Generating Queries from User-Selected Text

Chia-Jung Lee
Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts, Amherst
cjlee@cs.umass.edu

W. Bruce Croft
Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts, Amherst
croft@cs.umass.edu

ABSTRACT

People browsing the web or reading a document may see text passages that describe a topic of interest, and want to know more about it by searching. Manually formulating a query from that text can be difficult, however, and an effective search is not guaranteed. In this paper, to address this scenario, we propose a learning-based approach which generates effective queries from the content of an arbitrary user-selected text passage. Specifically, the approach extracts and selects representative chunks (noun phrases or named entities) from the content (a text passage) using a rich set of features. We carry out experiments showing that the selected chunks can be effectively used to generate queries both in a TREC environment, where weights and query structure can be directly incorporated, and with a "black-box" web search engine, where query structure is more limited.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Performance, Experimentation

Keywords

Query generation, query representation, text segments

1. INTRODUCTION

Annotation, such as circling or underlining text passages, is a common action that people do when reading documents. Such annotations, which are becoming more common in various tablet applications, can help improve understanding [1, 22] and, more importantly for this study, can reveal the underlying interests with respect to a specific document[10]. Golovchinsky et al [10] described how information seekers use annotations to markup relevant passages, and their experimental results in [10] showed that queries constructed from the annotated texts can be very effective. Similarly, when browsing web pages, people sometimes express their interests in text passages by implicitly hovering the mouse or explicitly selecting (highlighting) text, which could trigger potential subsequent web searches. Specifically, according to Cheng et al [9], a considerable portion (19.3%) of queries are issued right after users have browsed web pages, and 66% of such search requests come from the page content. This observation indicates that, similar to the annotation behavior described in [10], a user may find text passages of a web page interesting or unclear, and formulate their own queries based on such passages to do a deeper search after reading.

Manual query construction based on text passages is common; however, such formulation can involve considerable effort for users and an effective search is not guaranteed. To this end, Cheng et al [9] used search engine log history to rank possible queries issued after reading a web page and suggested them to users. This query suggestion-based method benefits from the diversity of crowd knowledge, but can be less focused on the content of the current page. Another approach is to use relevance feedback [24, 28] or "more-like-this" [26] techniques to expand the original query with significant words from the entire page. As suggested by [10], this approach can select expansion terms that, although statistically representative of the document, do not accurately reflect specific passages of interest in many cases.

In this paper, we propose techniques for generating queries from user-selected or annotated text passages. More precisely, we assume that the selected content of the document or web page is made up of a consecutive sequence of words (i.e., a text segment). A user can select any arbitrary text segment of interest while browsing, and we then automatically generate queries based on that text segment. This approach provides an alternative query mode that could potentially alleviate some of the burden of query construction. For example, Lee et al [21] showed that the average time spent by the subjects to formulate one query for a short piece of news segment is about 34.4 seconds, compared to 2.0 seconds required by the query generation algorithm proposed in [21]. Moreover, the nature of annotations such as selecting passages requires little effort from the users. In this paper, we describe a detailed study of machine learning-based retrieval techniques for both TREC and web search environments, rather than the more ad-hoc techniques used in [10].

To generate effective queries, we propose to identify important noun phrases and named entities, called "chunks" in this paper, within the selected text segment as the ba-

IIiX 2012, Nijmegen, The Netherlands

Table 1: Example of a text segment and its corresponding set of chunks.

Text Segment: TS	Chunks: C
Knee joint replacement may be recommended for: Severe arthritis (osteoarthritis or rheumatoid arthritis) of the knee that has not gotten better with medicine, injections, and physical therapy after 6 months or more of treatment. Your doctor may recommend knee replacement for these problems: Inability to sleep through the night because of knee pain. Knee pain that has not improved with other treatment. Knee pain that limits or keeps you from being able to do your normal activities, especially your daily activities such as bathing, preparing meals, household chores, and other things. Some tumors that affect the knee	household chores, knee replacement, knee pain, Severe arthritis, osteoarthritis, rheumatoid arthritis, injections, tumors, joint replacement, bathing, Inability, physical therapy, normal activities, knee, meals, daily activities, other treatment, medicine, treatment, 6 months

sic building blocks for query formulation. This task mainly involves extracting chunks, estimating chunk importance, and generating queries using important chunks. We will discuss each of these steps in detail in the following sections. Briefly, a Conditional Random Field (CRF) model is learned for estimating chunk importance, where various features that are useful in the determination of retrieval performance are considered. Query generation is subsequently carried out by considering different weighting techniques on the important chunks. We evaluate our approach on both a TREC test collection and a real web test collection using the search API provided by a commercial search engine. Experiments show that retrieval performance can be significantly improved relative to several baseline approaches including document-based term selection methods. These effectiveness improvements are consistent across different collections.

Another critical aspect of this paper is to construct a reliable test collection for evaluation. In particular, since the approach is evaluated in both a TREC and a "black-box" web search environment, a well-designed test collection of queries is needed for such cross comparison. To address this, we use only text segments from documents that are judged relevant to a specific query topic in the TREC Gov2 collection. Consequently, the "information needs" associated with the text segments should correspond to the TREC query topics, and thus the existing relevance judgments for the Gov2 collection can be used directly. An example of the alignment can be found in Table 1, where the text segment corresponds to the query numbered 812 in TREC Gov2[1]. This approach is designed to simulate the situation where a user selects a text passage of interest from a document, based on which we generate queries automatically.

The rest of this paper is organized as follows. Section 2 specifies the problem and provides a high-level overview of the key components in this paper. Sections 3 and 4 show the details of the process of query generation. We present our experimental results in Section 5. Finally, Section 6 discusses related work and Section 7 concludes the paper.

2. PROBLEM SPECIFICATION

In this section, we outline a general framework for generating queries from a user-selected text segment TS. Figure 1 illustrates a high-level overview of the process of query generation, where our problem can be decomposed in to three main components including "Chunk Extraction", "Chunk Selection", and "Query Generation".

"Chunk Extraction" involves the work of splitting original text segment TS into a set of chunks $C = \{c_1, c_2, ..., c_n\}$,

[1]Description query 812: What conditions lead doctors to recommend total knee replacement surgery and what complications can result from such surgery?

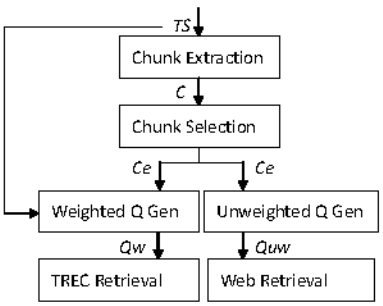

Figure 1: Framework of generating effective queries including "Chunk Extraction", "Chunk Selection", and "Query Generation".

where each chunk is a noun phrase or a named entity in our design. The use of noun phrases or named entities as the minimum semantic building blocks (i.e., chunks) has proven to be reliable in past research on information retrieval [6][5, 2, 8] and natural language processing[11]. Previous work suggested that, to deal with retrieval using long texts, identifying important words [5, 29] can be very effective. However, directly applying these approaches to the problem of generating queries from text segments may be computationally infeasible. The length of text segments is typically much longer than the query texts used in previous research, such as TREC description queries. Instead of using a word, considering a "chunk" as the unit for the query construction reduces computational burdens while keeping most of the interesting content such as preserving text structure and reducing topic drift. Table 1 shows an example of splitting an original text segment into a set of chunks.

The next step "Chunk Selection" selects a set of potentially effective chunks C_e from the set C. The "Query Generation" step then combines C_e using a number of different strategies to produce the final query. We hypothesize that using predicted effective chunks for query generation will have a positive impact on retrieval performance. In the following, we will explicitly describe the techniques used in "Chunk Selection" and "Query Generation" in Section 3 and Section 4, respectively.

3. CHUNK SELECTION

In this section, we propose the core techniques used for selecting a set of effective chunks C_e. The techniques are based on two approaches. First, we provide a simple frequency-based approach which selects chunks using simple statistical indicators. Our main approach, the learning-based approach, takes various features into account and determines effective chunks based on the models learned.

3.1 Frequency-based approach

As suggested by [6], we statistically measure the effectiveness of chunks based on simple frequency counting. We submit each chunk to a web search API and record the corresponding number of returned results as $N = \{n_1, n_2, ..., n_n\}$. Intuitively, as the web encompasses a huge amount of data, c_i is considered more important than c_j if $n_i < n_j$, following the common belief in the effectiveness of term inverse document frequency. Based on the number of returned results, we select the top k most infrequent chunks as the set of effective chunks C_e, and denote those as $WeightC(k)$. In Section 5.3, we show the impact of different numbers of chunks selected.

3.2 Learning-based approach

Alternatively, we use a learning-based approach for C_e construction. We use CRF models to identify important chunks in C, and construct C_e based on a number of different strategies and variants of CRF. In the following, we first describe the CRF models adopted. The construction of C_e is detailed in the next step, and finally, the set of features is described.

3.2.1 CRF-perf models

To identify important chunks in C, we regard the task as a labeling problem where each chunk $c_i \in C$ is assigned a label of "keep" or "don't keep". A labeling configuration $L = \{l_1, l_2, ..., l_n\}$ then decides a corresponding chunk combination, where $l_i = 1$ and $l_i = 0$ means "keep" and "don't keep" respectively. In this paper, we adopt the novel CRF model (CRF-perf) proposed by Xue et al [29] to solve the labeling problem. CRF-perf provides a formally well-founded framework to model the distribution of label sequences. More importantly, CRF-perf has the advantage of directly optimizing the expected retrieval performance rather than the labeling accuracy. Based on CRF-perf, the probability of being effective is predicted for all chunk combinations, including the special case where only a single chunk is labeled "keep".

Based on CRF-perf, the problem of identifying important chunks can be tackled using a training set containing instances $\{C, \{L, m(L)\}\}$, where C denotes the set of chunks extracted from a text segment and $\{L, m(L)\}$ represents the corresponding set of labeling configurations. Specifically, $\{L, m(L)\}$ contains all possible chunk combinations for C, and $m(L)$ denotes the retrieval performance measured by function $m(.)$ for L.

Given the chunks C observed, the probability of a specific labeling configuration $P(L|C)$ is calculated in Equations 1 and 2, where f_j associated with weight λ_j denotes a feature extracted from C and L. $Z(C)$ represents a normalizer over all labeling configurations for a specific C.

$$P(L|C) = \frac{exp(\sum_{j=1}^{J} \lambda_j f_j(L, C))}{Z(C)} \quad (1)$$

$$Z(C) = \sum_{L} exp(\sum_{j=1}^{J} \lambda_j f_j(L, C)) \quad (2)$$

In the training phase, the model parameters $\theta = \{\lambda_j\}$ are learned based on the objective function in Equation 3, where $\sum_L P(L|C)m(L)$ is the expected retrieval performance over the labeling configurations for a give C. As shown in Equation 3, the objective function directly optimizes the expected retrieval performance for each C in the training set. The corresponding log-likelihood expression of Equation 3 is calculated in Equation 4, where $R = \sum_j \lambda_j^2 / 2\delta^2$ represents a regularizer avoiding unbounded parameter values. To compute the final value of parameter λ_j that maximizes retrieval performance of the training set, one needs to take partial derivatives on Equation 4 with respect to each λ_j; the details of computation can be referenced in [29].

$$Obj(\theta) = \prod_{C} \sum_{L} P(L|C)m(L) \quad (3)$$

$$l(\theta) = \sum_{C} log \sum_{L} exp(\sum_{j} \lambda_j f_j(L, C))m(L) - \sum_{C} logZ(C) - R \quad (4)$$

We use the CRF-perf framework in two different ways. First, the training set includes all possible chunk combinations for a given C as described previously. In the testing phase, the probability distribution over the set of chunk combinations is predicted as $P(L'|C', \theta)$, given C' extracted from an unseen text segment TS'. The probability $P(L'|C', \theta)$, as a result, tells how likely the corresponding chunk combination is to be effective and yields good retrieval performance.

Another way of using the CRF-perf framework is to directly estimate the probability distribution for every single chunk $c_i \in C$. This implementation is straightforward in that only labeling configurations with exactly one $l_i = 1$ are considered for training. This approach is motivated by efficiency concerns as using single chunks could potentially save a significant amount of computation relative to the previous model. Therefore, in the testing phase, the probability distribution $P(L'|C', \theta)$ directly estimates the effectiveness of a single chunk.

We denote the two ways of applying CRF-perf as models CRF_{CombC} and $CRF_{SingleC}$. For succinct presentation, we simplify the notations by denoting the probability distribution $P(L'|C', \theta)$ as $P_c(.)$ for the model CRF_{CombC}, and $P_s(.)$ for the model $CRF_{SingleC}$.

3.2.2 Selecting the set of effective chunks

In the following, we introduce several ways to construct the final chunk set C_e, including $CombC$, $CombC$+$TopC(2)$ and $TopC(k)$.

Based on the model CRF_{CombC}, we directly select the best chunk combination $CombC$ (i.e. the chunk combination with the highest P_c) to construct C_e. In addition, we are interested in testing if incorporating other top-performing chunk combinations could make C_e more effective for search. In particular, we select the two single chunks with the highest probabilities P_c as the two top-performing chunk combinations. Such a selection, denoted as $CombC$+$TopC(2)$, is adopted in [5] where the best results showed that "2" is a reasonable choice.

C_e construction based on $CRF_{SingleC}$ may require a more sophisticated algorithm due to the possibly degraded effectiveness of $CRF_{SingleC}$ (a large portion of chunk combinations are ignored in $CRF_{SingleC}$). Specifically, we propose Algorithm 1 to construct C_e which contains the top k effective chunks for each query topic. One of the merits of Algorithm 1 is that the parameter k is automatically optimized and determined for each topic. In Section 5, it is

Table 2: Features used in this paper.

Source	Feature	Description
Text Segment	`tf`	Term frequency for concept e in the text segment
Document	`ptf`	Term frequency for concept e in the page
	`pdf`	Paragraph frequency for concept e in the page
TREC Gov2	`cf`	Term frequency for concept e in TREC-gov2
	`df`	Document frequency for concept e in TREC-gov2
Google ngram	`gf`	ngram count of concept e in Google ngram
MS Web N-Gram	`jp-ch`, `jp-co`	Joint probability of chunks and concepts
	`cp-ch`, `cp-co`	Conditional probability of chunks and concepts
MSN Query Log	`ql-exact`	Count of exact matches of a concept e and a query in the log
	`ql-exist`	Count of times concept e occurs within a query in the log
Wiki Title	`wiki-exact`	Count of exact matches of a concept as a Wikipedia title
	`wiki-exist`	Count of times concept e occurs within a Wikipedia title
Other	`length`	Total number of words a chunk contains
	$\mathtt{n_i}$	Number of results returned from search engine API given a chunk
	`cnt-ch`	Number of chunks in a chunk combination

Algorithm 1 Chunk selection

Input: the set of chunks $C = \{c_1, c_2, \ldots, c_n\}$.
Output: the set of effective chunks C_e.

```
find c_k ∈ C such that P_s(c_k) = max{P_s(c_i) | c_i ∈ C}
C_e ← {c_k}
C ← C − {c_k}
while C ≠ ∅ do
  find c_k ∈ C such that P_s(c_k) = max{P_s(c_i) | c_i ∈ C}
  if min_{c_j ∈ C_e}{P_s(c_k)/P_s(c_j)} > TH then
    C_e ← C_e ∪ {c_k}
    C ← C − {c_k}
  else
    C ← ∅
  end if
end while
return C_e
```

shown that the automatic k determination has a significant positive impact on retrieval performance. We now describe in detail the chunk selection algorithm. In the beginning, we keep an empty set of C_e and add the most effective chunk c_k (i.e., max $P_s(c_k)$), meaning that C_e at least contains one single chunk. The general idea of Algorithm 1 is considering whether to include the next most effective chunk or not, based on how much effectiveness could be gained by inclusion of that chunk. For example, for c_k with the second highest $P_s(c_k)$, we shall include it as part of C_e if its relative effectiveness gain ratio is greater than a threshold. Mathematically, the inclusion takes place if $P_s(c_{(2)})/P_s(c_{(1)}) > TH$[2]; otherwise, we discard all chunks with probabilities less than or equal to $P_s(c_{(2)})$. The algorithm traverses all chunks in C and terminates either when all chunks are visited or when the criteria of effectiveness gain ratio is break at some point.

3.2.3 Feature Set

In this section, we describe the features of a chunk, a unigram, and a bigram used in this paper. A unigram is a single word of a chunk, while a bigram is a consecutive sequence of 2 words from a chunk. By considering the features of unigrams and bigrams, we can capture the characteristics of both independent query words and dependent word sequences contained in a chunk. We refer to both a unigram or a bigram as a "concept" following [7]. Concerning the design of the feature set, we consider various types of collection sources from which useful statistical features can be extracted. In the following, we detail each of the sources and the features used, which are summarized in Table 2. Some of the features are adopted from [7].

Conventional `tf` and `df` counts are shown to be effective in designing features. Specifically, we compute `tf` and `df` counts of a concept in sources available from the benchmark collection. The sources include three aspects of different granularity: the original text segment, the document from which the text segment is selected, and the entire TREC Gov2 collection.

In addition to the benchmark collection, we are interested in using other large collections which may provide better coverage of terms. Large collections could provide better smoothing especially for sparse terms or phrases. Different collections also provide different domain knowledge for estimating term importance. In this work, we use 4 publicly available large collections as extra sources, including the Google n-grams corpus[3], Microsoft Web N-Gram[4], a sample of an MSN query log[5], and a snapshot of Wikipedia article titles[6].

The Google n-grams corpus, which contains the frequency counts of English ngrams generated from approximately 1 trillion word tokens from publicly accessible web pages, is used for estimating the frequency for a concept.

The Microsoft Web N-gram services provide access to real-world web-scale data using a cloud-based platform. Specifically, we use the N-gram models, constructed based on a web snapshot taken in April 2010, for estimating the conditional and joint probabilities for a concept as well as a chunk.

A large sample of query logs may help identify the importance of a concept based on the crowd knowledge of search engine users. The MSN Log, which consists around 15 million queries, is adopted to count the number of times a concept "appearing in" or "being exactly" a query in the log history.

[2] $P_s(c_{(i)})$ represents P_s of the i-th effective chunk

[3] Linguistic Data Consortium catalog
[4] http://web-ngram.research.microsoft.com/
[5] Available as a part of Microsoft 2006 RFP dataset
[6] http://download.wikimedia.org/enwiki/

The last additional collection is a snapshot of the Wikipedia article titles, which contains about 3 million English articles. The features for this collection are the number of times a concept "appears in" or "being exactly" an article title in Wikipedia.

Finally, we use some statistical indicators to describe a chunk. `length` simply records how many words there are in a chunk. $\mathtt{n_i}$, as introduced in Section 3, represents the number of search results using a chunk as query. Note that all the features described above are with respect to "a chunk". When it comes to the features of a chunk combination, the feature values are summed together over the chunks in that combination, and an extra `cnt-ch` feature will record how many chunks the combination contains.

4. QUERY GENERATION

To test the generality of our approach, we propose two different kinds of queries designed respectively for retrieval tasks in the TREC and web environments. Weighted queries Q_w are generated for the TREC environment, as weights and query structure can be directly incorporated into the retrieval model used for this collection. Unweighted queries Q_{uw}, on the other hand, are designed for the web search engine environment as query structure is more limited under a "black-box" framework.

4.1 Weighted query generation

In this step, 4 kinds of weighted queries are generated based on the 4 kinds of C_e proposed previously. For an original text segment TS, TS_n denotes the corresponding TS with no stopwords. The standard INQUERY stopword list[3] which contains 418 stopwords is used for the removal process.

$TS_n + WeightC(20)$: This type of query is constructed based on the statistical approach, where each $c_i \in C_e$ is associated with the number of returned results n_i from the search API. We then weight each chunk in inversely proportion to n_i. Specifically, if we take the reciprocal of n_i as r_i, the weight w_i of chunk c_i is computed as $r_i / \sum_i r_i$. The generated query is then constructed as,

$$Q_w \leftarrow \alpha(TS_n) + (1-\alpha)(\textstyle\sum_i w_i c_i)$$

$TS_n + CombC$: The query is generated by combining the best chunk combination (max P_c) with TS_n, resulting in,

$$Q_w \leftarrow \alpha(TS_n) + (1-\alpha)(CombC)$$

$TS_n + CombC + TopC_w(2)$: The kind of query is motivated by incorporating two effective single chunks in $TopC(2)$, each of which is associated with a probability $P_c(c_{(i)})$ estimated based on the CRF_{CombC} model. We highlight the importance of each single chunk with a weight w_i, which is equal to $P_c(c_{(i)}) / \sum_{i=1}^{2} P_c(c_{(i)})$. We denote the two chunks $\sum_{i=1}^{2} w_i c_i$ as $TopC_w(2)$, and modify the query based on $TS_n + CombC$ as follows,

$$Q_w \leftarrow \alpha(TS_n) + \beta(CombC) + (1-\alpha-\beta)(TopC_w(2))$$

$TS_n + TopC(k)$: The final approach is to generate the query using model $CRF_{SingleC}$ and Algorithm 1, and is used in the form of,

$$Q_w \leftarrow \alpha(TS_n) + (1-\alpha)(TopC(k))$$

Table 3: Examples of 4 kinds of weighted queries (#comb is short for #combine).

$TS_n + WeightC(20)$
#weight(0.8 #combine(TS_n) 0.2 #weight(.0021 #comb(treatment) .0161 #comb(normal activities) .0097 #comb(knee) .0179 #comb(physical therapy) .0026 #comb(medicine) .0747 #comb(osteoarthritis) .0643 #comb(rheumatoid arthritis) .0072 #comb(meals) .0445 #comb(Knee pain) .0892 #comb(Severe arthritis) .0423 #comb(joint replacement) .0490 #comb(tumors) .0272 #comb(bathing) .3248 #comb(household chores) .0906 #comb(knee replacement) .0566 #comb(injections) .0234 #comb(Inability) .0065 #comb(daily activities) .0029 #comb(other treatment) .0445 #comb(knee pain)))
$TS_n + CombC$
#weight(0.8 #comb(TS_n) 0.2 #comb(Knee pain household chores injections joint replacement knee pain knee replacement osteoarthritis rheumatoid arthritis tumors))
$TS_n + CombC + TopC_w(2)$
#weight(0.8 #comb(TS_n) 0.1 #comb(Knee pain household chores injections joint replacement knee pain knee replacement osteoarthritis rheumatoid arthritis tumors) 0.1 #weight(.7012#comb(knee replacement) .2988#combine(joint replacement)))
$TS_n + Top(k)$
#weight(0.8 #comb(TS_n) 0.2 #comb(knee pain knee replacement household chores Knee pain joint replacement))

Table 4: Examples of 2 kinds of unweighted queries.

$TopC(2)$
knee replacement joint replacement
$TopC(k)$
knee pain knee replacement household chores Knee pain joint replacement

To give a clear summary, Table 3 shows examples of the 4 types of queries generated, based on the text segment example in Table 1. $\alpha \in [0,1]$ and $\beta \in [0,1]$ are free parameters specifying the importance for each model, and are usually set to 0.8 and 0.1 according to [5, 23]. While α and β are fixed in this paper, it suffices the purpose of comparing results between different query models. We consider exploring different values of these parameters for maximizing performance as future work.

4.2 Unweighted query generation

We propose 2 methods for generating unweighted queries. Table 4 shows 2 corresponding query examples.

$TopC(2)$: Based on the weighted query generated using model CRF_{CombC}, the unweighted query is generated with the motivation of using the highest weighted chunks from $TS_n + CombC + TopC_w(2)$. A shorter query is also preferred for its tendency to be more effective than a longer one. Interestingly, we find that the chunks appearing in $TopC(2)$ will also appear in $CombC$ with no exception for all topics. And, of course, all chunks in $TopC(2)$ will also be included

in TS_n. The unweighted query is then simply,

$$Q_{uw} \leftarrow TopC(2)$$

$TopC(k)$: Using the same rationale, based on the model $CRF_{SingleC}$, we generate another type of unweighted query,

$$Q_{uw} \leftarrow TopC(k)$$

5. EXPERIMENTS

5.1 Experimental Setup

Experiments are conducted on the TREC Gov2 collection, which contains 25,205,179 documents and 150 query topics numbered from 701 to 850. Indexing is done using Indri[7], which supports term weighting schemes and flexible query structure. The documents are stemmed with the Krovetz stemmer[16] and there are no stopwords removed during the indexing process.

Algorithm 2 TS Set Selection

Input: Query topics QID, Relevance judgement RJ.
Output: Set $T = \{(qid, TS, Page)\}$ and $|T| = 50$.

```
T ← ∅;
while |T| ≤ 50 do
    randomly select qid ∈ QID;
    get the relevant documents RJ^r_qid for qid;
    select a document reldoc from RJ^r_qid;
    select a text segment textseg from reldoc;
    T ← T ∪ {(qid, textseg, reldoc)};
    QID ← QID − {qid};
end while
return T;
```

The query set used for evaluation is composed of 50 text segments, which are selected as specified in Algorithm 2. First we randomly choose 50 query topics (from the 150 topics) and locate their relevant documents in TREC Gov2 using the existing relevance judgments. Then for each topic, we manually identify a text segment from a relevant document, under the constraints that (1.) the text segment satisfies the specifications in the TREC description queries and (2.) the document is currently active on the web as a web page. The 50 text segments, as a result, could be evaluated on TREC Gov2 using the benchmark relevance judgments since they are aligned with TREC topics. For retrieval purposes, the set of text segments is stopped with a standard INQUERY stopword list[3]. We also stop the documents from which the text segments are extracted. The statistics are shown in Table 5. Chunks are extracted from text segments using the parser from the Stanford NLP group[15] before stopwords are removed. There are an average of 24.8 chunks per text segment after extraction.

To perform the web search task, we use the publicly available search API, Bing API version 2, which enables us to programmatically submit queries and retrieve results from the Bing Engine. For each of the queries generated based on the corresponding text segments, the top 10 results returned are collected for a three-level manual relevance judgment {0, 1, 2}. The three-level relevance respectively stands for "not relevant", "partially relevant" and "relevant". These judgements are done by a single judge and are guided by how the returned results match statements of the narrative queries given in the benchmark.

[7]http://www.lemurproject.org/indri/

Table 5: Average number of words in text segments and documents before/after removing stopwords for the selected 50 topics.

	before	after
Text Segment	99.54	57.7
Document	5555.02	3193.72

Results of standard performance measurements are reported for various techniques under different environments. Evaluation is done on the top 1000 documents retrieved using Indri (TREC Retrieval) and the top 10 results returned from the Bing search engine (Web Retrieval). It should be noted that both search engines occasionally return the "source document" that the text segment comes from. These source documents are discarded in order to avoid giving biased evaluation results. Finally we adopt MAP as m(.) and use 10-fold cross validation for training and testing the CRF-perf models. Following Xue et al [29], we reduce the exponential set of combinations for training CRF_{CombC} by keeping combinations with 3 to 6 chunks. The threshold TH in Algorithm 1 is cross-validated in the collection and is empirically set to 0.42.

5.2 Retrieval Performance on TREC Gov2

Table 6 shows the retrieval results for queries generated based on the various proposed techniques. The baseline method "$TS_{original}$" retrieves documents using the raw texts in text segments, and we construct TS_n by removing stopwords from the raw texts. Additionally, we include the run "$AllChunks$" that uses all the chunks extracted from a text segment as a query. The rest of the approaches are based on the core approaches proposed in Sections 3 and 4.

We also compare the performance with several existing approaches. Pseudo relevance feedback (PRF)[28] has been shown to be effective for retrieval. A typical version of PRF assumes that the top k ranked documents are relevant and expands original query with top 20-30 terms from these documents using tf-idf weights. Following PRF, we extract the top 10 and 20 tf-idf weighted terms from TS_n as well as $Page_n$ (i.e., the document without stopwords) to form the queries. Since there is no "original benchmark queries" in our scenario, we search directly using the feedback terms such as TFIDF10(TS_n). To be comparable to our approaches, we also issue queries using a combination of TS_n and the set of expanded terms such as TS_n+TFIDF10(TS_n).

Moreover, we explore the feasibility of applying FindSimilar[26] proposed by Smucker and Allan to our problem. FindSimilar, a feedback-like search tool, improves retrieval performance by finding similar documents to users' relevance feedback documents and re-organizing retrieval results. A number of details are addressed in [26]; however, here we only focus on the document-to-document similarity as it most resembles our problem. Based on FindSimilar, we rank documents using Kullback-Leibler divergence of the query model M_Q and the document model M_D, where both M_Q and M_D are language models for a document. Specifically, we compute a M_Q for each document in the set $T = \{(qid, TS, Page)\}$, and compare the M_Q to all other

Table 6: Retrieval performance of the proposed and related approaches on TREC Gov2. Paired t-tests are performed between each technique and the basic methods (original and no-stopwords) . ^{o} or $_n$ is respectively marked if p-value < 0.05 compared to $TS_{original}$ or TS_n.

Approach	Query	MAP	Prec@5	Prec@10	nDCG@5	nDCG@10
Text segment	$AllChunks$	0.1488	0.4480	0.4500	0.2944	0.3077
	$TS_{original}$	0.1402	0.4520	0.4360	0.2941	0.3063
	TS_n	0.1558^o	0.4679	0.4379	0.3048	0.3074
	$TS_n + WeightC(20)$	0.1581^o	0.4640	0.4619	0.3129	0.3251
	$TS_n + CombC$	0.1592^o_n	0.4600	0.4540	0.3058	0.3169
	$TS_n + CombC + TopC_w(2)$	0.1675^o_n	0.4680	0.4560	0.2961	0.3091
	$TS_n + TopC(k)$	0.1677^o_n	0.4800	0.4939^o_n	0.3189	0.3401^o_n
PRF-like	TFIDF10(TS_n)	0.1339	0.4600	0.4460	0.3114	0.3211
	TFIDF20(TS_n)	0.1393	0.4760	0.4479	0.3058	0.3109
	TFIDF10($Page_n$)	0.0936	0.2920	0.2740	0.1811	0.1906
	TFIDF20($Page_n$)	0.0946	0.2880	0.3020	0.1800	0.2047
	TS_n+TFIDF10(TS_n)	0.1599	0.4598	0.4591	0.3101	0.3154
	TS_n+TFIDF20(TS_n)	0.1589	0.4440	0.4459	0.2980	0.3096
	TS_n+TFIDF10($Page_n$)	0.1586	0.4540	0.4559	0.2877	0.3176
	TS_n+TFIDF20($Page_n$)	0.1560	0.4280	0.4219	0.2638	0.2877
FindSimilar	$D_{KL}(M_Q\|\|M_D)$ *regular*	0.0100	0.1240	0.1020	0.0213	0.0878
	$D_{KL}(M_Q\|\|M_D)$ *biased*	0.0107	0.1360	0.1060	0.0318	0.0913

Table 7: Retrieval performance for 4 kinds of queries of topic number 812.

Query	MAP	p@5	n@5
$TS_n + WeightC(20)$	0.1024	0.6	0.4468
$TS_n + CombC$	0.1098	0.6	0.4468
$TS_n + CombC + TopC_w(2)$	0.1776	0.8	0.6608
$TS_n + TopC(k)$	0.2077	1.0	1.0000

documents (each is a M_D) in the Gov2 corpus. As mentioned in FindSimilar, there are two types of similarity: *regular* and *query-biased*. The main difference between the two is that *regular* constructs M_Q using all words in a particular document, whereas *query-biased* builds M_Q by collecting all words within a certain distance W to all query terms in the document. Again, we have no "standard query" in our problem domain, but still we try to imitate *query-biased* by regarding the TS as query.

Though we compare our approach with PRF and FindSimilar, it is important to clarify the intrinsic difference between our problem and these approaches. Our problem simulates the scenario where a user finds a piece of text in a document interesting and intends to find more information about it. The text segment itself plays the role of the query. Most previous work attempts to improve the performance of existing benchmark queries by incorporating more information from related documents.

We address several important points in Table 6: (1.) All the techniques proposed can significantly improve baseline performance in terms of MAP. From the run of "$AllChunks$", it is implied that using only chunks for search can leave out some important words resulting in limited improvement. (2.) It is noted that "$TS_n + WeightC(20)$" significantly improves the MAP performance of the "$TS_{original}$" run but not for TS_n. The result is not all that surprising as we can see the chunks are nearly equally weighted using the frequency-based approach (see Table 3 for example), which means that effective chunks are difficult to distinguish from others. (3.) The improvements based on "$TS_n + CombC$" are moderate. Though the MAP score can be significantly improved, the rest of measurements cannot be effectively enhanced. One possible reason can be that the features designed in the experiments did not take into account the global dependencies between chunks within a chunk combination. Modeling the underlying relationships could be potentially helpful for better estimation as suggested by [29]. (4.) The "$TS_n + CombC + TopC_w(2)$" is motivated by [5] that takes advantages of stressing the important single chunks in addition to the best chunk combination "$CombC$". The results show that "$TS_n + CombC + TopC_w(2)$" can further outperform the run of "$TS_n + CombC$". (5.) Finally, the "$TS_n + TopC(k)$" method consistently outperforms the heuristic methods such as "$TS_{original}$" and "TS_n" using different types of measurements. We believe the effectiveness of generating queries based on "$TS_n + TopC(k)$" comes from the automatic determination of k for each topic. This means that each topic is not limited to a fixed size of selected chunks, but includes as many chunks as needed according to their characteristics.

A concrete example showing the relative retrieval effectiveness can be found in Table 7, where the queries in Table 3 are directly used for retrieval. The result again shows that "$TS_n + TopC(k)$" outperforms all other techniques.

Moreover, compared to the related work in Table 6, all our approaches perform significantly better than the PRF-like(using only the feedback terms as query) and Findsimilar methods (p-value < 0.05). Moreover, the PRF-like(using combination with TS_n) is comparable to our approaches such as TS_n+$WeightC(20)$ and TS_n+$CombC$. We can also observe that using a text segment consistently outperforms using a whole document for expanded terms, showing that the use of text segments is more appropriate and precise. FindSimilar does not perform well, although our results for FindSimilar are consistent with [26] in that *query-biased* outperforms *regular* since *query-biased* captures more precise vocabulary distributions with its window W.

5.3 Parameter setting on TREC Gov2

In this section, we show the impact of different values of parameters used in the experiments. Figure 2 shows

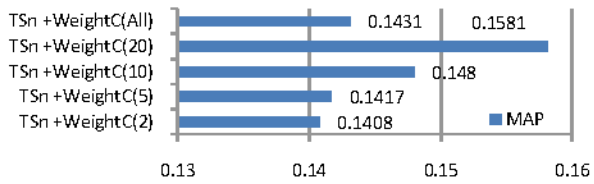

Figure 2: MAP of different number of chunks selected using frequency-based approach.

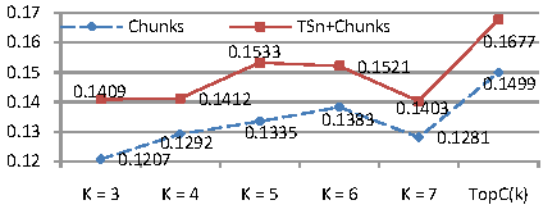

Figure 3: MAP of different number of chunks selected based on $TopC(k)$. Results of "Chunk-only" and "TSn+Chunks" are displayed.

the MAP of different numbers of chunks selected based on frequency-based approach. From Fig. 2, the MAP first increases as we add more chunks into the query, but too many additions can be noisy and thus the performance of using all chunks decreases. Using 20 chunks seems to be a reasonable choice for the frequency-based approach.

We also examine the impact of automatic determination of k in the learning-based approach $TopC(k)$. Figure 3 shows the MAP results of fixed $k = 3, 4, 5, 6, 7$ and also the k based on $TopC(k)$ (average value is 3.85). We can see that a fixed k value could not meet the requirements for all query topics, and thus using $TopC(k)$ is more effective than any fixed k. In addition to searching using chunks only, we provide the results of $TS_n + chunks$ for each k value. The MAP trend is similar, again validating that $TopC(k)$ based on Algorithm 1 can be effective.

5.4 Feature Analysis on TREC Gov2

Finding a set of key features that play important roles in retrieval is useful. This task can usually be done using regression diagnostics and subset selection. One common method is to remove a set of features and see the corresponding performance response. Assuming independence between the different features, a decrease in performance indicates how much the removed features contribute to the overall performance.

In Figure 4, we analyze the utility of the features described in Section 3.2.3 on the TREC Gov2 collection by removing a set of features at a time. A larger drop of MAP performance indicates better effectiveness of the features for predicting. In particular, we focus on the $TopC(k)$ approach for comparing the utility of features. From Fig. 4, we can see "Wiki title" and "MSN Query Log" are specifically helpful for selecting effective chunks. Google ngram corpus and MS Web ngram are less effective but still have a positive impact on retrieval performance. Removing typical features such as tf or df in the TREC gov2 collections causes a drop in performance. Other features such as tf in the text segment or the source page do not have obvious influence. Overall,

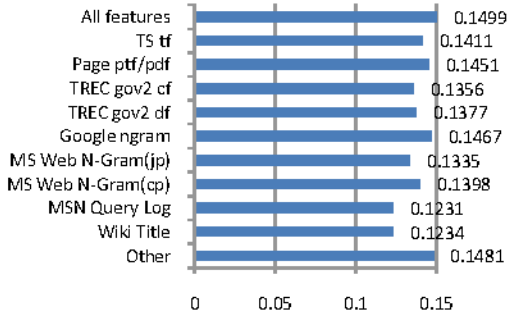

Figure 4: MAP of $TopC(k)$ when a category of features is removed. "All features" is the MAP using all features for predicting.

combining all features together gives the best performance.

5.5 Retrieval Performance on the Web

We performed retrieval experiments in the web environment, with aim of testing the effectiveness of the unweighted queries. For comparison, we also submit TS_n to the search API in addition to $TopC(2)$ and $TopC(k)$. The results of precision@k are shown in Table 8. We interpreted precision in 2 ways; "relaxed" judgments regard relevance levels $\{1, 2\}$ as relevant whereas "rigid" judgments count only level $\{2\}$ as relevant. We also report the results of nDCG@k in Table 9. One issue in calculating nDCG@k is that, if all top 10 pages are judged to be 0, the error of dividing by zero occurs due to all the zeros in the ideal ranked list. Thus we adopt two kinds of calculation addressing the problem. One is to regard nDCG@k as 0 for the all-zero topic. The other is to simply ignore that topic, resulting in 46, 45, 47 topics respectively for the three types of query.

A high level summary of Table 8 and Table 9 is that $TopC(k)$ is consistently better than TS_n and $TopC(2)$, and TS_n outperforms $TopC(2)$ in all cases. The effectiveness of $TopC(k)$ again validates that the automatic selection of k can bring success in both benchmark retrieval as well as web search.

Comparing the performance between TS_n and $TopC(2)$, we discover that TS_n tends to retrieve documents containing texts that are almost the same as the query text segment. Thus, despite removing the source document for evaluation, other similar pages keep being returned from the search engine as text reuse is a common phenomenon on the web. On the other hand, it can be challenging to select the 2 representative chunks (out of 24.8 in average) for $TopC(2)$; there is a good chance that the selected 2 chunks may not be relevant to original topic at all. Also, $TopC(2)$ sometimes could not cover all aspects of the information need, and thus the returned results are judged only "partially relevant".

$TopC(2)$ queries do have the advantages that the search results provide more information through higher diversity, compared to TS_n which often results in repeated text descriptions. In addition, some semantically correct $TopC(2)$ queries outperform both $TopC(k)$ and TS_n, showing that short queries can be more effective than long queries in the web environment. The standard deviations [8] of 6 performance measurements in Figure 5 show $TopC(2)$ results have

[8]Standard deviations are calculated using the "relaxed" judgments and count as 0 for precision@k and nDCG@k.

Table 8: Precision@k (p@k) of different unweighted queries.

		p@1	p@5	p@10
Relaxed	TS_n	0.7600	0.6600	0.5860
	$TopC(2)$	0.7200	0.6280	0.5660
	$TopC(k)$	0.8000	0.7439	0.6920
Rigid	TS_n	0.7000	0.5159	0.4479
	$TopC(2)$	0.5400	0.4599	0.4100
	$TopC(k)$	0.7200	0.6480	0.5640

Table 9: nDCG@k (n@k) of different unweighted queries. #: number of topics used for evaluation.

		#	n@1	n@5	n@10
Count as "0"	TS_n	50	0.7300	0.7409	0.8191
	$TopC(2)$	50	0.6500	0.6726	0.7710
	$TopC(k)$	50	0.7600	0.7745	0.8474
Don't count	TS_n	46	0.7934	0.8053	0.8903
	$TopC(2)$	45	0.7222	0.7474	0.8566
	$TopC(k)$	47	0.8085	0.8239	0.9015

larger deviations than TS_n and $TopC(k)$, validating that the results of $TopC(2)$ queries are more likely to produce more extreme results.

5.6 Summary

The most effective way of generating queries is based on $TopC(k)$, where the k top-performing chunks are automatically selected. Although "$TS_n + CombC + TopC_w(2)$" works well under the Indri retrieval system, the direct selection of $TopC(2)$ as a query does not give good results on the web search engine. A number of reasons for this were discussed previously. Interestingly, we find that simple techniques such as stopword removal actually perform better than expected. Specifically, we note that TS_n can significantly outperform the original texts, and the retrieval performance of TS_n even beats $TopC(2)$ in the web environment. This observation is consistent with the findings of Huston and Croft [12] that showed stopword removal can be very effective.

6. RELATED WORK

Previous work dealing with long or verbose queries are relevant to our paper. Specifically, selecting a subset of important words [2, 17, 18] from long queries has shown significantly improved results. Xue et al [29] modeled the distribution of sub-queries based on CRF-perf and construct effective queries according to the probabilities. In addition, Kumaran and Carvalho [19] learned to select sub-queries using various query quality predictors based on a Ranking SVM model. Another branch of research [7, 20, 23] improved performance for long queries by giving higher weights to important words. Despite the improvements reported, however, directly applying previous methods to this work could result in unrealistic computational demands since the length of text segments may be much longer than the queries such as description queries. We address the problem by dividing the original text segment into chunks.

Detection of important noun phrases [8] is also relevant to our work. Specifically, Bendersky and Croft [5] developed a technique for automatic extraction of key concepts. They learned a classification model by usage of different types of features and manual labeling on key concepts is required

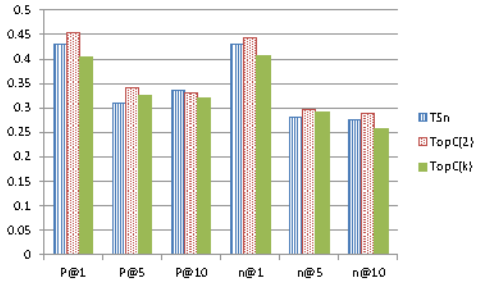

Figure 5: Standard deviation of 6 performance measurements for the 3 types of queries.

for the training set. Our method benefits from the direct optimization on retrieval performance and does not require human labeling. Moreover, the proposed algorithm in this paper can effectively decide the number of chunks k for each query, whereas the same number of key concepts was used for different topics in [5].

Another related research area involves the extraction of representative words from a document. Research on known-item search [4, 14] constructs (query, document) pairs by extracting important words from a document and formulating pseudo queries in a desktop search environment. This work, however, chooses words from a document based on some basic statistical indicators, which risks losing informative phrases and could be too simple to generate queries for long documents of 5000 words. Classic relevance feedback [24] and pseudo-relevance feedback [28, 13] have been used successfully to expand original queries and improve retrieval effectiveness. The major difference from our problem is that these techniques enhance the performance of an original query; however, we focus on automatically providing an effective query based on a selected text segment.

Other related techniques [25, 27, 30] view a document as a long query and retrieve other realted documents. Smucker and Allan [26] proposed a search tool FindSimilar by users' similarity browsing.

Lee et al [21] proposed a learning-based approach for query term ranking, and provided an application of automatic constructing queries from a web text segment. However, in [21], the decision of the number of terms used for a query was made manually, which is impractical in real applications. Also, they did not evaluate their set of text segments on publicly available benchmark collections, whereas our approach has shown to be effective in both TREC and web environments.

7. CONCLUSIONS

In this paper, we present approaches for generating queries based on user-selected text segments from a document. The task is important in that people reading a document or a web page may often see text passages that describe a topic of interest, and an automatically generated query based on that passage could potentially help people retrieve more relevant information. Also, the flexibility of letting users select any text segment could provide a better user experience, especially for applications on smart phones or tablets. One of the contributions in this paper is that we establish a means of constructing the set of query text segments, which makes it possible to evaluate performance across TREC and web

environments.

We propose several learning-based approaches to selecting effective chunks from the text segments. In the experiments, we show that our techniques can generate effective queries for both an open retrieval system (Indri search engine), and a black-box retrieval system (commercial search engine). In particular, we find that the technique $TopC(k)$ which has the advantage of automatic determination of k can significantly improve retrieval performance. Meanwhile, $TopC(k)$ is more efficient than other types of learning-based approaches since the training instances are composed of simple single chunks.

For future work, it would be helpful to take into account globally dependent features for chunk combinations, which may better capture the underlying relationships between the chunks inside the combination. In addition, we find that several segmentation problems occur during the process of chunk extraction. Examples include the mis-chunking of "type II diabetes" since "II" cannot be automatically identified as "2". We hope to alleviate the problem by providing a more robust way to do the segmentation.

Constructing applications on tablets or browsers could help collect feedback from real users. While we have demonstrated that the generated queries are effective across different collections, incorporating the generation algorithm into interactive user interfaces should be done in future work.

8. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by ARRA NSF IIS-9014442 . Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor. We appreciate the assistance of Xiaobing Xue and Michael Bendersky in providing codes and suggestions.

9. REFERENCES

[1] A. Adler, A. Gujar, B. L. Harrison, K. O'Hara, and A. Sellen. A diary study of work-related reading: design implications for digital reading devices. In *Proc. of SIGCHI*, CHI '98, pages 241–248, 1998.

[2] J. Allan, J. Callan, B. Croft, L. Ballesteros, J. Broglio, J. Xu, and H. Shu. Inquery at trec-5. In *Fifth Text REtrieval Conference TREC-5*, pages 119–132, 1997.

[3] J. Allan, M. E. Connell, W. B. Croft, F. Feng, D. Fisher, and X. Li. Inquery and trec-9. In *Proc. of the Ninth Text REtrieval Conference*, pages 551–562, 2000.

[4] L. Azzopardi, M. de Rijke, and K. Balog. Building simulated queries for known-item topics: an analysis using six european languages. In *Proc. of SIGIR*, SIGIR '07, pages 455–462, 2007.

[5] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *Proc. of SIGIR*, SIGIR '08, pages 491–498, 2008.

[6] M. Bendersky and W. B. Croft. Finding text reuse on the web. In *Proc. of WSDM*, WSDM '09, pages 262–271, 2009.

[7] M. Bendersky, D. Metzler, and W. B. Croft. Learning concept importance using a weighted dependence model. In *Proc. of WSDM*, WSDM '10, pages 31–40, 2010.

[8] J. P. Callan, W. B. Croft, and J. Broglio. Trec and tipster experiments with inquery. In *Information Processing & Management*, pages 31–3, 1994.

[9] Z. Cheng, B. Gao, and T.-Y. Liu. Actively predicting diverse search intent from user browsing behaviors. In *Proc. of WWW*, WWW '10, pages 221–230, 2010.

[10] G. Golovchinsky, M. N. Price, and B. N. Schilit. From reading to retrieval: freeform ink annotations as queries. In *Proc. of SIGIR*, SIGIR '99, pages 19–25, 1999.

[11] A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proc. of EMNLP*, EMNLP '03, pages 216–223, 2003.

[12] S. Huston and W. B. Croft. Evaluating verbose query processing techniques. In *Proc. of SIGIR*, SIGIR '10, pages 291–298, 2010.

[13] C. Jordan, C. Watters, and Q. Gao. Using controlled query generation to evaluate blind relevance feedback algorithms. In *Proc. of ACM/IEEE-CS JCDL*, JCDL '06, pages 286–295, 2006.

[14] J. Kim and W. B. Croft. Retrieval experiments using pseudo-desktop collections. In *Proc. of CIKM*, CIKM '09, pages 1297–1306, 2009.

[15] D. Klein and C. D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15, pages 3–10, 2003.

[16] R. Krovetz. Viewing morphology as an inference process. Technical report, Amherst, MA, USA, 1993.

[17] G. Kumaran and J. Allan. Adapting information retrieval systems to user queries. *Inf. Process. Manage.*, 44:1838–1862, November 2008.

[18] G. Kumaran and J. Allan. Effective and efficient user interaction for long queries. In *Proc. of SIGIR*, SIGIR '08, pages 11–18, 2008.

[19] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *Proc. of SIGIR*, SIGIR '09, pages 564–571, 2009.

[20] M. Lease. An improved markov random field model for supporting verbose queries. In *Proc. of SIGIR*, SIGIR '09, pages 476–483, 2009.

[21] C.-J. Lee, R.-C. Chen, S.-H. Kao, and P.-J. Cheng. A term dependency-based approach for query terms ranking. In *Proc. of CIKM*, CIKM '09, pages 1267–1276, 2009.

[22] C. C. Marshall. Annotation: from paper books to the digital library. In *Proc. of the second ACM international conference on Digital libraries*, DL '97, pages 131–140, 1997.

[23] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *Proc. of SIGIR*, SIGIR '05, pages 472–479, 2005.

[24] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *Knowl. Eng. Rev.*, 18:95–145, June 2003.

[25] R. Saraçoğlu, K. Tütüncü, and N. Allahverdi. A new approach on search for similar documents with multiple categories using fuzzy clustering. *Expert Syst. Appl.*, 34:2545–2554, May 2008.

[26] M. D. Smucker and J. Allan. Find-similar: similarity browsing as a search tool. In *Proc. of SIGIR*, SIGIR '06, pages 461–468, 2006.

[27] W. J. Wilbur and L. Coffee. The effectiveness of document neighboring in search enhancement. *Inf. Process. Manage.*, 30:253–266, March 1994.

[28] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *Proc. of SIGIR*, SIGIR '96, pages 4–11, 1996.

[29] X. Xue, S. Huston, and W. B. Croft. Improving verbose queries using subset distribution. In *Proc. of CIKM*, CIKM '10, pages 1059–1068, 2010.

[30] Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *Proc. of WSDM*, WSDM '09, pages 34–43, 2009.