a simple query such as:

```
Q1:
kill Abraham Lincoln
```

In the above query, "killed" has had its suffix removed, a common practice in Information Retrieval. Question answering systems may or may not remove suffixes; this distinction is not important here. A simple marked-up document may have sentence boundaries:

```
Q2:
sentence(kill Abraham Lincoln)
```

With the annotation of named entities in the corpus, a search may look like:

```
Q3:
sentence(kill person='Abraham Lincoln' person=?)
```

This search would look for the word kill in sentences that have a component of the person type that matched 'Abraham Lincoln' and contained another person component. The second person component in the query is there to indicate that another person should be present in the results, but we do not know who the person is. It is likely that the extraction component would extract that component matching "person=?" as the answer.

Adding part of speech information allows a simple constraint on the verb:

```
Q4:
sentence(kill[POS=VBD] person='abraham lincoln'
         person=?)
```

The notation "kill[POS=VBD]" may confuse some readers familiar with XPath, as this would match an XML component "<kill POS='VBD'/>". We do not wish to imply the XML document is marked up in this manner, we just borrowed the XPath notation here and applied it to attributes of terms. We will discuss in the next section queries and document structures that more closely match XPath and XML notations.

Using a semantic tagger on the corpus would allow even richer queries:

```
Q5:
sentence(event(kill[POS=VBD])
         patient(person='Abraham Lincoln')
         agent(person=?))
```

The usefulness of semantic role labeling should be very apparent with this query. Enforcing that Abraham Lincoln be the person that is killed in the sentence and ensuring that there also be a person that did the killing would give us high confidence that the sentence contains a good candidate answer. Even without the named entity tagging and part-of-speech tagging, semantic role labeling would be powerful. Consider:

```
Q6:
sentence(event(kill) patient(Abraham Lincoln)
         agent=?)
```

Document components satisfying this query would still provide good candidate answers.

Despite all of the nice aspects of query Q5, it still requires that the verb be 'kill' and that the named entity match "Abraham Lincoln". It would be nice to represent a set of words or a concept within a query. For example, the kill concept could be represented as a multinomial probability distribution:

```
Q7:
sentence(
  event(#model(0.4 kill 0.3 assassinate
               0.2 murder 0.1 shoot)[POS=VBD])
  patient(person='Abraham Lincoln')
  agent(person=?))
```

We use ♯ here to denote a query operator rather than an element type. Note that it is not uncommon for information retrieval systems to provide some form of query term weighting. Query weighting is very useful for automatic term expansion. The probability distribution for 'kill' in query Q7 could be constructed from outside resources such as WordNet or a thesaurus. We may additionally wish to expand on variants of 'Abraham Lincoln' using some outside resource.

```
Q8:
sentence(
  event(#model(0.4 kill 0.3 assassinate
               0.2 murder 0.1 shoot)[POS=VBD])
  patient(person=#model(0.4 'Abraham Lincoln'
                        0.4 'President Lincoln'
                        0.1 'honest Abe'
                        0.1 'Lincoln'))
  agent(person=?))
```

Representation of queries as in Q8 has several strengths. Flexibility on terms matching is allowed through the use of weighted query components. The term weighting allows the use of outside resources to incorporate world or domain knowledge to improve recall. The semantic role labels ensure that the sentence will have a good candidate answer, which the answer extraction component may verify using external resources. The use of part-of-speech and named-entities provides redundancy over the semantic role labels and may allow the system to still find a good candidate answer where the semantic role labeler failed to process the sentence properly.

On the other hand query Q8 is not likely a realistic query with today's technologies. Performing all of this processing prior to indexing would be very costly and probably impractical. Queries Q1 - Q3 all are realistic examples as many QA systems already index features of this kind. The use of term weighting in queries Q7 and Q8 is also practical, as this does not require additional processing for indexing. This processing would be done at query analysis time. Perhaps the most sophisticated realistic query would look like:

```
Q9:
sentence(#model(0.4 kill 0.3 assassinate
                0.2 murder 0.1 shoot)
         person=#model(0.4 'Abraham Lincoln'
                       0.4 'President Lincoln'
                       0.1 'honest Abe'
                       0.1 'Lincoln')
         person=?)
```

This query has many of the benefits of Q8, but additional text processing would still be required to ensure that the other person in the sentence did the action of killing Abraham Lincoln.

The addition of extracted knowledge in the index may take several forms. Consider the extraction of (object, property, value) triples were extracted as in [9][11]. The role of the retrieval engine would then be to match the "assassinated" property and the "Abraham Lincoln" value and returning the object:

```
Q10:
opv-triple(object=?
           property=#model(0.4 kill 0.3 assassinate
                           0.2 murder 0.1 shoot)
           value=#model(0.4 'Abraham Lincoln'
                        0.4 'President Lincoln'
                        0.1 'honest Abe'
                        0.1 'Lincoln'))
```

Many QA systems use template based methods. One template that may be generated for the query would be:

_____ killed Abraham Lincoln.

where the blank indicates the answer to be extracted. To effectively support these queries, the queries issued to the retrieval component would need to support sequences of terms. Some queries for the above template might look like:

```
Q11:
sentence(#sequence(kill Abraham Lincoln))

Q12:
sentence('kill Abraham Lincoln')

Q13:
sentence(#sequence(person=? kill
                   person='Abraham Lincoln'))
```

Query Q11 and Q12 represent a queries for a system that has only sentence boundaries marked, while Q13 represents a query for a system with both sentence boundaries and named entities. For QA systems that use templates for candidate answer extraction, the inclusion of sequences in query representations would be quite useful. Queries could be created and issued for specific templates, and the highly ranked components in the returned component lists should match the templates well.

4. CHALLENGES FOR REPRESENTING RICHER DOCUMENTS

The rich variety of queries presented in the previous section should make the usefulness of structured documents and structured queries for QA clear. However, we left out many details in the previous section and did not discuss how these structures could be represented in XML documents. In this section, we reanalyze the queries with respect to document components. We provide example document components in XML that would match the queries where possible, and discuss problems arising from alternative query and document representations.

Query Q1 is a standard flat text query, so any document component containing all three query terms would be ranked highly. Some examples of text that would match query Q2 are:

```
S1:
<sentence>
  John Wilkes Booth killed Abraham Lincoln.
</sentence>

S2:
<sentence>
  During the Black Hawk War, Captain Abraham
  Lincoln was forced to kill Native Americans.
</sentence>
```

Sentence S1 contains the desired answer, while S2, which also matches the query well, does not. The use of named entities in Q3 would eliminate this problem, as S2 only contains one person. Some text matching Q3 is:

```
S3:
<sentence>
  <person> John Wilkes Booth </person>
  killed
  <person> Abraham Lincoln </person> .
</sentence>
```

While the part-of-speech attribute in Q4 may be simple to represent in a manner similar constraints in XPath, this complicates how the text must be represented in the document. As individual terms cannot have attributes, kill must be represented as a document component. For instance:

```
S4:
<sentence>
  <person> John Wilkes Booth </person>
  <term POS='VBD'> killed </term>
  <person> Abraham Lincoln </person> .
</sentence>
```

In turn, a more accurate query would be:

```
Q14:
sentence(term[POS='VBD']='kill'
         person='abraham lincoln'
         person=?)
```

However, it is likely that part-of-speech tagging would be performed on every term, yielding a structure more like:

```
S5:
<sentence>
  <person>
    <term POS='NNP'> John </term>
    <term POS='NNP'> Wilkes </term>
    <term POS='NNP'> Booth </term>
  </person>
  <term POS='VBD'> killed </term>
  <person>
    <term POS='NNP'> Abraham </term>
    <term POS='NNP'> Lincoln </term>
  </person> .
</sentence>
```

Which then raises questions about how to represent person='abraham lincoln' in the query:

```
Q15:
sentence(term[POS='VBD']='kill'
         person(term='abraham' term='lincoln')
         person=?)
```

Already with just three types of simple markup we see complications arising in the document and query representations. However, the following structure appears at the first examination to be adequate for text matching queries Q6-Q13:

```
S6:
<sentence>
  <agent>
    <person>
      <term POS='NNP'> John </term>
      <term POS='NNP'> Wilkes </term>
      <term POS='NNP'> Booth </term>
    </person>
  </agent>
  <event>
    <term POS='VBD'> killed </term>
  </event>
  <patient>
    <person>
      <term POS='NNP'> Abraham </term>
      <term POS='NNP'> Lincoln </term>
    </person>
  </patient> .
</sentence>
```

Unfortunately, this structure is not adequate. The named-entity tagging and the semantic role labeling may be done by separate components and may give incompatible results. Consider the case where there may be errors in the semantic role labeler:

```
S7:
<sentence>
  <agent> John Wilkes </agent> Booth
  <event> killed </event>
  <patient> Abraham Lincoln </patient>
</sentence>

S8:
<sentence>
  <person> John Wilkes Booth </person>
  killed
  <person> Abraham Lincoln </person>
</sentence>
```

When merging the results of the role labeler in S7 and the results of the named-entity tagger in S8, we are faced with a problem. We may decide that the agent is contained within the person, yielding:

```
S9:
<sentence>
  <person>
    <agent> John Wilkes </agent>
    Booth
  </person>
```

```
  <event> killed </event>
  <person> Abraham Lincoln </person>
</sentence>
```

This will no longer match the query Q6 as well, due to the reversed order of the structure. The could also lead to an inconsistency of ordering component types in the hierarchy, and possibly violate a DTD. These are not the only problems that could arise in this situation. Suppose the named-entity tagger also produces errors:

```
S10:
<sentence>
  John <person> Wilkes Booth </person>
  killed
  <person> Abraham Lincoln </person>
</sentence>
```

Any attempts to merge S7 and S10 will have the problem that this can no longer be represented in XML using the simple hierarchical structure we have been using. One approach to deal with this problem would be to try to enforce a DTD when merging the output of different processing components. This would be effectively second-guessing the output of natural-language processors that have been optimized for their task, and any attempt to do this is likely to be heuristic at best.

Another approach would be to index the output of the processors separately at some level where there is likely to be no conflicts of this type, such as at the sentence level. This has the problem that we no longer can ask questions about the relationships between the different natural-language processes. A query on these documents might be:

```
Q16:
sentence(patient(Abraham Lincoln)
         person='Abraham Lincoln'
         event(kill)
         kill[POS='VBD']
         person=?
         agent=?)
```

Due to the loss of being able to identify that the person 'Abraham Lincoln' is the same as the patient 'Abraham Lincoln' is very limiting and we have no guarantees that the relationships we desire are preserved. This also would have the text duplicated by every processing component, increasing the collection size dramatically.

The problem of representing different hierarchies of markup can also be highlighted by considering different parses of a sentence. Ambiguous sentences may have multiple correct parses, and it is desirable to have the capability to index them. These parse trees will have overlapping components, and a single hierarchy no longer makes sense for a document. A parser may output multiple parses of sentences with a confidence value associated with each. Ideally, a system would be capable of indexing these alternatives and weighting them accordingly during retrieval.

4.1 Summary of Challenges

Before we present an alternative XML representation, we first summarize the previous discussion of using XML for QA. This section and the previous section has highlighted several challenges for XML systems:

1. Queries represent information needs and results should be ranked according to relevance.
2. There is a need to represent term weighting in XML queries.
3. Order and proximity are important for QA and this should be reflected in queries and indexing.
4. There are multiple overlapping hierarchies of structure for text that we would like to index.
5. It is important to be able to express relationships between components in these hierarchies.
6. The relationships between hierarchies may not be exact due to errors in language processing so approximate matching on structure and terms is important and should be reflected in the result rankings.

5. POTENTIAL SOLUTIONS

The challenges presented by trying to apply XML to QA are not insurmountable. However, they do require a different view of how to represent structure in XML documents. The single hierarchy solution is not appropriate for all tasks. Additionally, it is important to recognize the role of proximity, order, weighted query terms, and approximate matching of structure.

The first challenge of approximate matching and ranking according to relevance is a very important challenge. This has already been recognized by many XML researchers, and INEX is providing an evaluation forum for the understanding of this problem. They have also developed a query language called NEXI based on XPath for XML retrieval.

Challenges 2 and 3 are perhaps the easiest to accommodate within XML databases. Existing query languages can be expanded to include term weighting. Some query languages support only strict matches, and the incorporation of query term weighting is one way to provide feedback to the system about relative importance in rankings. Order and proximity can also be added to existing query languages. It is important to note here that XPath does have the ability to place some constraints on order. Order does have an impact on the indexing of XML documents. Term and document component locations are not indexed by all XML systems, but they are important for some retrieval tasks. Queries with phrases, sequential constraints, and proximity weights in the ranking all require term locations to be indexed.

The last three challenges are all related and there is at least one solution that addresses these concerns in a unified manner. Offset annotation can be used to represent the alternative and perhaps overlapping structures for text. In offset annotation, an index of structure separate from the original text of the document is created that contains cross-references to the original text. The cross-references are called offsets, and may be in terms of byte locations or token positions. To keep our example simple, we will use token positions and assume the original text has been tokenized in a uniform manner for all processing components. In this case, we will place each token on a separate line, with the term location in parentheses:

```
(1) John
(2) Wilkes
(3) Booth
(4) killed
(5) Abraham
(6) Lincoln
```

In practice it is much more likely to use byte offset annotation for a QA system and preserve the original text, as many pattern based techniques leverage punctuation. An example of an annotation for this text is:

```
<named-entities>
  <person begin='1' length='3'/>
  <person begin='5' length='2'/>
</named-entities>
<parse-trees>
  <sentence begin='1' length='6'>
    <noun-phrase begin='1' length='3'/>
    <verb-phrase begin='4' length='6'>
      <noun-phrase begin='5' length='2'/>
    </verb-phrase>
  </sentence>
</parse-trees>
<semantic-roles>
  <predicate begin='1' end='6'>
    <event begin='4' length='1'/>
    <agent begin='1' length='3'/>
    <patient begin='5' length='2'/>
  </predicate>
</semantic-roles>
<pos-labels>
  <NNP begin='1' length='1'/>
  <NNP begin='2' length='1'/>
  <NNP begin='3' length='1'/>
  <VBD begin='4' length='1'/>
  <NNP begin='5' length='1'/>
  <NNP begin='6' length='1'/>
</pos-labels>
<opv-triples>
  <opv-triple>
    <object begin='1' length='3'/>
    <property begin='4' length='1'/>
    <value begin='5' length='6'/>
  </opv-triple>
</opv-triples>
```

The example above is somewhat more verbose than it would need to be in practice. This annotation structure allows for natural representation of the different hierarchies of the sentence. It also allows for overlapping structures and alternative structures. For example, a sentence with multiple parse trees could simply have two sentence entries in the parse-trees component.

The importance of multiple annotations cannot be stressed enough. In order for a structured retrieval system to be flexible for a number of tasks, it must be capable of supporting arbitrary structures. In limited natural language tasks, it may be possible to impose a single strict hierarchy on the structure and embed it into the text. Even in these simple cases, this modifies the original text which may be undesirable. We saw in the examples above some complications of this in a very simple sentence. Language is ambiguous and multiple correct parses are not uncommon. Even if a sentence is not ambiguous, natural language processing tools are not perfect. It is not desirable to lose the structures represented in an alternative parse, as the alternative may be the correct parse. In other cases, one may wish to work with the output of multiple parsers and index these structures.

Offset annotation is a viable way to represent these structures. It can cleanly accommodate multiple parses of a sentence. It can also cleanly represent and separate different kinds of structure, such as extracted knowledge, syntactic structures, semantic role labels, and named entities.

Representing structure using offset annotation would have implications on the query language. XPath and NEXI would not satisfy the needs for querying this document and its structure in a simple manner. In particular, a useful query language would still look similar those in the example queries given above. The query language should have the power to suggest relationships between document components across types. That is, it should be able to simply express that a person named-entity should be contained within an agent. It should also be easy to refer to the text of the original document in a simple manner. We don't want to have to look up the begin and length of a component, then cross-reference that with the text of the document. This should be a simple and intuitive mechanism, as it will be performed often.

While query languages that resemble XPath and NEXI do not allow the desired simple expression discussed above, query languages that resemble these languages are not out of the question. The semantics of some of the underlying notations may be changed to refer to their natural analogs in this setting. Hierarchical relationships expressed between components in the query should be treated as suggestions. Explicit nestings found in the structural annotations should be preferred, and other hierarchical relationships should be given a score based on how much of the 'child' is contained by the 'parent'.

The 'contains' and 'about' operators used in XPath and NEXI should automatically look up the contents of the original text. Additionally, a 'weight' operator could be included as a term weighting strategy. For example, a realistic rewrite of Q8 might be:

```
Q15:
//sentence[
  .//event//VBD[weight(0.4 kill 0.3 assassinate
                       0.2 murder 0.1 shoot)]
  AND
  .//patient//person[weight(0.4 'Abraham Lincoln'
                            0.4 'President Lincoln'
                            0.1 'honest Abe'
                            0.1 Lincoln)]
  AND
  .//agent//person]
```

Loose interpretation of the structural requirements and the 'AND' clauses would allow for reasonable rankings of document components in such an XML retrieval system supporting QA. In this case, we could even have the system extract the desired answer by moving the 'agent//person' component out of the conjunction:

```
Q16:
//sentence[
  .//event//VBD[weight(0.4 kill 0.3 assassinate
                       0.2 murder 0.1 shoot)]
  AND
  .//patient//person[weight(0.4 'Abraham Lincoln'
                            0.4 'President Lincoln'
                            0.1 'honest Abe'
                            0.1 Lincoln)]
  ]//agent//person
```

The above example illustrates how queries and documents could be represented within XML for the assistance of extraction-based QA. These representations will impact the index structures used within databases to optimize for retrieval, but we will not go into details here, as optimizing indexing structures is an area of ongoing research.

Returning to the first challenge, ranking components according to how well they match the queries could be done in many ways. For example, language modeling and generative probability distributions [15] could be easily adapted to this context. Matching the 'weight' and 'about' clauses could be done with a simple language model comparison, and the loose structural requirements for the containment relationships could be ranked by estimating probabilities using observation of the document structure.

6. CONCLUSIONS

This paper examined the use of XML databases for assisting extraction-based QA systems. We argued that XML retrieval could allow for a tighter integration of the answer extraction and passage retrieval components of typical QA systems. We also looked forward beyond what is currently done most QA systems and presented a vision of QA that is closer to treating an open-domain corpus as a knowledge base. While everything we proposed is not practical now, there are clear steps that could be taken toward that goal.

The process of examining XML databases and QA together presented some challenges for XML systems. Ranking is important for XML retrieval, as information needs may be approximate and vague. It is important to have mechanisms for term weighting and proximity operations in queries. Finally, the structures of real documents may be overlapping and messy, so mechanisms for indexing and retrieval should be adapted to work with these idiosyncrasies. We feel that this process was informative, and should be carried out with other applications in mind.

We also proposed alternative representations for the documents and queries in order to neatly support and represent them for QA related tasks. These different representations will require adaptations to the indexing techniques and query processing techniques. We believe that the use of generative probability distributions is one effective way to rank document components for structured queries.

We feel the it is important to examine a wide variety of problems that can be addressed by structured retrieval systems. In order to best understand the needs of structured retrieval, it is necessary to look beyond simple ad-hoc retrieval tasks. This paper examined the application of structured databases to QA, and it resulted in some challenges for XML retrieval. This process should be carried out for additional retrieval tasks, as they will highlight additional challenges.

7. ACKNOWLEDGMENTS

The author would like to thank Benjamin Van Durme for his thoughtful discussions and comments on the paper. This research was sponsored by National Science Foundation (NSF) grant no. CCR-0122581. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implicit, of the NSF or the US government.

8. REFERENCES

[1] XML path language (XPath). Technical report. J. Clark and S. DeRose, editors. http://www.w3.org/TR/xpath.

[2] XQuery 1.0: An xml query language. Technical report. S. Boag, D. Chamberlin, M. Fernández, D. Florescu, J. Robie, and J. Siméon, editors. http://www.w3.org/TR/xquery/.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] C. Clarke and E. Terra. Passage retrieval vs. document retrieval for factoid question answering. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003.

[5] A. Deutsch, M. Fernàndez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. In *WWW The Query Language Workshop (QL)*, Cambridge, MA, 1998.

[6] A. Echihabi, U. Hermjakob, E. Hovy, D. Marcu, E. Melz, and D. Ravichandran. Multiple-engine question answering in textmap. In *The Twelfth Text REtrieval Conf.*

[7] N. Fuhr, N. Goevert, G. Kazai, and M. Lalmas, editors. *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)*. ERCIM, 2003.

[8] V. Jijkoun, G. Mishne, C. Monz, M. de Rijke, S. Schlobach, and O. Tsur. The university of amsterdam at the trec 2003 question answering track. In *The Twelfth Text REtrieval Conf.*

[9] B. Katz. From sentence processing to information access on the World Wide Web. In *AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, 1997.

[10] V. Keselj. Question answering using unification-based grammar. *Advances in Artificial Intelligence, AI 2001*, LNAI 2056, 2001.

[11] J. Lin and B. Katz. Question answering from the web using knowledge annotation and knowledge mining techniques. In *The Twelfth International Conference on Information and Knowledge Management*, pages 116–123, 2003.

[12] K. C. Litkowski. Use of metadata for question answering and novelty tasks. In *The Twelfth Text REtrieval Conf.*

[13] K. C. Litkowski. Question answering using xml-tagged documents. In *The Eleventh Text REtrieval Conf. (TREC-11), NIST SP 500-251*, pages 156–165, 2003.

[14] D. Moldovan, M. Pasca, S. Harabagiu, and M. Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Trans. Inf. Syst.*, 21(2):133–154, 2003.

[15] P. Ogilvie and J. P. Callan. Language models and structured document retrieval. In *Proc. of the First Annual Workshop of the Initiative for the Evaluation of XML retrieval (INEX)*, DELOS workshop, Dagstuhl, Germany, Dec. 2002. ERCIM.

[16] J. Prager, D. Radev, E. Brown, A. Coden, and V. Samn. The use of predictive annotation for question answering in trec 8. In *Proceedings of the Eighth Text REtrieval Conference*, 2000.

[17] G. Ramakrishnan, S. Chakrabarti, D. Paranjpe, and P. Bhattacharyya. Is question answering an acquired skill? In *Proceedings of WWW2004*, 2004.

[18] A. Trotman and B. Sigurbjörnsson. Narrow Extended XPath I. Technical report. Available at http://inex.is.informatik.uni-duisburg.de:2004/.

[19] B. Van Durme, Y. Huang, A. Kupsc, and E. Nyberg. Towards light semantic processing for question answering. Edmonton, Cananda, May 31 2003. HLT/NAACL Workshop on Text Meaning.

[20] W. A. Woods. Lunar rocks in natural English: Explorations in natural language question answering. *Linguistic Structures Processing*, 1977.