# Combining the Language Model and Inference Network Approaches to Retrieval

Donald Metzler and W. Bruce Croft

*Department of Computer Science*
*University of Massachusetts*
*Amherst, MA 01003-9264*

**Abstract**

The inference network retrieval model, as implemented in the InQuery search engine, allows for richly structured queries. However, it incorporates a form of *ad hoc tf.idf* estimates for word probabilities. Language modeling offers more formal estimation techniques. In this paper we combine the language modeling and inference network approaches into a single framework. The resulting model allows structured queries to be evaluated using language modeling estimates. We explore the issues involved, such as combining beliefs and smoothing of proximity nodes. Experimental results are presented comparing the query likelihood model, the InQuery system, and our new model. The results reaffirm that high quality structured queries outperform unstructured queries and show that our system consistently achieves higher average precision than InQuery.

*Key words:* Inference network model, language modeling, structured query language

## 1 Introduction

Most information retrieval systems require a user to formulate a query by transforming a complex representation of an information need into a short list of keywords. For example, if a user is interested in learning more about stemming in information retrieval they might formulate the natural language query "`stemming information retrieval`", or even more ambiguously just "`stemming`". A lot of semantics is lost by transcribing the information need into a set of keywords. Any *a priori* knowledge a user has may not be expressible using this type of query. For instance, in our example the user knows that any relevant documents will most likely contain the phrase *information*

*retrieval* and that *information retrieval* and *IR* should be treated as a single concept. Such a query can be formulated as a *structured query* that represents a more accurate encoding of the user's original information need. A structured query language allows term weighting, the use of proximity information among terms, and various ways of combining concepts. Thus, structured queries can be more expressive than their *flat* (natural language) query counterparts. Therefore, it is plausible that retrieval models that can evaluate structured queries have more potential to satisfy the user's information need.

The term "structured query" has been used in both the database and information retrieval communities. In the information retrieval community, the "structure" typically refers to operators that are used to express constraints between the words in the retrieved documents, such as Boolean and proximity operators. Constraints on word occurrences in specific parts of the document structure, such as fields, are also frequently used.

The inference network model (Turtle and Croft, 1991), as implemented in the InQuery retrieval system (Callan et al., 1992, 1995), allows for structured queries via a rich set of probabilistic operators (Greiff et al., 1999). The model is formal in the sense that it is based on the Bayesian network formalism. Term observation probabilities, however, are estimated by *ad hoc* normalized *tf.idf* weights (Callan et al., 1995). There is little justification for using these weights but it has not been clear that there are better alternatives. The language modeling approach to retrieval (Ponte and Croft, 1998; Berger and Lafferty, 1999) provides a theoretically well-grounded framework for estimating probabilities using smoothing techniques (Zhai and Lafferty, 2001a). This model has become increasingly popular and has been used to describe a number of aspects of information retrieval (Croft and Lafferty, 2003). None of the variations of this model, however, describe how structure can be used in queries. This causes a considerable limitation in the practical application of this model. In this paper we address the problem of probability estimation in the inference network model and the problem of expressing structure in queries in a language modeling system by combining the two frameworks. The combination uses inference nets to express complex queries and language models to estimate the probabilities needed to evaluate those queries. The remainder of the paper further details the synthesis of the inference network and language modeling approaches into a single retrieval model, and shows that this model produces results that are more effective than either the language modeling approach or the inference network approach on their own.

We provide an overview of the inference network retrieval model and language modeling in Section 2. Section 3 then describes our model, discusses how beliefs are combined, and looks at the smoothing of proximity nodes. Section 4 presents experimental results for flat and structured queries and provides a comparison of InQuery, the query likelihood model, and our approach. Finally,

Section 5 discusses future work.

## 2 Background

Inference networks and language modeling form the core of our retrieval method. In this section we explain the relevant previous work and the advantages and disadvantages of the two models.

### *2.1 Inference Network Model*

The inference network retrieval model, as we discussed in the introduction, is based on the formalism of Bayesian networks. Bayesian networks are directed, acyclic graphical models (DAGs). Each node in the graph represents an event with either a discrete or continuous set of outcomes. Furthermore, each non-root node stores a conditional probability table that fully describes the probability of the outcomes associated with that node occurring given its parent nodes. Each outcome associated with a root node is assigned a prior probability. Given a DAG, priors, conditional probability tables and any observed events (evidence), we are able to calculate the probability, or belief, of an outcome at any node by propagating beliefs through the network (Pearl, 1988).

Turtle and Croft (1991) showed that it was possible to formulate information retrieval as a Bayesian network. The resulting model is called the inference network retrieval model (Turtle, 1991). Figure 1 depicts a simplified inference network model. All events in the network are binary. That is, *true* and *false* are the only possible outcomes.

#### *2.1.1 Document Nodes*

The inference network model is comprised of four types of nodes. The first set, the $d_i$ nodes, correspond to the event that a document is observed. Therefore, we need a $d_i$ node for every document in our corpus. Furthermore, we assume that only one document is observed at any time. Thus, if document $d_j$ is observed then $P(d_{i \neq j} = true) = 0$ and $P(d_j = true) = 1$.

#### *2.1.2 Representation Nodes*

The $r_k$ nodes are representation nodes. A representation can be any easily indexed feature of a document. We distinguish between single term and prox-

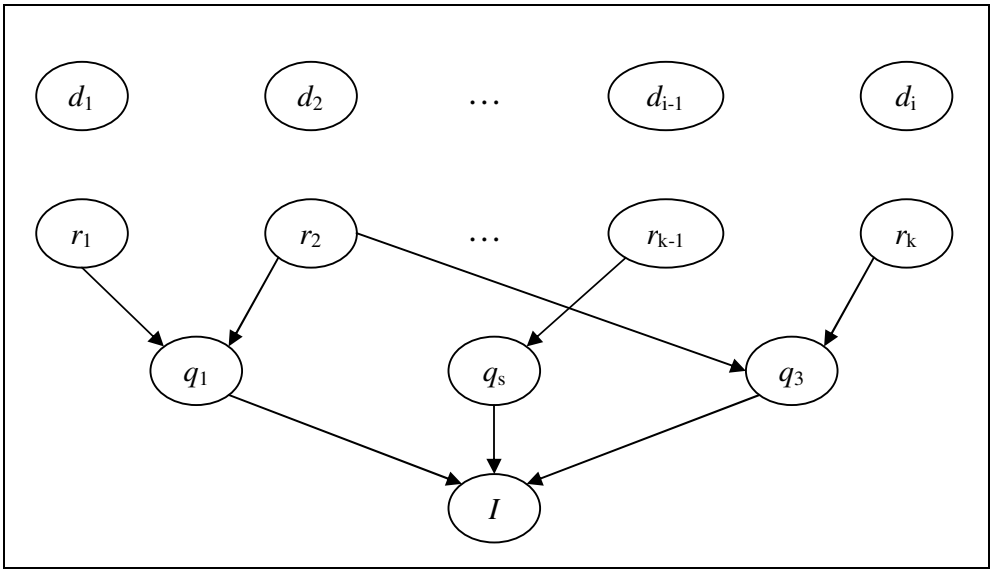


Fig. 1. Simplified inference network model

imity representations. A single term representation corresponds to some term in the corpus. Proximity representations are more complex. They can be used to represent phrases, terms appearing ordered or unordered within a fixed length window of words, and other such concepts. Returning to the example in the introduction, the phrase *information retrieval* is an example of a proximity representation. The event associated with it is the event that the phrase *information retrieval* is extracted from a subset of the corpus (Turtle and Croft, 1991). The nodes associated with proximity representations are *proximity nodes*. InQuery allows proximity representations to take on many different forms. These are discussed further in Section 2.1.5.

Each representation node has every document as a parent. For clarity, these connections are omitted from Figure 1. Since we assume that a single document is observed at any time, we must estimate $P(r_k = true | d_j = true, d_{i \neq j} = false)$ for all representation nodes, where $d_j$ is the observed document. For notational simplicity we define $P(q = true | d_j = true, d_{i \neq j} = false)$ as node $q$'s belief and denote it as $bel(q)$. Turtle and Croft (1991) propose estimating these beliefs as:

$$bel(r_k) = db + (1 - db)\hat{tf}_{r_k, d_j} idf_{r_k} \tag{1}$$

where $db$ is an arbitrary default belief and the $\hat{tf}_{r_k, d_j}$ and $idf_{r_k}$ values can be calculated using any standard method for estimating $tf.idf$ weights for the representation $r_k$. The default belief ensures that every representation is allocated a nonzero belief for the observed document, even if it is not present in the document.

The InQuery system uses $db = 0.6$, the Okapi (Robertson and Walker, 1994) $tf$ score, and a standard $idf$ score. The exact expressions for representation $r_k$ and document $d_j$ are:

$$\hat{tf}_{r_k,d_j} = \frac{tf_{r_k,d_j}}{tf_{r_k,d_j} + 0.5 + 1.5\frac{|d_j|}{|D|_{avg}}}$$

$$idf_{r_k} = \frac{\log\left(\frac{N+0.5}{df_{r_k}}\right)}{\log\left(N+1\right)}$$

where $tf_{r_k,d_j}$ is described below, $df_{r_k}$ is the number of documents that $r_k$ appears in, $|D|_{avg}$ is the average length of a document in the corpus, and $N$ is the total number of documents in the collection. Note that $\hat{tf}_{r_k,d_j}$ represents the Okapi $tf$ score, whereas $tf_{r_k,d_j}$ is the actual representation frequency.

If $r_k$ is a single term then $tf_{r_k,d_j}$ is the number of times $r_k$ appears in document $d_j$. Similarly, if $r_k$ is a proximity representation then $tf_{r_k,d_j}$ is the number of times the corresponding proximity representation is matched within the document. For example, the $tf_{r_k,d_j}$ value associated with the representation node corresponding to the exact phrase *information retrieval* is the number of times that *information retrieval* appears ordered in $d_j$.

Although scoring based on *tf.idf* weights has worked well in many information retrieval systems, including InQuery, it is heuristic and not based on any formal underpinnings. Additionally, the only future improvements possible require further tuning of the heuristic. We will show that $bel(r_k)$ can be efficiently estimated using smoothed language modeling techniques and how such estimates offer advantages over *tf.idf* estimates.

### *2.1.3 Query Nodes*

Next, the $q$ nodes correspond to the query nodes. They allow us to combine beliefs about representation nodes and other query nodes in a structured manner. We may assign them any possible conditional probability table. However, this quickly becomes infeasible as the number of parents associated with a node increases. Therefore, we must restrict ourselves to computationally and space efficient methods for storing and marginalizing over the conditional probabilities.

In Bayesian networks, *link matrices* (Greiff et al., 1999) provide a way to represent conditional probabilities. Suppose we are given a node $q$ with $bel(\pi_i) = p_i$ for all parent nodes $\pi_i$ of $q$ for $i = 1 \ldots n$. We define $q$'s link matrix to be the 2 by $2^n$ matrix that completely encodes $q$'s conditional probability table. Then

$q$'s link matrix has the form:

$$L = \begin{pmatrix} 1 - p_{0\ldots000} & 1 - p_{0\ldots001} & 1 - p_{0\ldots010} & \cdots & 1 - p_{1\ldots111} \\ p_{0\ldots000} & p_{0\ldots001} & p_{0\ldots010} & \cdots & p_{1\ldots111} \end{pmatrix}$$

and

$$p_{b_1 b_2 \ldots b_n} = P(q = true | \pi_1 = \hat{b}_1, \ldots, \pi_n = \hat{b}_n)$$

where $\hat{b}_i = true$ if $b_i = 1$ and $\hat{b}_i = false$ if $b_i = 0$. The second row of $L$ defines the belief that the event associated with node $q$ occurs for all of the possible $2^n$ configurations of the parents, which are encoded by a bit string, as in Greiff et al. (1999). Since we assume each node to be binary, the sum of the probabilities in each column must be 1.

We can then compute $bel(q)$ as follows:

$$\begin{aligned} bel(q) &= \sum_{\hat{b}_1, \ldots, \hat{b}_n} P(q = true | \pi_1 = \hat{b}_1, \ldots, \pi_n = \hat{b}_n) \prod_{i=1}^{n} P(\pi_i = \hat{b}_i | E) \\ &= \sum_{\hat{b}_1, \ldots, \hat{b}_n} p_{b_1 b_2 \ldots b_n} \prod_{i=1}^{n} P(\pi_i = \hat{b}_i | E) \end{aligned}$$

where $E$ is the evidence. Therefore, in general, computing beliefs requires time exponential in the number of parents and is not feasible in practice. However, a family of link matrices exist that allow this marginalization to be done efficiently. We now give a toy example of such matrices to further solidify the idea. The following are link matrix forms for a query node $q$ that has parent nodes $a$ and $b$

$$L_{and} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$L_{or} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$L_{wsum} = \begin{pmatrix} 1 & 1 - \frac{w_b}{w_a + w_b} & 1 - \frac{w_a}{w_a + w_b} & 1 - \frac{w_a + w_b}{w_a + w_b} \\ 0 & \frac{w_b}{w_a + w_b} & \frac{w_a}{w_a + w_b} & 1 \end{pmatrix}$$

where weights $w_a$ and $w_b$, associated with nodes $a$ and $b$, respectively.

These link matrices have very intuitive meanings. For instance, $L_{and}$ defines $P(q = true | a = true, b = true)$ to be 1 and all other outcomes for $a$ and $b$ to be 0. Thus $L_{and}$ mimics Boolean AND. For $L_{or}$, the belief event $q$ occurs is 1 only if one or more of the parent node events occurs. This is similar to Boolean OR. For $L_{wsum}$, each parent node event that occurs contributes some weight $w_i$ to the belief of $q$. Thus, the more parent nodes that occur, the more belief we have in $q$.

Marginalization over parent nodes can be done efficiently for belief operators defined by link matrices. For example, consider a node $q$ with conditional probability table defined by $L_{and}$ we compute $q$'s belief as:

$$\begin{aligned} bel_{and}(q) = & p_{00} P(a = false) P(b = false) \\ & + p_{01} P(a = false) P(b = true) \\ & + p_{10} P(a = true) P(b = false) \\ & + p_{11} P(a = true) P(b = true) \\ = & 0(1 - p_a)(1 - p_b) + 0(1 - p_a) p_b + 0 p_a (1 - p_b) + 1 p_a p_b \\ = & p_a p_b \end{aligned}$$

Thus, marginalization over $L_{and}$ can be done very efficiently.

The following is a list of closed form expressions for efficiently computing beliefs at query nodes. They are derived from marginalizing over their corresponding link matrices (Turtle, 1991). Both InQuery and our system limit query nodes to taking on one of these forms to allow for efficient belief propagation. For a node $q$ given $bel(\pi_i) = p_i$ for all parents $\pi_i$ of $q$ with weights $w_i$ for $i = 1 \ldots n$ we get:

$$bel_{not}(q) = 1 - p_1 \tag{2}$$

$$bel_{or}(q) = 1 - \prod_i (1 - p_i) \tag{3}$$

$$bel_{and}(q) = \prod_i p_i \tag{4}$$

$$bel_{max}(q) = \max\{p_1, p_2, \ldots, p_n\} \tag{5}$$

$$bel_{sum}(q) = \frac{\sum_i p_i}{n} \tag{6}$$

$$bel_{wsum}(q) = \frac{\sum_i w_i p_i}{\sum_i w_i} \tag{7}$$

As we will show in Section 2.1.5, query nodes are generated dynamically at query time and allow different beliefs within the network to be logically combined.

### *2.1.4 Information Need Node*

Finally, the $I$ node represents the event the information need is met. It is a query node that combines all of the evidence from the other query nodes into a single belief. Thus, to score a document we first instantiate the $d_i$ node it corresponds to. We then propagate beliefs through the network, starting with the document nodes, all the way down to the information need node. The belief value associated with $bel(I)$ is then interpreted as the document score. Doing so for every document in the corpus allows us to generate a ranked list of documents.

Therefore, the inference network retrieval model is both formal and robust. It allows us to efficiently combine beliefs about representations, such as terms and phrases, via a rich set of probabilistic operators described by link matrices. However, a drawback to this powerful model is the *ad hoc* estimation of representation probabilities in Equation 1. Thus, we turn to a more formal method of estimating these probabilities, such as language modeling.

### *2.1.5 Query Language*

Now that the framework has been set up, we describe how the representation, query, and information need nodes are dynamically generated given a structured query. As described in the introduction, InQuery uses a powerful structured query language. It is comprised of single terms, *proximity operators* and *belief operators*. Single terms and proximity operators correspond to representation nodes, and allow beliefs to be formed about single terms, phrases, terms appearing unordered within a fixed-sized window of text, among others. Belief operators, corresponding to query nodes, then allow us to combine these beliefs in a number of useful ways. Both proximity and belief operators have the general form `#OP(`$p_1 \ldots p_n$`)`, where `OP` is the operator name and the $p_i$'s are the parameters sent to the operator.

Single terms not occurring within a proximity operator generate single term representation nodes within the network. These nodes are implemented exactly the same as proximity nodes. Table 1 lists the proximity operators used in InQuery. The $q_i$ parameters are required to be proximity operators to ensure matching makes sense. A representation node (proximity node) is created for each proximity operator in the query. Given some document as evidence, beliefs are computed at these nodes using Equation 1. Table 1 details how matching is done for single terms and operators. These operators allow us to include many different kinds of representation nodes within our network beyond single term nodes. We limit ourselves to this set of operators, but note that practically any concept easily extracted from a document could be used.

| Operator | Name | Description |
|---|---|---|
| *term* | single term | A match occurs for every occurrence of the term in the document |
| `#ODN` $(q_1 \ldots q_n)$ | ordered window | A match occurs if the $q_i$'s appear in order with no more than $N$ words between adjacent terms. |
| `#UWN` $(q_1 \ldots q_n)$ | unordered window | Matches if the $q_i$'s appear in any order within a window of $N$ words |
| `#PHRASE` $(q_1 \ldots q_n)$ | phrase | Equivalent to `#OD3`$(q_1 \ldots q_n)$. |
| `#SYN` $(q_1 \ldots q_n)$ | synonym | The $q_i$'s are to be treated as synonyms. Each $q_i$ is treated as a match. |
| `#PASSAGEN` $(q)$ | passage | Evaluates $q$'s belief for every passage of length $N$ within a document and returns the highest belief. |

Table 1
Explanation of single terms and proximity operators from Callan et al. (1992).

| Belief Operators | Weighted Belief Operators |
|---|---|
| `#NOT` $(q_1)$ | `#WSUM` $(w_1\ q_1 \ldots w_n\ q_n)$ |
| `#AND` $(q_1 \ldots q_n)$ | `#WAND` $(w_1\ q_1 \ldots w_n\ q_n)$ |
| `#OR` $(q_1 \ldots q_n)$ | |
| `#MAX` $(q_1 \ldots q_n)$ | |
| `#SUM` $(q_1 \ldots q_n)$ | |

Table 2
List of query operators, where $w_i$'s are weights and $q_i$'s are either query or representation operators.

Table 2 lists the belief operators used in InQuery. Note that `#WAND` was not originally implemented in InQuery and will be discussed later. The $q_i$ parameters can either be proximity or query operators and the $w_i$'s are weights that allow parent beliefs to be assigned varying amounts of importance. A query node is created for each belief operator, and the nodes corresponding to each $q_i$ become the current node's parents. In this way, belief operators allow us to combine many different kinds of evidence in the network. Each belief operator corresponds to a link matrix that allows efficient marginalization within the network. Equations 2 – 7 are used to compute beliefs at these nodes.

Figure 2 shows the nodes dynamically generated for a simple query that contains both proximity and belief operators. For this query, documents would be ranked according to the belief associated with the `#WAND` node.

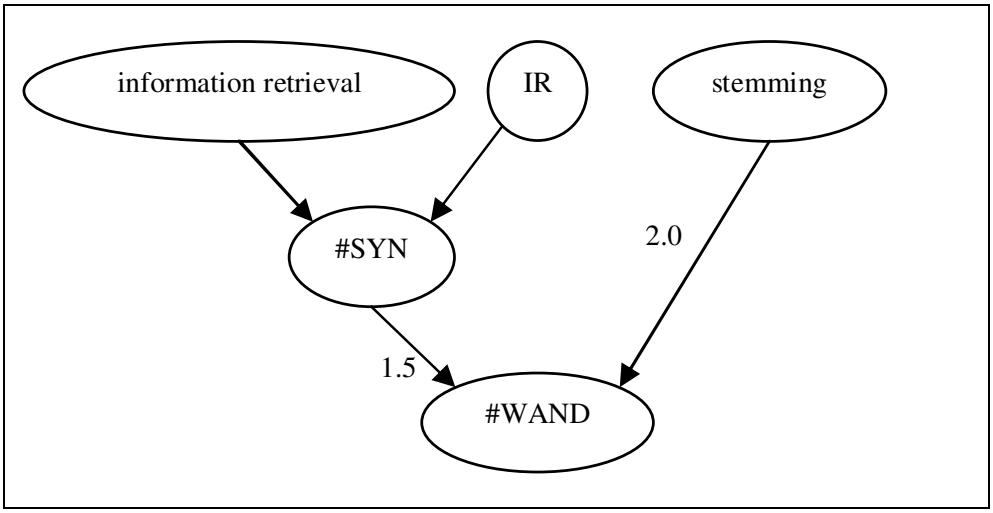


Fig. 2. Nodes generated for the query `#WAND` (1.5 `#SYN` (`#ODN3`(information retrieval) IR) 2.0 stemming)

### *2.2 Language Modeling*

#### *2.2.1 Overview*

Language modeling is a formal probabilistic retrieval framework with early roots in speech recognition (Rosenfeld, 2000) and statistical machine translation (Berger and Lafferty, 1999). The underlying assumption of language modeling is that human language generation is a random process. The goal is to model this process via a statistical model. We would then like to calculate the likelihood a sequence of language, such as a sentence, is generated by this model. Such statistical models are known as language models. A common class of language models, *n-gram* models, are used to calculate the probability a sequence of language is generated given the last $n-1$ words observed. If $M$ is a n-gram language model, then the probability some sequence of words $S = s_1, s_2, \ldots, s_k$ is generated is given by:

$$P(S|M) = \prod_{i=1}^{k} P(s_i|s_{i-1}, s_{i-2}, \ldots, s_{i-n+1}, M)$$

For $n = 1$ each term is generated independently of the previous terms. This is known as a *unigram* language model.

#### *2.2.2 Language Modeling in IR*

Language models have been successfully applied to information retrieval (Croft and Lafferty, 2003). Rather than using a single model for all of human lan-

guage, each document is assumed to be generated from a different model. This allows us to use statistical techniques to both estimate document models and score documents. We now describe the motivation behind the query likelihood model, which is one of the most widely used language modeling retrieval models in information retrieval.

Let us assume a simple unigram model for each document, where each document is represented as the standard "bag of words" and each document's language model is a distribution over a vocabulary of single words. The vocabulary for a given collection consists of the unique words that appear and are indexed. In most cases, the underlying document model is assumed to be a multinomial distribution, although other distributions are possible, such as the multiple Bernoulli (Ponte and Croft, 1998). Here, we assume it is a multinomial. A plausible but naive approach is to estimate $P(w|D)$ using the maximum likelihood estimate of $w$ occurring in $D$, which, for a multinomial distribution, is:

$$P_{ML}(w|D) = \frac{tf_{w,D}}{|D|}$$

where $tf_{w,D}$ is the number of times word $w$ appears in document $D$ and $|D|$ is the number of words in $D$.

Then, given the query $Q = q_1, q_2, \ldots, q_k$, consisting of single words, we can compute the likelihood it was generated given a document's language model $D$ as:

$$P(Q|D) = \prod_{i=1}^{k} P(q_i|D) \tag{8}$$

This likelihood is computed for each document and used for ranking. Ranking documents this way is known as the *query likelihood retrieval model*.

Although simple, maximum likelihood estimates have a serious flaw. When calculating document scores using Equation 8, the resulting likelihood is 0 if any of the query terms do not appear in $D$. It is unreasonable to give a document a score of 0 for a 10 word query if 9 of the query terms appear in a document and 1 does not. Therefore, to avoid this problem and overcome sparsity in the data, we must employ smoothing (Zhai and Lafferty, 2001b). We use *Jelinek-Mercer* smoothing, which is a method of interpolating between the document and collection language model. We chose Jelinek-Mercer smoothing for our experiments because structured queries tend to be long and it has been shown that Jelinek-Mercer smoothing outperforms other smoothing techniques on verbose queries (Zhai and Lafferty, 2001b). The smoothed $P(w|D)$ is calculated as follows:

$$P(w|D) = \lambda \frac{tf_{w,D}}{|D|} + (1-\lambda)\frac{cf_w}{|C|}$$

where $cf_w$ is the number of times word $w$ appears in the entire collection, $|C|$ is the number of words in the collection, and $\lambda$ is the smoothing parameter that may be set manually or automatically found (Zhai and Lafferty, 2002). Therefore, all terms in the vocabulary will be given non-zero probability in each document model and we avoid the 0 probability problem. Additionally, smoothing plays the important role of and *idf*-like component when scoring using the query likelihood model.

The query likelihood scoring function with smoothed estimates is:

$$\begin{aligned} P(Q|D) &= \prod_{i=1}^{k} \left( \lambda \frac{tf_{q_i,D}}{|D|} + (1-\lambda)\frac{cf_{q_i}}{|C|} \right) \\ &= \prod_{i:tf_{q_i,D}>0} \frac{\lambda \frac{tf_{q_i,D}}{|D|} + (1-\lambda)\frac{cf_{q_i}}{|C|}}{(1-\lambda)\frac{cf_{q_i}}{|C|}} \prod_{i=1}^{k} (1-\lambda)\frac{cf_{q_i}}{|C|} \\ &\stackrel{rank}{=} \sum_{i:tf_{q_i,D}>0} \log \left( \frac{\lambda \frac{tf_{q_i,D}}{|D|}}{(1-\lambda)\frac{cf_{q_i}}{|C|}} + 1 \right) \end{aligned}$$

As we see, the expression has both a *tf*- and *idf*- like component. Thus, smoothing plays an important role in the language modeling framework.

Language models are preferred over *tf.idf* weights because of their formal probabilistic meaning. Vector space models based on heuristic *tf.idf* weighting give us a high dimensional geometric interpretation of scores that are difficult to visualize. However, query likelihood scores, being probabilities, give document scores more of an intuitive meaning. Unfortunately, current language modeling-based retrieval systems have no way of evaluating the structured queries. Therefore, we focus on combining the structured probabilistic queries of the inference network with the formal language modeling term probability estimates under the umbrella of a single retrieval model to combine the benefits of these two powerful models. The following section details how this can be accomplished.

## 3 Language Modeling with Structured Queries

Combining the inference network model with language modeling is relatively straightforward. This section first describes our query model and how the representation probabilities are calculated. Next, we look at a new method for combining beliefs at query nodes. Finally, we explore smoothing techniques

applied to proximity nodes and show that our model subsumes the query likelihood model.

### *3.1 Model*

Our system is modeled after InQuery and uses a similar set of query operators. Table 2 lists the belief operators our system handles. See Section 2.1 for an explanation of each, except `#WAND` which is described shortly. Additionally, Table 1 lists and describes the proximity operators allowed in our model. These belief and proximity operators make up our model's query language. Figure 2 shows an example of how these operators can be used to combine single term representations, proximity representation (operators) and belief operators to form a structured query representation in the network.

The only remaining task is to describe how representation probabilities are estimated. The idea is simple. Rather than using the original *tf.idf* weights we will instead use probabilities estimated by smoothed language models. Thus, instead of using Equation 1, we choose to use the following estimate:

$$bel(r_k) = \lambda \frac{tf_{r_k,d_j}}{|d_j|} + (1-\lambda)\frac{cf_{r_k}}{|C|}$$

where $tf_{r_k,d_j}$ is defined as in Section 2.1.2 and $cf_{r_k}$, $r_k$'s collection frequency, is computed as $cf_{r_k} = \sum_j tf_{r_k,d_j}$. All necessary term statistics, including $cf_{r_k}$, can be efficiently calculated using the same machinery as InQuery.

Thus, we can estimate representation probabilities using the formal framework of language models while maintaining our ability to use structured query operators. This allows a wide range of possibilities and more flexibility over the original inference network framework. First, rather than having a single crude default belief for each representation node, we have representation dependent "default beliefs" (i.e. $(1-\lambda)\frac{cf_{r_k}}{|C|}$). Next, as we will discuss shortly, it is possible to apply different smoothing techniques to different kinds of nodes within the network to achieve better performance. Finally, as more sophisticated methods for estimating language models are developed, they can be used directly within the inference network framework as discussed above for even further improvement.

### *3.2 Combining Beliefs*

In the InQuery system, the information need node $I$ is very often represented by the `#WSUM` belief operator. That is, InQuery combines beliefs for the in-

formation need using a weighted averaging over $I$'s parent nodes. To see why this is reasonable, let us assume we have a simple query of the form `#WSUM` ($w_1$ $q_1, \ldots,$ $w_n$ $q_n$) for representations $q_i$. In the InQuery framework, propagating beliefs from the nodes corresponding to the $q_i$'s to $I$ for an observed document $d_j$ using the closed form equation for `#WSUM` gives:

$$\begin{aligned} bel_{wsum}(I) &= \frac{\sum_i w_i p_i}{\sum_i w_i} \\ &= \frac{\sum_i w_i (db + (1-db)\hat{tf}_{q_i,d_j} idf_{q_i})}{\sum_i w_i} \\ &\propto \sum_i w_i \hat{tf}_{q_i,d_j} idf_{q_i} \end{aligned}$$

Thus, applying `#WSUM` at node $I$ essentially gives us a form of *tf.idf* scoring. However, things go awry if we use `#WSUM` with language modeling representation probabilities. We would get the following scoring function for the same query:

$$\begin{aligned} bel_{wsum}(I) &= \frac{\sum_i w_i p_i}{\sum_i w_i} \\ &= \frac{\sum_i w_i (\lambda \frac{tf_{q_i,d_j}}{|d_j|} + (1-\lambda)\frac{cf_{q_i}}{|C|})}{\sum_i w_i} \\ &\propto \sum_i w_i \frac{tf_{q_i,d_j}}{|d_j|} \end{aligned}$$

As we showed, in language modeling for information retrieval, smoothing plays the same role as *idf* weighting in *tf.idf* based systems (Hiemstra, 2002; Zhai and Lafferty, 2001a). Using `#WSUM` with language modeling probability estimates results in the smoothing component $(1-\lambda)\frac{cf_{q_i}}{|C|}$ having no influence on document scoring. This leads to a poor scoring function based solely on term frequency. Therefore, combining beliefs at $I$ using `#WSUM` will not work for our model. A more appropriate choice is the `#WAND` , or *weighted and* operator. Suppose a query node $q$ has parent nodes $a$ and $b$ with weights $w_a$ and $w_b$, respectively. Also, let $bel(a) = p_a$ and $bel(b) = p_b$, then the link matrix for `#WAND` is defined as:

$$L_{wand} = \begin{pmatrix} 1 & 1 & 1 & 1 - w_q \\ 0 & 0 & 0 & w_q \end{pmatrix}$$

where $w_q = p_a^{w_a - 1} p_b^{w_b - 1}$. Using $L_{wand}$, the belief at $q$ can be efficiently computed as:

$$bel_{wand}(q) = \prod_i p_i^{w_i}$$

Although `#WAND` is not among the original query operator set, it is still a valid probabilistic model for combining probabilities within the network.

The operator has a number of interesting properties. $L_{wand}$ is a generalization of $L_{and}$ in the same sense that $L_{wsum}$ is a generalization of $L_{sum}$. $L_{wand}$ can be thought of as Boolean AND with uncertainty. That is, the belief associated with node $q$ is zero if any one of $q$'s parent node events did not occur, and is $w_q$, $0 \leq w_q \leq 1$, when all parent node events do occur. When $w_i = 1$ for all $i$, then $L_{wand}$ equals $L_{and}$. Thus, other non-uniform weights can be used to leverage our uncertainty. Using `#WAND` in place of `#WSUM` to compute the belief at $I$ gives:

$$\begin{aligned} bel_{wand}(I) &= \prod_i {p_i}^{w_i} \\ &= \prod_i (\lambda \frac{tf_{q_i,d_j}}{|d_j|} + (1-\lambda)\frac{cf_{q_i}}{|C|})^{w_i} \end{aligned}$$

which is akin to the query likelihood model and preserves the important dependence on smoothing (Zhai and Lafferty, 2001a). A closer look reveals that if each $q_i$ corresponds to a single term and each $w_i = qf_{q_i}$, then it is *exactly* the query likelihood model, when $qf_{w_i}$ is the number of times $w_i$ appears in the query. That is, `#WAND (2.0 wordA 1.0 wordB)` is equivalent to `#AND (wordA wordA wordB)` which produces the same ranked list as the flat query "`wordA wordA wordB`" evaluated under the query likelihood model. Therefore, `#AND` and `#WAND` are more well grounded methods for combining beliefs in our model.

Another elegant connection exists between `#WSUM` and `#WAND` . Without loss of generality we can assume that $\sum_i w_i = 1$. In this case `#WSUM` can be interpreted as the weighted arithmetic mean of its parents' beliefs and `#WAND` the weighted geometric mean. This intuitively makes sense since *tf.idf* scoring involves adding weights together and so the arithmetic mean is more appropriate, whereas language modeling probabilities are multiplied together making the geometric mean a good choice.

### *3.3 Smoothing*

Using language modeling probabilities allows us great flexibility in estimating representation probabilities. The smoothing parameter $\lambda$ can be set by hand tuning, using a held-out set of documents, or automatically (Zhai and Lafferty, 2002). Our system allows many smoothing options. Rather than having a single, fixed value for every representation node, it allows for different

smoothing parameters for each individual node or each type of node. Although it may be possible to "smooth" belief nodes, we focus our attention only on representation nodes.

Smoothing can be used to implicitly implement ranked variants of Boolean AND, OR, and NOT operators within the network. Suppose we are given the query `#AND (information retrieval)` and set the smoothing parameter $\lambda$ of the representation nodes corresponding to `information` and `retrieval` to 1. In this case, if the document does not contain both query terms the resulting score is 0. Similar results hold for `#OR` and `#NOT` . Thus, using no smoothing (i.e. $\lambda = 1$), our model can perform a scored variant of Boolean retrieval.

Smoothing can also be used to overcome data sparsity. We can use different smoothing parameters for each type of representation node. Since there is more data available for estimating the single term node beliefs than there is for the other types of proximity nodes, it is reasonable to believe that the single term nodes will need to be smoothed less. The results presented in the next section support this hypothesis, although a more detailed analysis of the different aspects smoothing plays on each type of representation node is necessary.

## 4 Results

This section describes our experimental setup and a comparison of the query likelihood model, InQuery, and our proposed model. We also present an analysis of proximity node smoothing. The goal of the experiments is twofold. First, we wish to reiterate the importance of structured queries over flat ones. Second, and more important, we aim to establish that our model performs at least as well as the InQuery system, except in a more formal framework.

### *4.1 Experimental Setup*

Our retrieval system was implemented by modifying the InQuery code in the Lemur Toolkit (Ogilvie and Callan, 2002). All InQuery test runs made use of the standard Lemur InQuery implementation with default parameters. The query likelihood retrieval system used is a straightforward implementation the results in Section 2. Each query was stemmed via the Porter stemmer and stoplisted via the standard InQuery stoplist. All experiments except those involving proximity smoothing have the single term smoothing parameter $\lambda = 0.6$ and proximity node smoothing parameter $\lambda_{prox} = 0.1$ unless otherwise specified. The specific value for $\lambda$ was chosen because it yielded good results on experiments in the past. Additional experiments are necessary to establish

how sensitive results are to this parameter. The value for $\lambda_{prox}$ was chosen because it yielded the best performance on the proximity smoothing experiments described later.

For our experiments we chose 4 sets of queries, all of which had been used with the InQuery system at past TREC conferences and had varying degrees of structure and verbosity. We tested our system on TREC 4, 6, 7, and 8 queries.

The TREC 6, 7, and 8 queries were automatically generated from their corresponding TREC topic and description fields. See Allan et al. (1997) for details of the automatic query generation process. Note, however, that we remove all terms added by query expansion from these queries. The generated queries are generally relatively short and contain little or no structure and few term weights. The corpus used to run these queries against is composed of TREC disks 4 and 5, minus the Federal Register data.

For the TREC 4 data, two sets of structured queries were available. The first set consists of automatically generated queries (Allan et al., 1995). These queries are very long, highly structured, and contain many noisy (irrelevant) terms. The second set is made up of manually edited versions of the automatic queries (Allan et al., 1995). This higher quality set contains mostly short, highly structured queries and considerably fewer noisy terms. Both query sets were used as is. These two sets allow us to compare the impact of query quality on retrieval performance. These sets were run against TREC disks 2 and 3.

The structured queries are used to compare InQuery and our model. They contain a mixture of belief and proximity operators. Flat versions of each structured query described above were created by removing all proximity operators. These queries were used in evaluating the query likelihood model since it can not evaluate structured queries.

The following is the TREC 4 manually edited, structured version of the flat query "`capital punishment deterrent crime`"

```
#WAND (1.0 #WAND ( 1.0 capital 1.0 punishment
                   1.0 deterrent 1.0 crime
                   2.0 #UW20(capital punishment deterrent)
                   1.0 #PHRASE(capital punishment)
       1.0 #PASSAGE200 ( 1.0 capital 1.0 punishment
                         1.0 deterrent 1.0 crime
                         1.0 #PHRASE(capital punishment) ) )
```

This query is one of the shorter structured queries from the data set. It serves to illustrate how much more expressive structured queries are over flat queries.

| | QL | InQuery | StructLM |
|---|---|---|---|
| TREC-6 | 0.1854 | 0.1622 | 0.1863 |
| TREC-7 | 0.1972 | 0.1803 | 0.2004 |
| TREC-8 | 0.2396 | 0.2343 | 0.2498 |

Table 3
Comparison of average precision for query likelihood (QL), InQuery, and our model for TREC-6, 7, and 8 topics.

### *4.2 Ad Hoc Retrieval Experiments*

Table 4 compares retrieval results for the query likelihood model, the InQuery system, and our model for both sets of TREC 4 queries. The table shows the standard TREC output used to compare retrieval performance across systems. First, we compare the performance of the flat and structured queries. For the automatically generated queries there is some improvement in average precision when using structured queries over flat ones. These queries are very long and contain many irrelevant term and proximity representations, thus the added structure does very little in terms of boosting performance. However, when the queries are more intelligently created there is a more noticeable improvement in average precision. There is a 12.1% increase for InQuery and 15.9% increase for our model for the manually edited versions of the structured queries over the manually edited flat queries. Therefore, query quality has a large effect on retrieval accuracy. Thus, high quality structured queries should lead to better results over their unstructured counterparts. Finally, as Table 4 shows, our model slightly outperforms InQuery in terms of average precision for both sets of queries. Therefore, we can conclude that our model is valid and capable of achieving results at least as good as InQuery.

Table 3 shows the results obtained for the TREC 6, 7, and 8 queries on the same set of systems. As was stated previously, these queries contained very little structure and were generally short. The simple query likelihood model outperforms the InQuery system on all 3 query sets. Our model achieved the highest average precision on each set. The proximity operators that do appear in these queries are low quality which lead to degraded retrieval performance by InQuery. Smoothing is the reason our model fares better than InQuery under such conditions. The InQuery system uses a crude form of smoothing that assigns a fixed default belief to any representation that does not appear in a document. This value may be overly optimistic or pessimistic for certain representations. Since our model uses $\lambda_{prox} = 0.1$ we heavily smooth these representation nodes and thus assign them more realistic beliefs.

| | QL | | InQuery | | StructLM | |
|---|---|---|---|---|---|---|
| | Auto | Manual | Auto | Manual | Auto | Manual |
| Rel | 6086 | 6086 | 6086 | 6086 | 6086 | 6086 |
| Rret | 3190 | 3371 | 3306 | 3679 | 3355 | 3737 |
| 0.0 | 0.6761 | 0.7166 | 0.7188 | 0.7896 | 0.6888 | 0.7893 |
| 0.1 | 0.4796 | 0.5082 | 0.4944 | 0.5601 | 0.4983 | 0.5479 |
| 0.2 | 0.3801 | 0.4156 | 0.3942 | 0.4581 | 0.3926 | 0.4486 |
| 0.3 | 0.3220 | 0.3465 | 0.3310 | 0.3883 | 0.3300 | 0.3997 |
| 0.4 | 0.2672 | 0.2960 | 0.2844 | 0.3317 | 0.2769 | 0.3329 |
| 0.5 | 0.2087 | 0.2315 | 0.2241 | 0.2552 | 0.2268 | 0.2631 |
| 0.6 | 0.1546 | 0.1708 | 0.1622 | 0.1849 | 0.1713 | 0.2079 |
| 0.7 | 0.0903 | 0.1033 | 0.0975 | 0.1236 | 0.1331 | 0.1520 |
| 0.8 | 0.0480 | 0.0567 | 0.0544 | 0.0727 | 0.0763 | 0.0894 |
| 0.9 | 0.0056 | 0.0175 | 0.0194 | 0.0300 | 0.0246 | 0.0422 |
| 1.0 | 0.0021 | 0.0038 | 0.0016 | 0.0014 | 0.0048 | 0.0024 |
| Avg | 0.2179 | 0.2397 | 0.2312 | 0.2688 | 0.2376 | 0.2779 |
| 5 | 0.5020 | 0.5102 | 0.5265 | 0.6082 | 0.5020 | 0.5796 |
| 10 | 0.4510 | 0.4714 | 0.4735 | 0.5551 | 0.4531 | 0.5490 |
| 15 | 0.4204 | 0.4422 | 0.4190 | 0.5034 | 0.4190 | 0.5007 |
| 20 | 0.3959 | 0.4235 | 0.4071 | 0.4694 | 0.3969 | 0.4602 |
| 30 | 0.3544 | 0.3748 | 0.3578 | 0.4211 | 0.3571 | 0.4156 |
| 100 | 0.2376 | 0.2516 | 0.2412 | 0.2794 | 0.2380 | 0.2843 |
| 200 | 0.1735 | 0.1856 | 0.1771 | 0.2039 | 0.1747 | 0.2064 |
| 500 | 0.1030 | 0.1114 | 0.1071 | 0.1192 | 0.1067 | 0.1227 |
| 1000 | 0.0651 | 0.0688 | 0.0675 | 0.0751 | 0.0685 | 0.0763 |
| RPr | 0.2761 | 0.2904 | 0.2763 | 0.3174 | 0.2872 | 0.3282 |

Table 4
Comparison of query likelihood (QL), InQuery, and our model for automatically and manually created TREC-4 queries. *Rel* is the number of relevant documents in the corpus, *Rret* is the number of relevant documents retrieved, *Avg* is the uninterpolated average precision, and *RPr* is the recall-precision breakeven point.

| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 0.580 | 0.580 | 0.588 | 0.600 | 0.610 | 0.612 | 0.608 | 0.608 | 0.612 |
| 10 | 0.549 | 0.555 | 0.541 | 0.549 | 0.547 | 0.553 | 0.551 | 0.553 | 0.551 |
| 15 | 0.501 | 0.499 | 0.498 | 0.498 | 0.499 | 0.499 | 0.491 | 0.494 | 0.499 |
| 20 | 0.460 | 0.459 | 0.455 | 0.458 | 0.461 | 0.464 | 0.467 | 0.465 | 0.470 |
| 30 | 0.416 | 0.417 | 0.420 | 0.424 | 0.421 | 0.420 | 0.422 | 0.425 | 0.422 |
| 100 | 0.284 | 0.285 | 0.280 | 0.279 | 0.277 | 0.275 | 0.276 | 0.274 | 0.271 |
| Avg | 0.278 | 0.277 | 0.275 | 0.274 | 0.272 | 0.271 | 0.270 | 0.268 | 0.265 |

Table 5
Non-interpolated precision for varying document cutoffs and average precision for a range of $\lambda_{prox}$ values on the TREC-4 manual queries.

### *4.3 Smoothing Experiments*

As we just discussed, smoothing of proximity nodes plays a very important role in our model, since the amount of data available to estimate these probabilities accurately is minimal. Therefore, we set out to find the proximity node smoothing parameter that yielded the best average precision. Figure 3 plots average precision against $\lambda_{prox}$, the smoothing parameter applied to the proximity nodes, for each query set previously discussed. Note, for non-proximity representation nodes (i.e. single term representations) we hold $\lambda$ fixed at 0.6.

The best value, in terms of average precision, is achieved at $\lambda_{prox} = 0.1$ for all data sets. Thus, data sparsity forces us to use a great deal of smoothing to achieve good results. Most of the query sets, with the exception of the TREC 4 automatic queries, are relatively insensitive to proximity smoothing. Therefore, shorter, less structured queries are less sensitive to proximity smoothing, whereas more verbose, highly structured queries are much more sensitive to smoothing. Zhai and Lafferty (2002) give analogous results.

Although average precision is an important metric, in some settings such as an interactive retrieval environment, it is desirable to have more relevant documents earlier in the ranked list of documents. As Table 5 illustrates, our system achieves higher precision at the beginning of the ranked list for larger values of $\lambda_{prox}$. However, this is only possible at the the cost of lower average precision. Although not explored here, varying the value of the smoothing parameter $\lambda$ for single term nodes will have an additional effect on performance. Thus, smoothing within our model allows great flexibility within different retrieval settings.

## 5 Conclusion and Future Work

We have presented an information retrieval model that combines the ability to use a rich structured query language with the formal probabilistic framework of language modeling. The `#WAND` structured operator was introduced for combining beliefs and a glimpse into potential smoothing options was explored. We also showed that our combined model is robust and capable of achieving better retrieval results than the InQuery system and reaffirmed that high quality structured queries have the potential to increase average precision.

There are several areas of potential future work. First, Turtle (1991) showed that combining multiple query representations gave a significant increase in average precision. Thus, a possible extension to our model would combine language modeling query representations with InQuery *tf.idf* based ones.

Second, as Section 4.3 discussed, a more detailed study of smoothing applied to the different types of query nodes is necessary. We looked at one method, but a number of options exist including automatic methods (Zhai and Lafferty, 2002).

Third, our model combines generative language models and Bayesian networks. As a result, it is not immediately clear what intuitive meaning, if any,

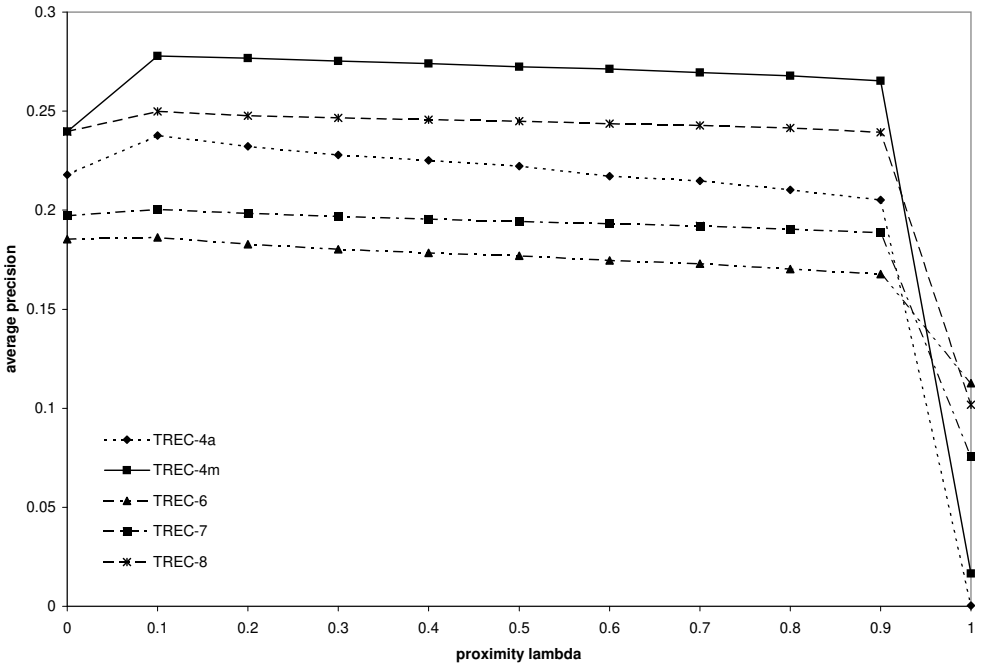


Fig. 3. Results of varying the smoothing parameter associated with proximity nodes.

the representation nodes in our network have. In the inference network framework, representation nodes correspond to the event that the representation is "about" the concept requested (Turtle, 1991). In our model the nodes correspond to the event that a representation was *generated* by a document's language model. However, another interpretation is possible. The original inference network model contains a layer of text representation nodes between the document and representation nodes. Each of these nodes is associated with the event that a given text representation is observed, where a text representation is defined to be the full text of the document. Furthermore, a one-to-one correspondence between these nodes and the document nodes was assumed, thus conflating the two layers into one. Figure 1 depicts this combined model, with the two layers of nodes represented as a single document node layer. In our model, text representations can be interpreted as *subsets* of text, such as single terms, ordered windows, etc, that have a one-to-one correspondence with the representation nodes. Then, rather than interpreting our language modeling probability estimates as the likelihood a representation is generated from a document, we can interpret them as the likelihood a certain text representation is observed in a document. Such an interpretation is more theoretically sound since our language modeling probabilities can be viewed as the likelihood of observing a representation when randomly sampling representations of the same form from a document. The difference between the two models is subtle, but must be addressed. Therefore, future work should explore more of the theoretical issues involved with these node interpretations.

Finally, in theory, any probability distribution can be used to estimate representation probabilities. InQuery's *tf.idf* method and our language modeling approach are just two of many possible techniques that yield good retrieval performance. A more detailed study of estimation techniques could prove to be beneficial for both inference network-based models and information retrieval systems in general. The inference network model provides a useful framework to carry out such experiments in.

We are currently implementing the Indri system (a version of Lemur) that will serve as the testbed for large scale experiments on language modeling and inference network integration.

## Acknowledgments

This work was supported in part by the Center for Intelligent Information Retrieval and in part by SPAWARSYSCEN-SD grant number N66001-02-1-8903 and by Advanced Research and Development Activity under contract number MDA904-01-C-0984. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect

those of the sponsor.

## References

Allan, J., Ballesteros, L., Callan, J., Croft, W., Lu, Z., 1995. Recent experiments with inquery. In: Proceedings of the 4th Text Retrieval Conference (TREC-4). pp. 49–63.

Allan, J., Callan, J. P., Croft, W. B., Ballesteros, L., Byrd, D., Swan, R. C., Xu, J., 1997. INQUERY does battle with TREC-6. In: Text REtrieval Conference. pp. 169–206.

Berger, A., Lafferty, J. D., 1999. Information retrieval as statistical translation. In: Research and Development in Information Retrieval. pp. 222–229.

Callan, J., Croft, W. B., Broglio, J., 1995. An overview of the inquery system. Information Processing and Management 31 (8), 327–344.

Callan, J. P., Croft, W. B., Harding, S. M., 1992. The INQUERY retrieval system. In: Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications. pp. 78–83.

Croft, W. B., Lafferty, J., 2003. Language Modeling for Information Retrieval. Kluwer Academic Publishers.

Greiff, W. R., Croft, W. B., Turtle, H., 1999. PIC matrices: a computationally tractable class of probabilistic query operators. ACM Transactions on Information Systems 17 (4), 367–405.

Hiemstra, D., 2002. Term-specific smoothing for the language modeling approach to information retrieval: the importance of a query term. In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press, pp. 35–41.

Ogilvie, P., Callan, J., 2002. Experiments using the lemur toolkit. In: Proceedings of the 2001 Text REtrieval Conference (TREC 2001). pp. 103–108.

Pearl, J., 1988. Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc.

Ponte, J. M., Croft, W. B., 1998. A language modeling approach to information retrieval. In: Research and Development in Information Retrieval. pp. 275–281.

Robertson, S. E., Walker, S., 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval. Springer-Verlag New York, Inc., pp. 232–241.

Rosenfeld, R., 2000. Two decades of statistical language modeling: Where do we go from here. Proceedings of IEEE 88 (8).

Turtle, H., Croft, W. B., 1991. Evaluation of an inference network-based retrieval model. ACM Transactions on Information Systems (TOIS) 9 (3), 187–222.

Turtle, H. R., 1991. Inference networks for document retrieval. Ph.D. thesis, University of Massachusetts.

Zhai, C., Lafferty, J., 2001a. Dual role of smoothing in the language modeling approach. In: Proceedings of the Workshop on Language Models for Information Retrieval.

Zhai, C., Lafferty, J., 2001b. A study of smoothing methods for language models applied to ad hoc information retrieval. In: Research and Development in Information Retrieval. pp. 334–342.

Zhai, C., Lafferty, J., 2002. Two-stage language models for information retrieval. In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. ACM Press, pp. 49–56.