

# A Generative Retrieval Model for Structured Documents

Le Zhao and Jamie Callan  
Language Technologies Institute  
School of Computer Science  
Carnegie Mellon University  
{ lezhao, callan }@cs.cmu.edu

## ABSTRACT

Structured documents contain elements defined by the author(s) and annotations assigned by other people or processes. Structured documents pose challenges for probabilistic retrieval models when there are mismatches between the structured query and the actual structure in a relevant document or erroneous structure introduced by an annotator. This paper makes three contributions. First, a new generative retrieval model is proposed to deal with the mismatch problem. This new model extends the basic keyword language model by treating structure as hidden variable during the generation process. Second, variations of the model are compared. Third, term-level and structure-level smoothing strategies are studied. Evaluation was conducted with INEX XML retrieval and question-answering retrieval tasks. Experimental results indicate that the optimal structured retrieval model is task dependent, two-level Dirichlet smoothing significantly outperforms two-level Jelinek-Mercer smoothing, and with accurate structured queries, the proposed structured retrieval model outperforms keyword retrieval significantly, on both QA and INEX datasets.

## Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: *Retrieval Models, Query formulation*

## General Terms

Algorithms, Experimentation, Theory

## Keywords

Structured Retrieval, Generative Model, Indri Query Language, Language Model, XML Retrieval, Question Answering

## 1. INTRODUCTION

From the earliest days of information retrieval, search systems have used field-based retrieval to confine a search to just a portion of a document's text or metadata, for example, the title, author, source, or publication date. In recent years, the widespread use of HTML and XML, and the increasing use of text annotations such parts-of-speech, named-entities, and semantic role labels for some applications have prompted renewed interest in document structure. Two well-known examples are retrieval of texts or text elements expressed in XML [1], and retrieval of texts or text

elements for question answering [3, 8]. These two examples are not necessarily distinct, but typically XML retrieval focuses on structure introduced by the document author(s), whereas question answering focuses on structure introduced by annotators. These two types of structure typically have different characteristics, for example, differing size, scope, and reliability. One very important difference is that the elements of an XML document must be nested strictly, whereas annotations have no such restriction, thus we frame the problem in terms of *fields*, not XML elements.

Document fields are specified by embedded and/or offset ("standoff") annotations. A field is defined by a unique id, a field type, a starting position, and an ending position. Fields may have hierarchical relationships. The document is considered a field – the outermost field. The search process ranks and returns field types specified by a query, for example, documents, titles, sections, sentences, or targets (a type of semantic role). A query can consider evidence from a contained (inner) or descendent field when evaluating a containing (outer) or ancestor field [1, 8].

The traditional approach to field-based retrieval considers only the text within specified field types (an *exact-match* constraint). However, approximate matching is often preferred because users may not have a complete understanding of how structure is used in documents ("imprecise queries"), or the document structure may not be correct in all documents. The former type of mismatch applies to both author-written fields (e.g., title, section) and annotator-generated fields (e.g., semantic role labels). The latter type of mismatch applies primarily to automatically-generated fields. Both types of mismatch are common. Some retrieval models for XML documents use smoothing to obtain approximate matching (e.g., [14]), but this approach is less common.

When a retrieval model allows evidence from a contained (inner) field to be used when evaluating a containing (outer) field, it must specify how to combine evidence when multiple contained fields match; this is particularly true if approximate matching is permitted. For example, when ranking sentences for a question answering application, a long sentence may contain two spans annotated as *subject*. If approximate matching is permitted, both match to some degree. The query or retrieval model must specify how this evidence is combined to produce a score for the containing sentence.

This paper provides a principled solution to the problems of approximate matching and combination of evidence described above. We start with the Indri search engine's basic term-based language model [9], and extend it to provide a generative structured retrieval model. We investigate the related problem of hierarchical smoothing methods that incorporate document level evidence, and smoothing methods for combining evidence from fields of different lengths. Our work provides a principled method of doing approximate matching, and an empirical study of methods for combining evidence from inner fields.

We begin with an introduction to the language modeling retrieval framework and the current field retrieval and structured

retrieval models of the Indri search engine [8, 9]. Then, we focus on extending the field retrieval model into a new approximate structured retrieval model. A connection between the new structured retrieval model and the traditional term based language model is revealed. Sample structured queries are used to demonstrate the effectiveness of the new model. In the experiments, two applications, question answering and XML element retrieval, are used to demonstrate the performance of the new model with new smoothing methods. Finally related work is discussed, and conclusions follow.

## 2. EXISTING WORK

We start by introducing the language modeling framework for information retrieval, and the Indri search engine’s current field retrieval and structured retrieval capabilities [13, 8]. These lay the foundation for the remainder of the paper.

### 2.1 Language modeling

In its very basic form, the language modeling approach to IR involves computing the generation probability of a query given a document language model [5]. Query terms are assumed to be independent of each other. The document model is smoothed with the collection model, for example using Jelinek-Mercer or Dirichlet smoothing methods [6].

$$P(q | D) = \prod_{i=1}^{|q|} P_s(q_i | D) \quad (2.1)$$

$$P_s(q_i | D) = \frac{\text{tf}(q_i, D) + \mu P(q_i | C)}{\text{length}(D) + \mu} \quad (2.2)$$

Equations 2.1 and 2.2 show the basic language model with Dirichlet smoothing.  $P_s(q_i | D)$  is the smoothed probability of document  $D$  generating query term  $q_i$  and  $\mu$  is the scale parameter for Dirichlet smoothing. Indri uses language models and an inference network model for retrieval and scoring [12, 13]. Inference networks typically implement query operators that dynamically produce new index terms (e.g., `#1(wipe out)`; `#syn(destroy #1(wipe out))`) or combine scores produced by index terms or other query operators, for example, `#combine( structured #1(information retrieval) )` or `#max( #combine(information retrieval) IR )`.

### 2.2 Field retrieval

Indri’s default field retrieval model directly extends the traditional term-based language model by using the probability that the desired field – instead of the whole document – generated the query, and returning the queried fields, instead of documents, as results. Equation 2.3 is used instead, where  $q$  is the query,  $F$  is the desired field type,  $D$  is a document containing  $F$ , and  $P_s(q_i | F, D)$  is the smoothed probability for the field to generate query term  $q_i$ .

$$P(q | F) = \prod_{i=1}^{|q|} P_s(q_i | F, D) \quad (2.3)$$

The major difference between document retrieval and field retrieval is that the surrounding context of a query term shrinks from a document to a field. Thus, term frequency becomes occurrences of the term within the field, and for each query term, the background smoothing probability comes from not only the collection but also the document containing the field. Including the document level smoothing is shown to be effective in INEX XML retrieval [14]. Two level smoothing is achieved by interpolating the field  $F_j$  model with both the document and the

collection models. Equation 2.4 shows two level Jelinek-Mercer smoothing, in which  $P(q_i | D)$  is the unsmoothed document model.

$$P_s(q_i | F_j, D) = \lambda_1 \frac{\text{tf}(q_i, F_j)}{\text{length}(F_j)} + \lambda_2 P(q_i | F, D) + (1 - \lambda_1 - \lambda_2) P(q_i | C) \quad (2.4)$$

where the document model comes from all  $F$  fields in  $D$ :

$$P(q_i | F, D) = \frac{\text{tf}(q_i, F | D)}{\text{length}(F | D)}$$

This two level smoothing model is a general case of and can be backed off to the single level document smoothing by equating  $\lambda_2$  in Equation 2.4 to 0. Two level Dirichlet smoothing is not present as this two level formulation is not general enough to be applied to Dirichlet smoothing. We return to this point in Section 3.

In the Indri query language, a field query is expressed as `#combine[F]( q1 q2 ... qn )`, for example `#combine[sentence]( take measures )`, where `[sentence]` restricts the target scoring field to be `sentence` fields and the query terms are `take` and `measures`. The `#combine` (probabilistic AND) operator multiplies the generation probabilities from `take` and `measures` to produce the final generation probability for a sentence. For a graphical model representation of this example query, see Table 1.

### 2.3 Going Structural

In Indri, XML elements and text annotations are treated uniformly as *extents* – spans of text in a document. An *extent restriction* on a query operator constrains the result extents to be of a specific type, for example, the query `#combine[F]( q )`, where  $F$  defines the field type and  $q$  is a sub-query, evaluates `#combine( q )` on all  $F$  fields of a document. Here,  $q$  can be a complex query which may also contain its own extent restrictions, for example, `#combine[document]( #combine[section]( design usability guidelines ) user interface )`. By embedding the section extent restriction inside the document extent restriction, this query retrieves documents, but uses relevance scores from sections as evidence, and combines them to yield the final score for the document. This kind of evidence-combination queries is frequently seen in the INEX datasets [15].

When executing the above example query from INEX 2006, for each document, `#combine[section]` will return a list of scores for each section that appears in the document; the scores depend on the relevance of the section to the keyword query `#combine( design usability guidelines )`. Then, the list of section probabilities together with the relevance scores from the query terms `user` and `interface` are combined (probabilistic AND, because of the `#combine` node at the `[document]` level) to form the final score for the document.

In the above example, fields are nested in the query, which means that the text span of the inner field is a substring of the outer field’s text span; for example, the section text is part of the document text. Indri also supports fields that are related hierarchically (parent-child), but that do not satisfy a text span containment constraint. Such fields are convenient for semantic role labels, where the agent (usually `arg0`) and patient (usually `arg1`) are both linked to the target verb as their parent, even though the arguments are not contained within the target’s text span [8]. Relationships among fields are specified using “.” to indicate a child-parent relationship between an extent restriction and its immediate outer extent restriction in the query. For example,

```
#combine[document](
  #combine[target]( Loves
    #combine[./arg0]( John )
    #combine[./arg1]( Mary ) ) )
```

returns documents based on evidence from target fields. The “./” operator ensures that the matched `arg0` and `arg1` fields are actually children of the containing field in the query, i.e. the target field.

## 2.4 Smoothing

One central problem for structured retrieval is the ability to do approximate matching. Typically, when constructing the query, the user does not know for sure what the structures in a relevant result will be like. Studies also shows that the user will very likely guess wrong [16], thus soft matching is necessary. We show how it is done currently through smoothing.

Term level smoothing in language models allows a soft match of fields with query terms. For example, `#combine[section]( Symphony #combine[title]( Music ) )` returns sections containing either a `title` field with the term `Music` or a term `Symphony`<sup>1</sup>. Partial match cases are ranked lower than the full match case where the section contains a `title` field that contains the term `Music` and the section also matches `Symphony`. Because of the two level smoothing in effect, the final score also depends on the document language model (and also document length if using Dirichlet smoothing).

Besides term level smoothing, field level smoothing is also necessary when combining evidence from different types of fields. For example for the query `#combine[section]( Symphony #combine[title]( Music ) )`, if a section does not have a `title` field, without field level smoothing, using strict field matching, this section will end up having 0 probability. The field level smoothing in the version 4.5 of Indri adds an empty and 0-length extent for a field only when the field is missing. This feature allows the missing field match to back off to the background smoothing probabilities through the empty extent.

This way of doing approximate structured retrieval, instead of using the fields as hard constraints, treats the fields in a document only as evidence for relevance. An example is, “find me sections that are likely to have title fields within the section and the title is likely to be relevant to `Music`”. As defined in INEX evaluations starting from 2003, the query structure is intended to be a hint, not necessarily a strict requirement [2].

## 2.5 Problems in the existing model

Indri’s current field and structured retrieval model have two important problems that motivate our research.

### 2.5.1 Unbalanced smoothing

Length-biased term smoothing is used at the term level, for terms that appear in fields of varying lengths. For example, in the query `#combine[section]( Symphony #combine[title]( Music ) )`, `Symphony` has a context of a section, which is typically much longer than a `title`, in which the query term `Music` appears. Thus, when using Jelinek-Mercer smoothing, the difference between having and not having the word in a short extent is much larger than that in a longer extent. Equation 2.5 illustrates the problem, which follows Equation 2.4.

$$\frac{P_s(q_i | F, D, \text{tf}(q_i, F) = 1)}{P_s(q_i | F, D, \text{tf}(q_i, F) = 0)} \approx \frac{\lambda_1 \frac{1}{\text{length}(F)}}{\lambda_2 P(q_i | D) + (1 - \lambda_1 - \lambda_2) P(q_i | C)} \quad (2.5)$$

This ratio used to be the IDF (inverse document frequency) part of the retrieval model, but now emphasizes the length of the field, and prefers shorter fields proportional to the inverse of the extent length. This unexpected length bias causes the retrieval model to greatly prefer matching of the `#combine[title]( Music )` part of the query over matching of `Symphony` in a section.

For field level smoothing, as outlined in the previous subsection, the empty extent method works reasonably well for the question answering task, but lacks theoretical justification: why not introduce the empty extent even when the field exists, why not more empty extents, or why not empty extents with a non-zero length? Section 3 provides a generative framework for doing approximate structural matching, in which more intuitions will be developed to guide the use of field level smoothing.

### 2.5.2 Irregular evidence merging

Another problem that happens as we go structural is merging evidence from fields. In Indri version 4.5, the extent restriction (e.g. the `[f]` in the query `#combine[f]( q )`) returns a list of probabilities  $P(\#combine(q) | f_i)$  from matched fields, but does not merge them. For example, for the query `#combine( #combine[section]( Pop Music ) )2`, if a document contains two sections, this query first returns the list of section extents, each section associated with its generation probability of the query terms (`Pop Music`). Then the final score for the document will be the multiplication of the two section probabilities, because of the outer `#combine` operator. Since language model probabilities are almost always smaller than 1, matching *more* fields yields more multiplications, and thus a *smaller* relevance score. This effect violates the intended query semantics; we call it the *evidence merging problem*.

To overcome this problem, the user needs to consider carefully how to combine the two section probabilities in the query. One possibility is to set the document score based on the best matching section, using a `#max` query operator: `#combine( #max( #combine [section] ( Pop Music ) ) )`. This approach avoids penalizing a document for having many relevant sections, because only the maximum scoring section contributes its score to the document score. Although `#max` works well for some tasks, e.g., question answering, there is no clear justification for it. It also does not *reward* a document for having multiple matching sections.

## 3. A GENERATIVE STRUCTURED RETRIEVAL MODEL

Our goal is a coherent probabilistic model in which the problems described above are more easily analyzed and approached. This section develops improved retrieval models and smoothing methods to achieve that goal.

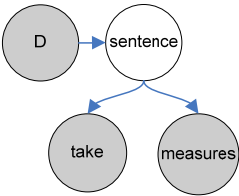

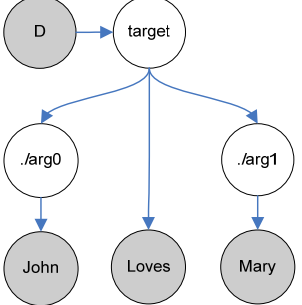
### 3.1 The model

Consider a simple canonical structured query with embedded field retrieval and using fields as evidence, as shown below.

<sup>1</sup> The section must match at least one query term to be returned by the search engine. This strategy follows the convention of most experimental IR systems, where a document is matched if and only if it contains at least one of the query terms.

<sup>2</sup> Since Indri defaults the outer-most extent to be document, we use the query `#combine( #combine [section]( ... ) )`.

**Table 1. Graphical model representations for sample structured queries. The query Q includes both the term distributions and the generation process represented in the graphical model.**

Indri Query	Probability	Graphical model
<code>#combine( #combine[sentence]( take measures ) )</code>	$P(Q D)$ ranking documents	
<code>#combine[section]( #combine[title]( Music ) )</code>	$P(Q section)$ ranking sections	
<code>#combine( #combine[target]( Loves #combine[./arg0]( John ) #combine[./arg1]( Mary ) ) )</code>	$P(Q D)$ ranking documents	

`#combine( #combine[f1]( q1 #combine[f2]( q2 ) ) )`

In this discussion it is simplest to think of  $q_1$  and  $q_2$  as single terms, but the model supports sets of terms and query operators.

The generative story is that the document needs to generate the  $f_1$  field models, and then each  $f_1$  model will generate the query term  $q_1$  and the structural term `#combine[f2]( q2 )`. Next, each  $f_2$  field model in each  $f_1$  model will generate the query term  $q_2$ . Since which  $f_1$  field model generates  $q_1$  is unknown, it is treated as a hidden variable, and summed (or rather averaged) over all the  $f_1$  models in the document. The same is true for  $f_2$  in each  $f_1$  field.

$$\begin{aligned}
 & P(\text{\#combine}(\text{\#combine}[f_1](q_1 \text{\#combine}[f_2](q_2))) | D) \\
 &= \frac{1}{\text{tf}(f_1, D)} \sum_{f_{1i} \in D} P_s(\text{\#combine}(p \text{\#combine}[g](q)) | f_{1i}, D) \\
 &= \frac{1}{\text{tf}(f_1, D)} \sum_{f_{1i} \in D} \{P_s(q_1 | f_{1i}, D) \cdot P_s(\text{\#combine}[f_2](q_2) | f_{1i}, D)\} \\
 &= \frac{1}{\text{tf}(f_1, D)} \sum_{f_{1i} \in D} \left\{ P_s(q_1 | f_{1i}, D) \cdot \frac{1}{\text{tf}(f_2, f_{1i})} \sum_{f_{2j} \in f_{1i}} P_s(q | f_{2j}, f_{1i}, D) \right\} \\
 &= \text{Avg}_{f_{1i} \in D} \left\{ P_s(q_1 | f_{1i}, D) \cdot \text{Avg}_{f_{2j} \in f_{1i}} \{ P_s(q_2 | f_{2j}, f_{1i}, D) \} \right\}
 \end{aligned} \tag{3.1}$$

In the above derivation,  $\text{tf}(f, D)$  is the number of times a field of type  $f$  occurs in document  $D$ , and each  $f_i$  is a specific instance of  $f$  in the document model  $D$ . The output probability is now  $P_s(t | f, D)$  which is obtained through merging (averaging over) the list of probabilities  $P_s(t | f_i, D)$  for  $i=1$  to  $\text{tf}(f, D)+1$ .

Table 1 presents more queries and interpretations of them in their graphical models. Because of the generative interpretation, the structured retrieval model is very different from models for flat keyword queries. In a traditional keyword query (e.g., `#combine(Music Symphony)`), the query can be seen as a

distribution of words, and the retrieval model calculates the KL divergence between the query and document distributions. In the structural case, the query does not specify a distribution of structural components. Instead, the whole query is a generative procedure that describes how the document should generate the terms in the query. Thus, to construct a structured query, instead of using an example structure or “imagining” how the terms should appear in different fields, chances are better off if the query takes into account how different relevant structures in the documents will follow the generation process specified in the query.

The structured retrieval model ranks fields by how well they satisfy the queried generation process. When the goal is to rank documents, the model combines the probabilities from the fields in the document to produce a single probability for the document.

The main contributions in this section are i) a generative framework for field-based retrieval, and ii) a specific way of merging the probabilities of multiple matching fields, through averaging, so that the resulting model has a generative probabilistic interpretation. Next, based on this generative model, we show the smoothing methods that are used to estimate the document and field language models. This constitutes the approximate matching ability of the retrieval model.

### 3.2 Term and field smoothing

Smoothing plays a central role for the approximate matching in field retrieval, and is effective for both containment and parent-child relations between fields. When queries become more complex and fields smaller, mismatches occur more often. In language modeling terms, smaller fields are more sparse, and thus are less likely to produce reliable probability estimates. Smoothing improves the estimates. Because the generative structured retrieval model generates both terms and fields, smoothing can happen at both the term level and the field level.

At the term level we propose a new two level smoothing scheme which applies to Dirichlet smoothing.

### 3.2.1 The general two level term smoothing

In order to solve the length bias problem for the Jelinek-Mercer smoothing (Section 2.5.1), the smoothing method must take into account the length of the context that a query term appears in. Dirichlet smoothing [6] has this property. However, simply smoothed with the collection model, Dirichlet is no better than the document + collection level Jelinek-Mercer smoothing [8].

We extend the two level smoothing scheme to Dirichlet smoothing by generalizing the two level smoothing procedure as follows – apply the smoothing twice, recursively, by first smoothing the document with the collection model and then smoothing the field model with the already smoothed document model.

$$P_s(q_i | F, D) = \lambda_d \frac{tf(q_i, F)}{\text{length}(F)} + (1 - \lambda_d) \{ \lambda_c P(q_i | D) + (1 - \lambda_c) P(q_i | C) \} \quad (3.2)$$

$$P_s(q_i | F, D) = \frac{tf(q_i, F) + \mu_d P_d(q_i | D, C)}{\text{length}(F) + \mu_d} = \frac{tf(q_i, F) + \mu_d \frac{tf(q_i, D) + \mu_c P(q_i | C)}{\text{length}(D) + \mu_c}}{\text{length}(F) + \mu_d} \quad (3.3)$$

Equations 3.2 and 3.3 are the two level smoothing methods corresponding to Jelinek-Mercer and Dirichlet smoothing in their general form. Both can be “backed off” to the original one level smoothing described in [6] by equating  $\lambda_c$  in Equation 3.2 to 0, and  $\mu_c$  in Equation 3.3 to +infinity. Equation 3.2 is equivalent to Equation 2.4.

Now that we have the Dirichlet smoothing working in the two level smoothing setting, it is easy to show that two level Dirichlet smoothing does not have the length bias problem, i.e. the difference between having a term match and not having a term match does not depend on the context length. For field level matching, the difference does not depend on field length:

$$\begin{aligned} & \frac{P_s(q_i | F, D, tf(q_i, F) = 1)}{P_s(q_i | F, D, tf(q_i, F) = 0)} \\ &= \frac{1 + \mu_d P_d(q_i | D, C)}{\text{length}(F) + \mu_d} \div \frac{\mu_d P_d(q_i | D, C)}{\text{length}(F) + \mu_d} \\ &= \frac{1 + \mu_d P_d(q_i | D, C)}{\mu_d P_d(q_i | D, C)} \end{aligned}$$

At the document level, if there is field level term match ( $tf(q_i, F) > 0$ ), field level match will dominate, as background smoothing probabilities from the document or collection are typically very small. If there is no field level term matching, the result is the same, i.e., the difference does not depend on the document length:

$$\begin{aligned} & \frac{P_s(q_i | F, D, tf(q_i, F) = 0, tf(q_i, D) = 1)}{P_s(q_i | F, D, tf(q_i, F) = 0, tf(q_i, D) = 0)} \\ &= \frac{1 + \mu_c P(q_i | C)}{\text{length}(D) + \mu_c} \div \frac{\mu_c P(q_i | C)}{\text{length}(D) + \mu_c} \\ &= \frac{1 + \mu_c P(q_i | C)}{\mu_c P(q_i | C)} \end{aligned}$$

In practice, we need to tune the optimal parameter for Dirichlet smoothing. Its optimal parameter value is dependent on the average length of the returned unit in order to satisfy the term discrimination constraint [7]. For sentence retrieval with question answering queries that use semantic role label structures, we find 5 for  $\mu_d$  and 50 for  $\mu_c$  to be optimal. For XML document retrieval,

where typical queried fields are article, section, and figure, the optimal  $\mu_d$  is around 100, and  $\mu_c$  around 500. We expect a field- and query-dependent smoothing to be a best Dirichlet smoothing.

### 3.2.2 Field level smoothing

At the field level, for smoothing the queried fields in a document, Dirichlet smoothing suggests the use of empty fields of the same type being added to the document model. Referring back to Equation 3.1, assume, given document model  $D$ , the probability of generating any one instance  $f_i$  of the field typed  $f$  is just the probability of randomly picking  $f_i$  out of all  $\{f_i, i=1..tf(f, D)\}$ , where the document model  $D$  also includes the empty fields for field level smoothing. Since it is empty, only the background smoothing scores will be filled in for all query terms for that field. This way, even without a queried field present, a document can be retrieved and ranked as long as there is a term match. If the number of empty fields added is 1, this field level smoothing is a Laplacian smoothing, i.e. Dirichlet smoothing with the scale parameter of the Dirichlet prior equal to 1.

Because of the effect of smoothing, a partially matched document will have a relevance score lower than the documents where field and term both match.

There are two parameters to tune for field-level smoothing. First is the Dirichlet scale parameter – how many empty fields to add; having more empty fields typically brings the field closer to the document language model. Second is the length of the empty fields; using a non-zero length will down-weight the document and collection background probabilities. We find that adding one empty field is usually enough, and the length of the empty field does not affect performance much, thus we leave it as 0.

### 3.2.3 Context-aware Extent Restriction Constraints

In order for merging evidence from fields and retrieving fields to work at the same time, each extent restriction constraint must be aware of its context. In one case, the outer most extent restriction constraint (for example, the [section] in the 2<sup>nd</sup> query of Table 1) means field retrieval, which does not need to return empty fields for smoothing, and a need to return a score for each matched field. In the other case, extent restriction constraints on any other belief operators (e.g., #combine, #max) mean using these fields as evidence, and require empty fields be added and scores merged to return one single probability for the node (e.g. the [title] in the 2<sup>nd</sup> query of Table 1).

Being context aware can also help increase retrieval speed. This is because in field retrieval, the outermost fields in the query that do not match a query term need not be scored at all, which typically cuts computational time in half or less. For example, in the 2<sup>nd</sup> query of Table 1, unmatched [section] extents of matched documents can be skipped, whereas every *title* of a matched *section* should be scored, and the scores merged to form the relevance score for the *section*.

## 3.3 Structured Retrieval as Generalized Term Retrieval

Under the language modeling framework for text retrieval, the structured retrieval model proposed above can be seen as a natural generalization of the classical term retrieval model, where there is a special type of field, let’s call it the “term” field, surrounding each term in the document. This “term” field is special in the sense they are all of length 1 always, and they cover every term in every document.

This section shows that the structured retrieval model bears a close relation with the traditional term language model. This is shown in Equation 3.4 by casting the term retrieval model into the structured retrieval framework.

### 3.3.1 The generative story – the structured retrieval version of term retrieval model

For illustrative purposes, we calculate the maximum likelihood estimate of the probability for a document  $D$  to generate a term  $t$ . To cast it into the structured retrieval model, we assume every term in the document is surrounded by a “term” field of length 1, covering exactly the term itself, and the document must generate the field “term” first before generating term  $t$ . Which “term” field generates the query term  $t$  is hidden, and thus must be summed over, i.e. the probability of generating  $t$  is the sum of generating  $t$  from each “term” field in the document. Since the number of “term” fields in a document equals the length of the document, generating any one “term” field given  $D$  has probability  $1/\text{length}(D)$ . For simplicity we also assume no smoothing, i.e. a “term” field containing the term  $t$  generates  $t$  with probability 1, otherwise, 0. The following full derivation shows how structured retrieval, through combining evidence from “term” fields, yields exactly the term retrieval model:

$$\begin{aligned}
P(t | \text{term}, D) &= \sum_{x=1}^{\text{length}(D)} P(\text{term}_x | \text{term}, D) P(t | \text{term}_x, D) \\
&= \frac{1}{\text{length}(D)} \sum_{x=1}^{\text{length}(D)} P(t | \text{term}_x, D) \\
&= \text{Avg}_{x=1}^{\text{length}(D)} \{P(t | \text{term}_x, D)\} \\
&= \frac{1}{\text{length}(D)} \sum_{i=1}^{\text{tf}(t,D)} 1 \\
&= \frac{\text{tf}(t, D)}{\text{length}(D)} \\
&= P_{\text{MLE}}(t | D)
\end{aligned} \tag{3.4}$$

From Equation 3.4, it is easy to see that the classical term language model is just a special case. The empty “term” fields for smoothing do not appear in the above derivation because smoothing is a separate issue. For example, term level Dirichlet smoothing with scale parameter  $\mu$  is equivalent to adding any background smoothing “term” fields to the document and setting  $\mu_d$  to  $+\text{inf}$ ; Jelinek-Mercer is just linear interpolation with the smoothing field. In Indri, Equation 3.4 corresponds to the query `#combine( #combine[term]( t ) )`, which is equivalent to the term retrieval query `#combine( t )`. For the case of two query terms such as `#combine( u v )`, the corresponding structured query is `#combine( #combine[term]( u ) #combine[term]( v ) )`, where the outside `#combine` operator multiplies the generation probabilities of  $u$  and  $v$ .

## 3.4 More-Complex Structures

For more complex structures, such as:

```
#combine(
  #combine[target]( loves
    #combine[./arg0]( John )
    #combine[./arg1]( Mary ) ) )
```

the document generates the target fields, which in turn generates the children `arg0` and `arg1` fields, which at last generates the words. The graphical model diagram in Table 1 describes the model corresponding to the query. See [8] for more motivated

examples of queries that use semantic roles. During the evaluation of this query, the relevance scores are calculated and propagated from the leaf nodes to the top level recursively. Within each document, the scores for the target fields are accumulated, within each target field, the children `arg0` and children `arg1` scores are accumulated, and within each children field, the scores from the query terms are accumulated.

Suppose the document model  $D$  has  $k$  [target] extents; the  $i^{\text{th}}$  [target] extent model contains  $l_i$  [arg0] extents and  $m_i$  [arg1] extents. Let  $Q_i$  be the [target] extent restricted query: “`#combine( loves #combine[./arg0]( John ) #combine[./arg1]( Mary ) )`”,  $Q_{a0}$  = “`#combine( John )`”, and  $Q_{a1}$  = “`#combine( Mary )`”. The above query  $Q$  yields the following score when evaluated on  $D$ :

$$P(Q | D) = \frac{1}{k} \sum_{i=1}^k P_s(Q_i | t_i) \tag{3.5}$$

where

$$P_s(Q_i | t_i) = \left\{ \frac{1}{l_i} \sum_{j=1}^{l_i} P_s(Q_{a0} | a^0_j) \right\} \cdot \left\{ \frac{1}{m_i} \sum_{j=1}^{m_i} P_s(Q_{a1} | a^1_j) \right\} \cdot P_s(\text{love} | t_i) \tag{3.6}$$

## 3.5 Alternative models for belief combination

The task of the models described here is to combine scores from multiple fields to generate a single score for the outer containing field, e.g. in Table 2, collapsing the scores from `[./arg1]`’s for each [target], and collapsing scores from the [target] fields for the containing sentence. In the degenerate term retrieval model in Equation 3.4, collapsing is done by the average of relevance scores from the [term] fields.

The generative model in Equation 3.4 generally just takes in a list of probabilities from the matching fields and outputs a single score – the average – for the list. We term this process *belief combination*. There are other ways of collapsing the list of probabilities in generating this single score. We consider three cases: i) average (AVG), ii) the maximum of the list (MAX), and iii) Probabilistic OR (OR). It should be noted that only AVG has a direct connection to the term based language model (Section 3.3.1). For very small values of  $P_i$ , the Probabilistic OR  $1 - \prod_{i=1}^n (1 - P_i)$  which equals  $1 - (1 - \sum_{i=1}^n P_i + o(P_i))$  is approximately the sum of all probabilities (by ignoring higher order terms). Thus, the main difference between the Probabilistic OR and the AVG model is the normalization by the total number of matching fields.

These three combination methods all exhibit three common properties: i) monotonicity – adding a relevant field will not decrease the final score, ii) probability preserving mapping – the final score is still in the  $[0, 1]$  range, thus could still be interpreted as a probability and iii) submodular – “diminishing return”, as the number of relevant fields increases, the benefit of adding additional relevant fields decreases.

The most salient difference among the three is that MAX and OR care mostly about the highest scoring field, disregarding other fields (e.g. mismatched fields), with the output of OR even increasing with more irrelevant fields. For example, when combining probabilities 0.1 and 0.9, MAX will output 0.9 and OR will output 0.91, while AVG yields 0.5. Here, AVG is the only measure that penalizes distracting fields, which, following Section 3.3.1, is equivalent to the length normalization in term-based language models, where distracting terms increase the document length and thus decrease the probability of generating the query

**Table 2.** Smoothed log probabilities of the three combination methods on example sentences. The query is `#combine[sentence](#combine[target](take #combine[./arg1](measures ) ) )`. Through smoothing, both document and collection language models affect the final scores. Bold faced scores are the largest for each method. For each sentence, brackets with types show the relevant semantic role labels in the sentence.

AVG	OR	MAX	Sentence
-1.545	-0.5075	-0.5085	1) It must [take] <sub>target</sub> [measures] <sub>arg1</sub> .
-1.881	-0.8467	-0.8520	2) U.S. investments worldwide could be in jeopardy if other countries [take] <sub>target</sub> up [similar measures] <sub>arg1</sub> .
-2.349	<b>-0.2425</b>	-0.5030	3) [Chanting] <sub>target</sub> the slogan "[take] <sub>target</sub> [measures] <sub>arg1</sub> before they [take] <sub>target</sub> [our measurements] <sub>arg1</sub> ," the Greenpeace activists [set] <sub>target</sub> up a coffin outside the ministry to [draw] <sub>target</sub> attention to the deadly combination of atmospheric pollution and [rising] <sub>target</sub> temperatures in Athens, which are [expected] <sub>target</sub> to [reach] <sub>target</sub> 42 degrees centigrade at the weekend.
-2.401	-0.4817	<b>-0.5012</b>	4) SINGAPORE, May 14 (Xinhua) -- The Singapore government will [take] <sub>target</sub> [measures] <sub>arg1</sub> to [discourage] <sub>target</sub> speculation in the private residential property market and [tighten] <sub>target</sub> credit, particularly for foreigners, Deputy Prime Minister Lee Hsien Loong [announced] <sub>target</sub> here today.

terms. AVG ensures that the more focused a document is about the query terms, the higher the ranking. This makes much sense in keyword queries, where users may expect the whole document to be relevant to the query instead of just a small part of the document being relevant to the query. However, in cases other than keyword document retrieval, for example, sentence level question answering, since the unit of retrieval is much smaller and the users' effort to create structured queries is much larger, the user might ignore the effort of examining the whole result sentence to find just a small clause or phrase that is relevant to the query. Thus, a sentence with just a small relevant portion would be considered highly relevant, as long as the relevant portion shares a close match with the query.

In short, more often than not, the users or the different application scenarios should determine which model to use. More detailed examples and results are shown in Table 2 and Table 4.

### 3.6 A detailed example

Table 2 shows the relevance scores for an example query to illustrate the three different methods in action.

OR prefers longer sentences with many target verbs, even though most of the target words do not match, as shown in Table 2, sentence 3. This is because of the incremental values of the multiple smoothing probabilities of the unmatched target fields.

AVG prefers short sentences with concentrated occurrences of matching fields. This behavior is also present in the case of term-based language models. Generally this is the same bias toward short sentences that is present in the traditional language model retrieval model with Dirichlet smoothing.

MAX is the only one of the three combination methods that does not care about sentence length or the number of queried fields in a sentence; it only cares about the generation probability of the best field in the sentence. When the field language model is smoothed with the document model, MAX prefers matching fields appearing in highly matched documents. This is actually why the scores of sentences 1 and 4 in Table 2 are different for the MAX combination method, even though both contain perfect matches.

## 4. EXPERIMENTS

### 4.1 Data sets

Experiments were done with two datasets, the "INEX" and "QA" datasets that reflect the two usage scenarios described above.

The INEX dataset consisted of the Wikipedia collection of XML documents, using INEX 2006 and 2007 topics, which contain structural hints (Content And Structure queries). INEX queries can retrieve XML elements of any type however our goal was to investigate the use of evidence from contained (inner) fields in ranking containing (outer) fields. To better match our goals, topics and relevance assessments were transformed in three ways: i) element retrieval queries were surrounded by a #combine operator so that they would use evidence from contained fields to rank and return whole documents, ii) a document was considered relevant if it contained at least one relevant element, and iii) for the NDCG metric, a document's degree of relevance was determined by the proportion of its text contained in relevant elements.

The QA dataset consisted of the AQUAINT corpus, annotated with ASSERT semantic role labels, as described in [8]. This corpus has a matching set of TREC topics. Structured queries were created from pre-identified relevant sentences, i.e. the structured queries are best-case queries with the same set of keywords as the original queries. As in prior research [8], all of the relevant sentences are used to generate structured queries for each topic. However, a difference from [8] is that each constructed structured query in turn is used to retrieve all the relevant sentences for the original topic. Thus, the evaluation not only measures how accurate the retrieval model is, but also the model's generalization ability, i.e. how well it handles the structural mismatches between the query and the rest of the relevant sentences that are not used to generate the query.

### 4.2 Experiments

In all the experiments, a training set was used to learn the optimal smoothing parameters. A grid parameter sweep was used to learn the parameters. The optimal values are listed below.

Datasets	Queries	Jelinek-Mercer		Dirichlet	
		$\lambda_1$	$\lambda_2$	$\mu_d$	$\mu_c$
INEX	Keyword	0.6	0.2	Any	800
	Structured	0.7	0.1	100	800
QA	Keyword	0.6	0.2	10	1000
	Structured	0.7	0.1	5	50

#### 4.2.1 Comparison of term smoothing methods

Since term level smoothing is the basis for all the retrieval models described in this paper, we start by finding the most effective term

**Table 3.** A comparison of two-level Jelinek-Mercer and two-level Dirichlet smoothing on the INEX and QA datasets.

Collections	Metric	Structured query			Keyword query		
		2-level Jelinek-Mercer	2-level Dirichlet	Improvement%	2-level Jelinek-Mercer	2-level Dirichlet	Improvement%
INEX06 (training)	MRR	0.7302	0.7872	+7.806*	0.7017	0.7560	+7.738*
	P@10	0.4694	0.4710	+0.3409*	0.4790	0.5081	+6.075**
	MAP	0.2900	0.2927	+0.9310	0.2918	0.2956	+1.302**
INEX07 (test)	MRR	0.7061	0.7386	+4.603	0.6515	0.7734	+18.71
	P@10	0.4517	0.4633	+2.568	0.4433	0.4667	+5.279**
	MAP	0.2838	0.2871	+1.163	0.2839	0.2979	+4.931**
QA (training)	MRR	0.6371	0.6729	+5.619	0.5211	0.5227	+0.3070
	P@10	0.2253	0.2371	+5.237***	0.2098	0.2336	+11.34***
	MAP	0.1402	0.1460	+4.137***	0.1651	0.1755	+6.299**
QA (test)	MRR	0.5128	0.5138	+0.1950	0.1649	0.1791	+8.611
	P@10	0.1684	0.1764	+4.751*	0.0808	0.0808	0.0000
	MAP	0.1634	0.1623	-0.6732**†	0.1197	0.1189	-0.6683

Notations, \*: significance level < 0.04, \*\*: significance level < 0.002, \*\*\*: significance level < 0.00001, †: Dirichlet is still significantly better by sign test even though the average MAP is lower than Jelinek-Mercer. Same notation is used in Table 4.

**Table 4.** A comparison of field level evidence combination methods (top 1000 results). For structured queries, the best performing smoothing method – two-level Dirichlet – was used. The best keyword retrieval results are also listed.

Structured query	Metric	AVG	MAX	OR	Keyword query
INEX 06 (training)	MRR	0.7560	0.7818	<b>0.8093</b>	<b>0.8538</b>
	P@10	<b>0.5081</b>	0.5065	0.4661	<b>0.5274</b>
	MAP	0.2956	<b>0.3094</b>	0.2914	<b>0.3538*</b>
INEX 07 (test)	MRR	<b>0.7734</b>	0.7552	0.7386	<b>0.8129</b>
	P@10	<b>0.4667</b>	0.4583	0.4633	<b>0.4783</b>
	MAP	0.2979	<b>0.3006</b>	0.2871	<b>0.3463*</b>
QA (training)	MRR	0.4265	<b>0.6452*</b>	0.0762	0.5227
	P@10	0.1725	<b>0.2373</b>	0.0313	0.2336
	MAP	0.1251	<b>0.1501</b>	0.0225	<b>0.1755**</b>
QA (test)	MRR	0.3947	<b>0.5138***</b>	0.0701	0.1791
	P@10	0.1428	<b>0.1752***</b>	0.0312	0.0808
	MAP	0.1264	<b>0.1617</b>	0.0364	0.1189

smoothing method. Fixing the two collections and their corresponding sets of queries, we compared two-level Jelinek-Mercer with two-level Dirichlet. For each query, the top 1000 results were returned, if possible. For structured queries, the best performing field evidence combination method was used – MAX for QA and AVG for INEX. For keyword queries, since the INEX queries are for document retrieval, two-level smoothing is the same as collection-level smoothing, while for QA sentence retrieval queries, they are distinct.

Results as summarized in Table 3 show two-level Dirichlet to be consistently better than two-level Jelinek-Mercer on all evaluation metrics and both structured retrieval tasks. Comparison with traditional single level (collection level) smoothing is omitted as previous study has already shown document level smoothing to be helpful [8], and our experiments also confirm that on both tasks.

The next experiment fixed the term smoothing and compared different structured retrieval models as described in Section 3.5. Results are presented in Table 4 and 5.

#### 4.2.2 Structured retrieval models on INEX dataset

From Table 4, first two rows, OR combination method works comparably with MAX and AVG, and the differences are not distinguishable. We hypothesize that the reason OR combination

method is better than AVG is because of the way the test is conducted. When converting the INEX element level judgements into document level, a document is considered relevant if any one of its element is relevant. This way the evaluation does not penalize long documents with only a small relevant part.

In order to take into account the lengths of non-relevant parts of result documents, a slightly customized version of the NDCG measure [17] is adopted, with two simple modifications. In the INEX datasets, the sizes of the relevant elements (in bytes) for a relevant document are given, and the length of a document can be easily measured by extracting the content text of the XML documents excluding XML tags. Thus, first, the percentage of the relevant text (in bytes) of a document is used to quantize the relevance of a document, and second, when calculating the discounting factor, instead of using the rank of the document, the sum of lengths of the preceding documents in the rank list is used.

As presented in Table 5, under NDCG, the AVG combination is significantly better than OR, almost always. From the average lengths of the returned top ranked documents, it can be seen that OR returns documents that are much longer than AVG. In this case, most of the large documents will only have a small portion relevant, leading to small NDCG.

**Table 5.** Performance on INEX collection measured in NDCG @ top 10, 20 and 30 results. We also give the average document length of the top 10, 20 and 30 results to observe the length bias of the methods. The differences between keyword queries and AVG structured retrieval (bold-faced results) are not significant.

Structured retrieval		AVG		MAX		OR		Keyword query	
Smoothing method		J-M <sup>1</sup>	Dir <sup>1</sup>	J-M	Dir	J-M	Dir	J-M	Dir
INEX 06 (training)	NDCG@10	0.4821	<b>0.5017*</b>	0.4320	0.4478	0.4048	0.4238	0.4555	<b>0.5281</b>
	NDCG@20	0.5490	<b>0.5669*</b>	0.5012	0.5164	0.4671	0.4857	0.5376	<b>0.5977</b>
	NDCG@30	0.5847	<b>0.5889*</b>	0.5184	0.5388	0.4903	0.5091	0.5816	<b>0.6514</b>
	AvgLen@10	5818.6	6268.8	8296.6	7981.1	12635	12361	5138.9	9906.8
	AvgLen@20	5327.4	5621.5	7715.2	7708.2	11220	10459	5106.3	8717.4
	AvgLen@30	5140.0	5228.1	7654.7	7086.6	11144	9792.4	5063.1	8065.9
INEX 07 (test)	NDCG@10	0.4755	<b>0.5291*</b>	0.4555	0.4794	0.4984	0.4827	0.4652	<b>0.5063</b>
	NDCG@20	0.5405	<b>0.5974*</b>	0.5141	0.5509	0.5537	0.5362	0.5570	<b>0.5797</b>
	NDCG@30	0.5788	<b>0.6390*</b>	0.5574	0.5893	0.5895	0.5729	0.6212	<b>0.6388</b>
	AvgLen@10	3819.6	5218.7	6488.6	6943.8	10712	10900	3464.8	8736.7
	AvgLen@20	4135.3	4704.1	6233.7	6576.1	9759.2	9553.5	3367.1	7784.5
	AvgLen@30	3875.3	4493.9	5955.0	6066.1	8853.6	8668.3	3493.9	7228.8

<sup>1</sup> J-M for Jelinek-Mercer smoothing, Dir for Dirichlet. For structured queries, two level smoothing is used.

\* The AVG combination method with Dirichlet smoothing is significantly better than any of the other methods (for the 62 training and 60 test topics, significant at  $p < 0.007$ ).

**Table 6.** Performance on the corrected queries of INEX 07

NDCG	Keyword	Structure	Change	Significance
@10	0.5063	0.5530	+20.2%	$p < 0.006$
@20	0.5797	0.6408	+20.4%	$p < 0.0004$
@30	0.6388	0.6854	+18.1%	$p < 0.002$

In terms of length biases, the Jelinek-Mercer smoothing is biased toward shorter documents as compared to Dirichlet smoothing. The OR combination method is biased toward longer documents as compared to AVG. The optimal performance in NDCG is from AVG combination method + two-level Dirichlet smoothing.

The optimal structured retrieval algorithm is on par with the best keyword performance. On INEX 06 dataset, keyword is slightly better, but not significant, while on INEX 07, structure is slightly better. The performance of the structured queries depends largely on how well the queries are constructed to match the structures in the relevant documents. We did an error analysis of the structured queries on the INEX06 collection and the result suggests that in many cases, the queries are not well constructed. For example, i) the topic asks for figures, but the query doesn't use evidence from [figure] extents, ii) the topic asks for "coordinates" or "populations" of "European capital cities", while the query uses "coordinates" and "populations", iii) using "origin of universe" as an exact phrase match which is overly restricting, instead, an unordered window operator should be used. Thus we corrected these structured queries whose semantic interpretation is obviously different from the topic description. We also changed the queries to use the document model instead of the article field language model, which is a minor change that can be made automatically.

Manually correcting the structured queries only requires an understanding of the topics themselves, and for the test topics, no results were referenced in any way.

With the proofread INEX07 structured queries, structured retrieval outperforms keyword retrieval significantly, with degree of freedom 30, details in Table 6. As far as we know, this is the first time significant improvement over keyword retrieval is observed on the INEX datasets.

With accurate structured queries, the AVG model outperforms keyword retrieval significantly. The QA queries demonstrate this effect more (Table 4), as they are more accurately constructed.

#### 4.2.3 Structured retrieval models on QA dataset

Several points can be made about the results on the QA dataset: i) well constructed structured queries perform better than keyword in top precision. Since the queries come from true relevant sentences, they are more accurate than the INEX human conjectured queries. ii) The model also generalizes from the structured query to the other relevant sentences reasonably well. P@10 and MAP were comparable or better than keyword. iii) The MAX combination method outperforms OR or AVG, probably because the sentence level judgment does not discount irrelevant parts of a sentence. iv) The difference between two-level Dirichlet and two-level Jelinek-Mercer is larger than that on the INEX data. The reason might be that because the optimal parameter values of Dirichlet smoothing depends on the average length of the queried extents [7], and since INEX queries contain extents of very different lengths (section, paragraph, figure etc.), it's difficult to find one optimal value for all the queries. In this case, a field dependent smoothing is more desirable, and we leave it to future work.

## 5. RELATED WORK

There is considerable prior research about combining evidence from the different fields of a document (e.g. combining title, in-link anchor and body text of HTML). The evidence combination in this paper is different. It deals with the case of the same type of field occurring multiple times in a result, e.g. combining evidence from all paragraphs of a document, or from targets in a sentence.

[11] shows a use of mixture language models for XML retrieval, which essentially flattens the XML element structure and mixes the language model of the current element with all its ancestors' language models. This flat structure among elements makes the model more robust to inaccurate queries where the user is not sure whether a word should appear in a child field, but writes the query in that way anyway. For the same reason, there is not much sensitivity to the structured query when it is in fact accurate.

Using element length as a prior for preferring longer XML elements has been found useful in the past [10]. However, since the experiments described in our work are document or sentence level retrieval, where the unit of the results is of more-or-less uniform lengths, length priors are not used.

[14] uses a hierarchical language modeling approach for retrieval of XML elements. In that model, elements are smoothed with the containing parent and ancestor elements. That model targets keyword queries with structured documents, while this work explores query structures. In terms of smoothing, the two-level smoothing methods (extents smoothed with document and collection models) used in this work can be seen as a special case of the hierarchical smoothing used in [14]. The two-level Dirichlet smoothing is new in this work and has not been applied to the structural retrieval problem in the literature, to the best knowledge of the authors.

## 6. CONCLUSIONS

This paper proposed a generative retrieval model for using structured queries to retrieve structured texts. The model is investigated in the context of the open-source Indri search engine, although it could be applied in other search engines, as well. The model uses smoothing to provide approximate matching at both the field level and term level. Field level smoothing is a generalization of the term level smoothing in traditional term-based language modeling approaches. For term level smoothing, our experimental results show that two-level (document + collection) Dirichlet smoothing significantly outperforms two-level Jelinek-Mercer smoothing on two datasets.

This paper also identifies evidence merging, in which the retrieval model combines evidence from individual occurrences of the same type of field, as a serious issue for structured retrieval. We investigate the use of Average (AVG), Maximum (MAX), and probabilistic OR combination strategies. Example queries show that AVG is biased towards shorter fields, MAX is not as biased, and OR prefers long fields. Experiments show that MAX is best on the Question Answering task, in which fields tend to be very short and irrelevant portions within a relevant sentence is not much of a factor. However, in XML document retrieval, where fields tend to be longer and redundancy within a document more indicative of relevance, the three combination methods are about equal when measured by MAP, MRR, and P@10, while AVG is best when measured by NDCG.

Overall, the performance of the best structured retrieval models is not worse and sometimes better than the best performing keyword retrieval methods. When the structured queries are reasonably accurate, i.e. for the QA dataset and the corrected INEX queries, structured retrieval significantly outperforms keyword. The results of this paper show that our generative structured retrieval model is robust under less accurate queries and outperforms keyword with high quality structured queries.

## 7. ACKNOWLEDGMENTS

The authors thank Paul Ogilvie and Matthew Bilotti for their work on the Indri search engine, on INEX and TREC QA datasets, and for helpful suggestions and discussions.

This work is supported by National Science Foundation grant IIS-0534345, and AQUAINT program award NBCHC040164. The views and conclusions contained in this document are those of the authors' and do not necessarily reflect those of the sponsor.

## 8. REFERENCES

- [1] Norbert Gövert and Gabriella Kazai. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002
- [2] Börkur Sigurbjörnsson and Andrew Trotman. Queries: INEX 2003 working group report. 2003
- [3] A. Echihiabi and D. Marcu. A Noisy-Channel Approach to Question Answering. In *Proceedings of the 41st Annual of the Association for Computational Linguistics*. 2003
- [4] J. Prager, J. Chu-Carroll, E. W. Brown and K. Czuba. Question Answering By Predictive Annotation. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research in Information Retrieval*. 2000.
- [5] Jay M. Ponte and W. Bruce Croft. A Language Modeling Approach to Information Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1998
- [6] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to Ad Hoc information retrieval. In *Proceedings of 24th International ACM SIGIR Conference on Research in Information Retrieval*. 2001
- [7] Hui Fang, Tao Tao and Chengxiang Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. 2004
- [8] M. W. Bilotti, P. Ogilvie, J. Callan and E. Nyberg. Structured Retrieval for Question Answering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in information retrieval*. 2007
- [9] INDRI - Language modeling meets inference networks. <http://www.lemurproject.org/indri/>. As of May 2, 2008
- [10] Jaap Kamps, Maarten de Rijke and Börkur Sigurbjörnsson. Length normalization in XML retrieval. In *Proceedings of the 27th annual international ACM SIGIR Conference on Research and Development in Information Retrieval*. 2004
- [11] Djoerd Hiemstra. Statistical Language Models for Intelligent XML Retrieval. *Lecture Notes in Computer Science*. Vol 2818/2003 pp 107-118. 2003
- [12] Don Metzler and Bruce Croft. Combining the Language Model and Inference Network Approaches to Retrieval, *Information Processing and Management*, 40(5), 2004.
- [13] T. Strohman, D. Metzler, H. Turtle, and B. Croft. Indri: A language-model based search engine for complex queries (extended version). *Technical Report IR-407*, Department of Computer Science, University of Massachusetts. 2005.
- [14] P. Ogilvie and J. Callan. Hierarchical Language Models for Retrieval of XML Components. In *Proceedings of the Initiative for the Evaluation of XML Retrieval Workshop (INEX 2004)*. 2004.
- [15] Saadia Malik, Andrew Trotman, Mounia Lalmas, Norbert Fuhr. Overview of INEX 2006. In *INEX 2006 Workshop Proceedings*, pp1-11. 2006
- [16] Andrew Trotman and Mounia Lalmas. Why Structural Hints in Queries do not Help XML-Retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006.
- [17] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2000.