# Indri: A language-model based search engine for complex queries
## (extended version)

*Trevor Strohman, Donald Metzler, Howard Turtle and W. Bruce Croft*
Center for Intelligence Information Retrieval
University of Massachusetts Amherst
Amherst, MA, 01003, USA
*strohman@cs.umass.edu*

**Keywords:** Search and Retrieval, Question Answering

### Abstract

Search engines are a critical tool for intelligence analysis. A number of innovations for search have been introduced since research with an emphasis on analyst needs began in the TIPSTER project. For example, the Inquery search engine introduced support for specification of complex queries in a probabilistic inference network framework. Recent research on language modeling has led to the development of Indri, a search engine that combines the best features of inference nets and language modeling in an architecture designed for large-scale applications. In this paper, we describe the Indri system and show how the query language is designed to support modern language technologies. We also present results demonstrating that Indri is both effective and efficient.

## 1. Introduction

Search and detection technology has been a focus of DARPA and ARDA research programs since the TIPSTER program began in the early 1990s (Harman 1992). A number of innovations have been developed in this research, resulting in very significant improvements in the effectiveness of search tools. The Inquery search engine (Callan *et al.* 1995), developed at the University of Massachusetts for the TIPSTER project, provided a query language capable of representing complex queries in a probabilistic framework and was used in a number of government and commercial applications.

In the years since Inquery was developed, there has been significant progress, both in terms of information retrieval (IR) research and in the development of other language technologies and applications, such as information extraction and question answering. These new technologies interact with search and provide new requirements for a search engine. In addition, the ever-increasing volume of searchable data requires that search engines be scalable to the level of multi-terabytes. In response to these requirements, we have recently developed Indri, a scalable search engine that combines the advantages of the inference net framework used in Inquery with the language modeling approach to retrieval that has been the subject of much recent IR research (Croft and Lafferty 2003). Indri is part of the ARDA-sponsored Lemur project[1].

The Indri search engine is designed to address the following goals:

- The query language should support complex queries involving evidence combination and the ability to specify a wide variety of constraints involving proximity, syntax, extracted entities, and document structure.
- The retrieval model should provide superior effectiveness across a range of query and document types (e.g. Web, cross-lingual, ad-hoc[2]).
- The query language and retrieval model should support retrieval at different levels of granularity (e.g. sentence, passage, XML field, document, multi-document).
- The system architecture should support very large databases, multiple databases, optimized query execution, fast indexing, concurrent indexing and querying, and portability.

In this paper, we describe the most important aspects of the Indri retrieval model, query language, and system architecture. We give some examples of the types of complex queries that can be supported, and illustrate the effectiveness and efficiency of the system using results from the 2004 TREC Terabyte track.

[1] http://www.lemurproject.org. Indri is available as a download from this site.

[2] "ad-hoc" refers to the TREC track that focuses on finding as many relevant documents as possible using queries of varying complexity

| Operator | Name | Description |
|---|---|---|
| #uwN($t_1$ $t_2$ …) | Unordered Window | Matches unordered text |
| #odN($t_1$ $t_2$ …) | Ordered Window | Matches ordered text |
| #any:*field* | Any operator | Finds any text appearing in a field named *field* |
| term.*field* | Field restriction | Finds the word *term* appearing in a field named *field* |
| #combine($q_1$ $q_2$ … ) | Combine operator | Combines beliefs from other operators to form a single score for a document |
| #weight( $w_1 q_1$ $w_2 q_2$ … ) | Weight operator | Combines beliefs from other operators to form a single score for a document, using weights to indicate which operators should be trusted most |
| #greater( *field* n )<br>#less( *field* n )<br>#equal( *field* n ) | Numeric range operators | Finds any occurrence of *field* with a numeric value less than, greater than, or equal to *n* |
| #date:before( d )<br>#date:after( d )<br>#date:between( b a ) | Date range operators | Finds any occurrence of a date occurring before or after a date, or between two dates. |
| #*operator*[*field*]($q_1$ $q_2$ …) | Extent retrieval | Evaluates *operator* on every occurrence of *field*; useful for passage retrieval |
| #filrej( c s ) | Filter reject | Evaluate the expression *s* only if *c* is not satisfied |
| #filreq( c s ) | Filter require | Evaluate the expression *s* only if *c* is satisfied |

Table 1: Indri query language operators

## 2. Retrieval Model

The retrieval model implemented in the Indri search engine is an enhanced version of the model described in (Metzler 2004b), which combines the language modeling (Ponte and Croft 1998) and inference network (Turtle and Croft 1991) approaches to information retrieval. The resulting model allows structured queries similar to those used in Inquery to be evaluated using language modeling estimates within the network, rather than *tf.idf* estimates. As in the original inference network framework, documents are ranked according to $P(I|D, \alpha, \beta)$, the belief the information need $I$ is met giving document $D$ and hyperparameters $\alpha$ and $\beta$ as evidence.

### 2.1. Document Representation

Typically, in the language modeling framework, a document is represented as a sequence of tokens (terms). Based on this sequence, a multinomial language model over the vocabulary is estimated. However, it is often the case that we wish to model more interesting text phenomena, such as phrases or the absence of a term. Here, we represent documents as multisets of binary feature vectors. The features can be nearly any interesting binary observation of the underlying text.

Traditional language modeling approaches are concerned only with word occurrences; this can be modeled by binary features vectors that are the length of the corpus vocabulary. Each word in the document is then encoded by a feature vector with a single non-zero entry representing that term.

However, we may wish to model document features that are not words; for instance, marking that a word appears at the end of a sentence or that it is capitalized. These facts can also be expressed as binary features. In a complex formulation of this model, each feature vector may have many non-zero entries, indicating all features that occurred at that position.

### 2.2. Multiple Bernoulli Models

Since our event space is now binary we can no longer estimate a single multinomial language model for each document. Instead, we estimate a multiple-Bernoulli model for each document, as in Model B of (Metzler 2004b). This overcomes the theoretical issues encountered in (Metzler 2004a). Note that the multiple-Bernoulli model imposes the assumption that the features ($r_i$'s) are independent, which may be a poor assumption depending on the feature set.

We take a Bayesian approach and impose a multiple-Beta prior over the model, $\theta$. The Beta is chosen for simplicity, as it is the conjugate prior to the Bernoulli distribution. Our belief at node $\theta$ is then:

$$Beta(\#(r_i,D)+\alpha_i \mid D \mid -\#(r_i,D)+\beta_i)$$

for each $I$ where $\#(r_i,D)$ is the number of times feature $r_i$ is set to 1 in document $D$'s multiset of feature vectors.

Indri estimates this model for the full text of each document. In addition, it creates models for each tagged subsection of a document, such as paragraphs and abstracts. These tagged regions are considered pseudo-documents, and can be retrieved as if they were full documents.

### 2.3. Inference Networks

The inference network approach to retrieval, first introduced in (Turtle and Croft 1991), formed the basis of the Inquery system, and is now a major component of the Indri retrieval model. The inference network model provides a principled way to combine many sources of evidence of document relevance.

In the inference network formulation, we suppose that a query is composed of a series of concepts, where these concepts may be terms, phrases, or more complex entities. We suppose that a document is relevant to a user precisely when it contains the concepts listed in the query.

It is important to note that a term concept is not analogous to a term simply appearing in a document; for instance, a document may contain the word 'terrorism' but not be about terrorism. However, the occurrence of 'terrorism' in a document provides evidence that this document is about terrorism. It is this evidence that we wish to estimate.

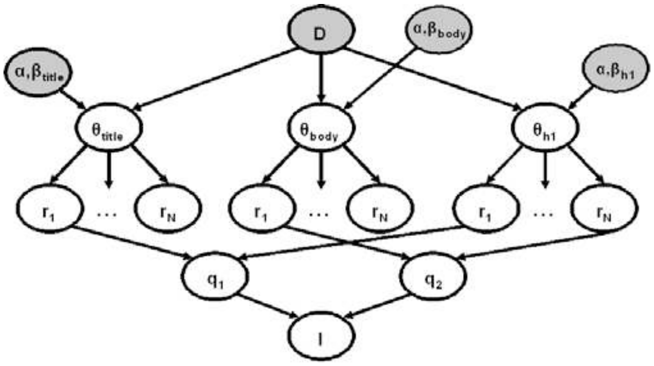

Figure 1: Sample inference network

Figure 1 shows an inference network. In this figure, *D* represents the document, which is an observed quantity. The language models $\theta$ are estimated based on hyperparameters $\alpha$ and $\beta$ combined with the observed document *D*. From these models, document features (represented by $r_i$ nodes) can be presented as evidence to concept nodes $q_i$, forming a basis of evidence of relevance at node *q*.

## 2.4. Representation Nodes

The $r_i$ nodes correspond to document features that can be represented in an Indri structured query. The belief at a given representation node is computed as:

$$P(r_i \mid D,\alpha,\beta) = \int_{\theta_i} P(r \mid \theta_i)P(\theta_i \mid D,\alpha_i,\beta_i)$$

which reduces to:

$$P(r_i \mid D,\alpha,\beta) = \frac{\#(r_i,D)+\alpha_i}{|D|+\alpha_i+\beta_i}$$

Furthermore, selecting $\alpha_i = \mu P(r_i \mid C)$ and $\beta_i = \mu(1-P(r_i \mid C))$ we get the multiple-Bernoulli model equivalent of the multinomial model's Dirichlet smoothing (Zhai 2004) estimate:

$$P(r_i \mid D,\alpha,\beta) = \frac{\#(r_i,D)+\mu P(r_i \mid C)_i}{|D|+\mu}$$

where $\mu$ acts as a tunable smoothing parameter.

## 2.5. Query Nodes

The goal of the query process is to establish the probability that a document is relevant to a query. In the figure, our belief that document *D* is relevant is found at node *I*. Between node *I* and the representation nodes ($r_i$) lie a set of query nodes that define how our belief of document relevance should depend on the document representation.

As an example, if our query is related to organized crime, the occurrence of words like 'crime', 'mob' and 'weapons' could indicate document relevance. Other words, like 'red' or 'car', are not likely to indicate anything about document relevance. Therefore, the query nodes for this kind of query would connect the node *I* to representation nodes for 'crime', 'mob' and 'weapons', but would leave the nodes for 'car' and 'red' unconnected.

The edges only define part of the function of query nodes, however. Query nodes combine evidence at the representation nodes to estimate the belief that a concept is expressed in a document. However, different query nodes may perform inference in different ways. The actual arrangement of the query nodes and the way the nodes combine evidence is dictated by the user through the query language.

# 3. Indri Query Language

Inference networks, combined with language feature models, give a solid theoretical basis for expressing information needs. In order to harness this model, Indri provides a query language that can express complex concepts.

The Indri query language is based on the successful Inquery structured query language. Both query languages are composed of operators, each of which can be considered a query node in an inference network. The Indri language contains the most popular operators from Inquery, along with many new operators that express concepts related to document structure.

## 3.1. Inquery operators

The Indri query language includes the window operators from Inquery. These operators allow the user to indicate that the location of query terms in a document affects relevance. The ordered window operator expresses that the terms should appear in a particular order in the document, while the unordered window operator merely requires terms to appear close together. Both operators have a distance parameter, *N*, that defines how close the terms need to be to each other.

Indri also includes the #combine and #weight operators, which are similar in usage to the #sum and #wsum operators from Inquery. These terms allow users to combine beliefs from a variety of other query nodes effectively. Mathematically, the #combine operator corresponds to the #and operator from Inquery, while the #weight operator corresponds to the #wand operator proposed by Metzler (Metzler 2004a).

Indri also incorporates the filter-require (#filreq) and filter-reject (#filrej) operators from Inquery, which are useful for filtering operations. The filter-require operator indicates that all relevant documents match a particular pattern; filter-reject indicates that relevant documents do not match a pattern.

### 3.2. Field operators

In addition to the Inquery operators, Indri adds operators for dealing with document structure. The simplest of these operators is the period operator, (used as *term.field*) which suggests that *term* is only relevant to the query if it appears within *field*.

Fields can be any tagged information from a document. Therefore, a field can be a large segment of a document, like a chapter; a smaller segment, like a paragraph; or just a few words, as in a noun phrase. A field can appear more than once in a single document.

For instance, the construction *wash.np* can be used to find the word ‘wash’ appearing in a noun phrase, (as in “car wash”) as opposed to as a verb.

By using the #any operator, Indri can search for the existence of a field in a document. This is especially useful when nested inside proximity expressions.

### 3.3. Extent retrieval

Indri also allows fields to be used as regions for scoring. In the #combine[field]( $q_1 \ldots q_n$ ) formulation, each occurrence of the tag *field* in the corpus is considered to be a separate document. The query #combine( $q_1 \ldots q_n$ )is then used to score and rank every one of these pseudo-documents. This provides a convenient way to perform passage retrieval on document structures, like paragraphs or sentences..

### 3.4. Date and numeric retrieval

Indri can be instructed to recognize numeric quantities, including dates. For referencing numeric quantities, Indri provides the #less, #greater and #equal operators. For ease of dealing with dates, Indri provides the #date:before, #date:after and #date:between operators.

## 4. System Architecture

The four goals for Indri put forth in the introduction are in conflict with one another. We wanted the system to be fast at indexing and retrieval, and still be able to handle complex data and information needs. In addition, this system was required to handle concurrent indexing and querying. Finally, as this system is meant to be usable in an academic setting, we wanted the code to be clear and easy to modify.

During the development of the system, we constantly made decisions that supported one goal at the expense of another. However, we believe that the Indri system has achieved a functional balance between its design goals.

### 4.1. Parsing

Indri comes with a variety of parsers for known document formats like TREC formatted text, XML, HTML, and plain text documents. These parsers translate the documents into an intermediate representation, called a ParsedDocument, that the indexer can store directly. For custom applications, the parsers can be bypassed entirely, and a hand-constructed ParsedDocument can be passed directly to the indexer.

We expect that the HTML and XML parsers will be the most used parsers in Indri, since they can extract document structure along with text. These parsers can be configured to pass tag information from documents on to the indexer so that this can be used for querying document structure.

The ParsedDocument contains a list of terms in the document and where they occur, and also contains information about fields in the document. Each field can contain a numeric representation of its contents. Additionally, the ParsedDocument contains the full unparsed text of the document, and the locations of all terms in this unparsed text. This unparsed text and the associated term positions can be used in retrieval scenarios where users may want to see query terms highlighted in the document.

Indri provides a small library of Transformation objects for parsing as well. Transformation objects transform a ParsedDocument into another ParsedDocument; therefore, they can be easily chained together. The Indri system includes implementations of the Porter and Krovetz stemmers as Transformations, and also includes a stopword removal Transformation. More of these may be added in the future.

### 4.2. Indexing

The indexing system builds compressed inverted lists for each term and field in memory. Periodically, as memory gets scarce, this data is flushed to disk. The data that is written to disk is self-contained: it contains all information necessary to perform queries on that data. In a sense, an Indri index can be considered a set of smaller indexes. The retrieval system has been written to be able to query many indexes together.

The indexer also stores a copy of the incoming document text in compressed form. This text is commonly used to produce document snippets at retrieval time.

The index subsystem is capable of storing any text that can be represented in Unicode.

### 4.3 Retrieval

When a query is submitted to the Indri system, it is parsed into an intermediate query representation. This intermediate representation is then passed through a variety of query transformations. Some of these transformations are for performance reasons; for instance, expressions that do not use term proximity information are selected for a slightly faster execution path. Other transformations expand complex query operators into a series of simpler internal operators for evaluation.

Indri is capable of evaluating queries against many indexes simultaneously, and indexes do not need to reside on the same machine. In the event that the indexes are not on the same machine as the query director process, the query director connects to an Indri daemon on the remote machine which performs some of the query processing.

Query evaluation proceeds in two phases. In the first phase, statistics about the number of times terms and phrases appear in the collection are gathered. In the second phase, the statistics from the first phase are used to evaluate the query against the collection.

The query evaluation code in Indri incorporates the max-score optimization in order to speed query evaluation (Turtle and Flood 1995).

### 4.5 Concurrency

Indri supports multithreaded operation, where document insertions, deletions and queries can be processed simultaneously. This recent addition to the engine has been added to support retrieval against dynamic collections of data, like news feeds.

In the implementation, we have been careful to arrange data such that locks are held as briefly as possible. Our goal has been to never force an operation to block for longer than a second. In most cases we are able to achieve this bound.

Indri stores indexed documents in a repository, which is composed of an ordered set of indexes. At any one time, only one of these indexes can receive new documents; all other indexes are read-only. The index that receives new documents resides in main memory, and contains only a small fraction of the total indexed documents. This means that the majority of indexed documents are in read-only indexes, which simplifies concurrent execution significantly.

When the active in-memory index fills, it is marked read-only and written to disk asynchronously. While the write is taking place, a new in-memory index is created to receive any incoming documents. When the write completes, the old in-memory index is deleted, and the copy on disk takes its place. During this index write, queries can continue to run, and documents can still be indexed. A similar process is used to merge many indexes together into a single index.

## 5. Query Language Examples

The operators in the Indri query language allow users to construct extremely detailed queries. In contrast to the short keyword queries that most systems encourage, the Indri query language is capable of expressing the complexity in real information needs.

We expect that, in general, users will not form these queries directly; rather, they will interact with a domain-specific interface that will form these complex queries based on user input. In this section, we give examples of what these automatically generated queries might look like.

Consider the following information need: "I want paragraphs from news feed articles published between 1991 and 2000 that mention a person, a monetary amount, and the company InfoCom."

This need can be expressed in the following Indri query:

```
#filreq(
  #band( NewsFeed.doctype
    #date:between(1991 2000) )
  #combine[paragraph](
    #any:person
    #any:money InfoCom ) )
```

This query requires that the index be built with person, money, doctype, date and paragraph tags.

In a similar construction, a user may wish to search for reports after 1995 mentioning the person "Elashi" interacting with any company.

```
#filreq(
  #band( FieldReport.doctype
    #date:after(1995) )
  #combine( #person(Elashi)
    #any:company ) )
```

A user may be interested in all articles by Thomas Friedman published before 2000 that discuss the Oil for Food program:

```
#filreq(
  #band( #ow3( Thomas Friedman ).author
    #date:before( 2000 ) )
  #ow5( oil for food ) )
```

As these examples show, the Indri query language has been designed to leverage document structure as well as text, giving users added ability to hone in on relevant documents.

## 6. Effectiveness and Efficiency

Even though Indri supports query operators that allow users to express very complex information needs, Indri can achieve solid results even with short queries. In the 2004 TREC Terabyte track (Metzler *et al.* 2005), the Indri entry from the University of Massachusetts was the most effective title query entry. Not only did Indri perform well without any use of Indri query language operators, but the query effectiveness improved significantly by using automatically generated query proximity expressions.

The TREC Terabyte track data consists of a 426GB collection of web documents, known as the GOV2 collection, and a set of 50 queries. Table 2 shows the average precision of the best title run at the conference (submitted by UMass), and the best run with all query data (submitted by Glasgow). Since the conference, we have fixed bugs in our system in order to achieve the higher average precision values shown in the right column.

Indri evaluated simple queries in approximately 1.3 seconds each over the GOV2 collection using a 6 machine cluster. Recent research into query optimization has enabled Indri to run these same queries in 1.7 seconds using the same data on a single Pentium 4 2.6 GHz machine. We expect that these times will continue to come down as research continues.

## 7. Conclusion

Inquery started a tradition at the University of Massachusetts of building information retrieval systems that are simultaneously useful for academic, intelligence and corporate tasks. The powerful inference network framework made Inquery useful for research, while making it possible for non-academics to express complex information needs.

The University of Massachusetts has continued this tradition with Indri, a system that combines the inference network framework with new theoretical advances in language modeling. In addition, Indri provides new query language constructs incorporating fields, tags and numbers to support query activity in question answering and cross-lingual retrieval. Indri also handles much larger collections than Inquery, and is capable of scaling up to clusters of machines for efficient retrieval. Indri also includes new support for dynamic collections.

The University of Massachusetts is committed to supporting the wide use of the Indri system, as it did with the Inquery system. To this end, it is under continued development to support new needs that we are just coming to understand. It is available as an open source system, so it is free to pick up and use, although Indri will be available in a commercial package from a third-party in case enterprise support is needed.

## Acknowledgements

This work was supported in part by the Center for Intelligent Information Retrieval and in part by Advanced Research and Development Activity and NSF grant #CCF-0205575 . Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

## References

J. P. Callan, W. B. Croft, and J. Broglio. TREC and TIPSTER Experiments with INQUERY. In *Readings in Information Retrieval*, ed. Karen Sparck Jones and Peter Willett, 436-445. San Francisco, CA: Morgan Kaufmann, 1997. [Originally published in: *Information Processing & Management* 31 (1995): 327-332.

Croft, W.B. and Lafferty, J. eds., *Language Modeling for Information Retrieval*. Kluwer International Series on Information Retrieval, Volume 13, Kluwer Academic Publishers, (2003).

D. Harman, The DARPA TIPSTER Project, SIGIR Forum, 26, 26-28 (1992).

D. Metzler and W.B. Croft. Combining the language model and inference network approaches to retrieval. *Info. Proc. and Mgt.* 40(5):735-750, 2004.

D. Metzler, V. Lavrenko and W.B. Croft. Formal multiple Bernoulli models for language modeling. In *SIGIR 2004*, pp. 540-541.

D. Metzler, T. Strohman, H. Turtle, and W. B. Croft. “Indri at TREC 2004: Terabyte Track,” to appear in the Online Proceedings of 2004 Text REtrieval Conference (TREC 2004).

J. Ponte and W. B. Croft, A language modeling approach to information retrieval. In *SIGIR 1998*, pp. 275-281.

H. Turtle and W. B. Croft, Evaluation of an inference network based retrieval model. *Trans. Inf. Syst.*, 9(3):187-222, 1991.

H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Info. Proc. and Mgt.*, 31(6):831-850, 1995.

C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179-214, 2004.

| Query type | Best results from TREC 2004 | | Best Indri results (recent experimentation) | |
|---|---|---|---|---|
| | Average Precision | Precision at 10 | Average Precision | Precision at 10 |
| Short queries (title) | 0.2838 (UMass) | 0.5510 (UMass) | 0.2874 | 0.5663 |
| Long queries (title+desc+narr) | 0.3088 (Glasgow) | 0.6163 (Glasgow) | 0.3293 | 0.6306 |

Table 2: Results from tests with the TREC GOV2 collection