Generating Reformulation Trees for Complex Queries

Xiaobing Xue W. Bruce Croft
Center for Intelligent Information Retrieval
Computer Science Department
University of Massachusetts, Amherst, MA, 01003, USA
{xuexb,croft}@cs.umass.edu

ABSTRACT

Search queries have evolved beyond keyword queries. Many complex queries such as verbose queries, natural language question queries and document-based queries are widely used in a variety of applications. Processing these complex queries usually requires a series of query operations, which results in multiple sequences of reformulated queries. However, previous query representations, either the "bag of words" method or the recently proposed "query distribution" method, cannot effectively model these query sequences, since they ignore the relationships between two queries. In this paper, a *reformulation tree* framework is proposed to organize multiple sequences of reformulated queries as a tree structure, where each path of the tree corresponds to a sequence of reformulated queries. Specifically, a two-level reformulation tree is implemented for verbose queries. This tree effectively combines two query operations, i.e., subset selection and query substitution, within the same framework. Furthermore, a weight estimation approach is proposed to assign weights to each node of the reformulation tree by considering the relationships with other nodes and directly optimizing retrieval performance. Experiments on TREC collections show that this reformulation tree based representation significantly outperforms the state-of-the-art techniques.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Performance

Keywords

Reformulation Tree, Verbose Query, Information Retrieval

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.
Copyright 2012 ACM 978-1-4503-1472-5/12/08 ...$15.00.

1. INTRODUCTION

Although short keyword queries are still very common in web search, the increasing diversity of search applications and information needs has led to increasing complexity in queries. For example, verbose (or long) queries have become more and more popular in web search. In community-based Q&A, users pose natural language questions as queries. In patent retrieval, the whole document (patent application) is considered as the query. These complex queries help users express their information need naturally and save the effort of picking keywords. However, processing these queries poses a significant challenge for search systems.

Dealing with complex queries usually requires a series of query operations. For example, a typical process of dealing with a verbose query can be described as follows. The system first selects a subset of query words from the original query to remove noisy information. Then, the generated subset query is further modified to handle vocabulary mismatch. Finally, weights are assigned to queries generated at each step. Depending on the application, the above process could become more complicated. For example, in cross-lingual retrieval, the original verbose query needs to be translated into a foreign language query before applying any further operation. The above process will generate multiple sequences of reformulated queries, where each sequence records a way of modifying the original query using several query operations. These reformulation sequences capture the relationships between the reformulated queries. Fig. 1 displays some examples of the reformulation sequences, where the subset query is selected from the original query at the first step of the sequence and the second step further substitutes the subset query.

However, previous query representations cannot model these reformulation sequences well. The "bag of words" representation is widely used in information retrieval. Using this representation, the original query is transformed into a set of weighted words. Some extensions to this representation introduce phrases [19] and latent words [14][20].

Reformulation Sequence
Q→reductions iraqs foreign debt→reduce iraqs foreign debt
Q→iraqs foreign debt→iraqs foreign debts
Q→iraqs foreign debt→iraqs external debt

Figure 1: The reformulation sequences generated for a verbose query Q "any efforts proposed or undertaken by world governments to seek reduction of iraqs foreign debt"

This representation does not explicitly model a reformulated query, which serves as the basis of the reformulation sequences. An example of the "bag of words" representation is shown in Fig. 2 (a). Recently, the "query distribution" representation [27] was proposed to transform the original query into a set of reformulated queries. For example, Xue et al [30] represents a verbose query as a set of subset queries. This representation indeed considers a reformulated query as the basic unit, but it fails to capture the relationships between the reformulated queries. Therefore, the sequences of reformulated queries still cannot be modeled using this representation. An example of the "query distribution" representation is shown in Fig. 2 (b).

In this paper, a novel query representation is proposed to transform a complex query into a reformulation tree, where the nodes at each level of this tree correspond to the reformulated queries generated using a specific query operation. Using this representation, a reformulation sequence is naturally modeled as a path from the root node to the leaf node. The construction of the reformulation tree simulates the process of applying a series of query operations to the complex query. Furthermore, weight is assigned to each node of the reformulation tree, which indicates the importance of the corresponding reformulated query. The estimation of the weight for a node considers not only the characteristics of this node itself, but also its relationships with other nodes. Different with previous reformulation models that treat retrieval models as independent steps, we estimate the weights on the reformulation tree by directly optimizing the performance of retrieval models, which considers the reformulation model and the retrieval model in a joint view.

Verbose queries, as a typical example of complex queries, have attracted much attention recently. Previous research on verbose queries either weights the query words in the original query [16, 15, 3] or selects the best subset of query words from the original query [12]. Relatively little research considers combining multiple query operations together for improving verbose queries. Therefore, as an implementation of the reformulation tree framework, a two-level tree structure is constructed for verbose queries, where the first level corresponds to the subset query selection operation and the second level corresponds to the query substitution operation. A weight estimation method is also described, which incorporates the relationships between different reformulated queries and directly optimizes the retrieval performance.

Fig. 2 (c) shows an example reformulation tree. The first level of this tree consists of two subset queries extracted from the original query, i.e., "reductions iraqs foreign debt" and "iraqs foreign debt". At the second level, each subset query is further modified to generate query substitutions. For example, "iraqs foreign debt" has been modified to "iraqs external debt". Furthermore, weight is assigned to each node of this tree, which measures the importance of each reformulated query. Compared with other representations, the reformulation sequences as shown in Fig. 1 are captured using the reformulation tree.

The contributions of this paper can be summarized as four folds. First, a tree based query representation is proposed to deal with complex queries, which models a series of query operations and captures the relationships between the reformulated queries. Second, a specific implementation, i.e., the two-level reformulation tree, is introduced for verbose queries, which combines two important operations, subset

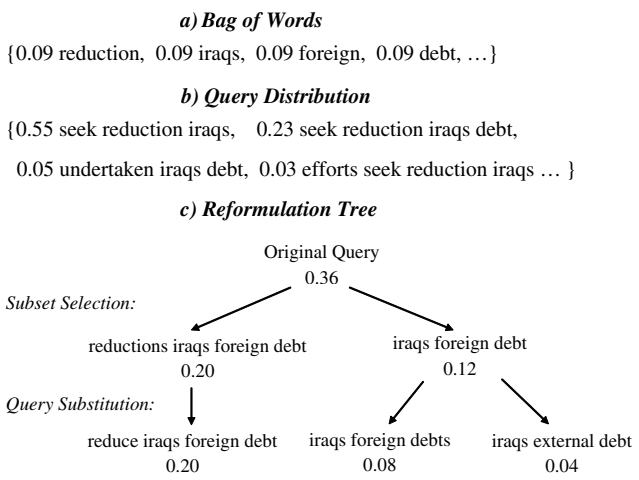


Figure 2: Different query representations for a verbose query "identify any efforts proposed or undertaken by world governments to seek reduction of iraqs foreign debt"

query selection and query substitution. Third, a weight estimation method is designed by incorporating the relationships between different reformulated queries and directly optimizing retrieval performance. Fourth, detailed experiments are conducted to show that the tree-based representation outperforms other query representations for verbose queries.

2. RELATED WORK

In this section, we first describe previous work on complex queries, especially on verbose queries and then we review previous query representation approaches.

2.1 Complex Query

As described in the introduction, complex queries have been widely used in different applications. Some examples include the verbose query, the natural language question query and the document-based query.

Kumaran and Allan [11] studied shortening a verbose query through human interaction. Bendersky and Croft [2] discovered key concepts from a verbose query. These key concepts were combined with the original query to improve the retrieval performance. Kumaran and Carvalho [12] learned to automatically select subset queries using several query quality predictors. Balasubramanian et al [1] extent [12] for web long queries.

Lease et al [16] developed a regression model to assign weights to each query word in the verbose query by using the secondary features. Lease [15] further combined their regression model with the Sequential Dependence Model, which achieved significant performance improvement. Bendersky et al [3] proposed a unified framework to measure the weights of words, phrases and proximity features underlying a verbose query.

A natural language question query is widely used in a community-based Question and Answer service such as Yahoo! Answers and Quora. Previous work [8, 9, 24] studied effectively finding previously answered questions that are relevant to a new question asked by a user. Different retrieval models have been proposed to calculate the simi-

larity between questions. For example, a translation based retrieval model [8] were developed to deal with the vocabulary mismatch between the semantically related questions.

A document-based query allows users to directly submit a document as a query. A typical example is in patent retrieval [13], where the whole patent application is submitted to the search system in order to find the relevant patents. Many features are extracted from a patent application and these features are the basis of retrieving relevant patents.

In this paper, a tree-based representation is proposed to improve the complex query. A specific implementation for verbose queries is described. This implementation combines subset selection and query modification within the same framework, which has not been explored in previous work.

2.2 Query Representation

In this section, we review two types of query representations, "bag of words" and "query distribution".

The "bag of words" representation transforms the original query into a set of terms, either weighted or not. These terms include the words and phrases from the original query and the latent words and phrases extracted from the corpus. For example, the relevance model approach [14] adds latent words to the original query, the sequential dependency model [19] detects the phrase structure, and the latent concept expansion model [20] uses proximity features and latent words. This type of representation does not consider how to use words and phrases to form actual reformulated queries. In other words, a reformulated query is not explicitly modeled in this representation.

The "query distribution" representation transforms the original query into a set of reformulated queries, where each query is assigned a probability. This probability helps measure the importance of the query. For example, Xue et al [30] represented a verbose query as a distribution of subset queries and a modified Conditional Random Field is proposed to estimate the probability for each subset query. This type of representation indeed considers the reformulated query as the basic unit, but it assumes independence between the reformulated queries. When multiple query operations are applied, this independence assumption usually does not hold.

In this paper, the proposed "reformulation tree" representation extends the "query distribution" representation by modeling the relationships between reformulated queries using the tree structure.

Some previous work also considers the relationships between queries. Boldi et al [4] proposed to build a query-flow graph that modeled web users' search behaviors. Specifically, the directed edges between two queries indicated that they were likely to belong to the same search mission. Mei et al [17] presented a general framework to model search sequences, where a search sequence is represented as a nested sequence of search objects. The above work focuses on short keyword queries and uses query logs to capture the relationships between the queries submitted within the same search session. In contrast, in this paper, we study complex queries and model the relationships between the reformulated queries. Furthermore, the construction of the reformulation tree proposed in this paper is closely related to the final retrieval performance, while previous work studies the query graph or sequence independently from the retrieval model.

Guo et al [6] proposed a CRF-based model for query refinement, which combines several tasks like spelling correction, stemming and phrase detection. This model focuses on morphological changes of keyword queries such as spelling correction and stemming, but does not consider complex queries.

3. BACKGROUND

In this section, we summarize several basic concepts used in this paper.

A *complex query* (q) is more complicated than a short keyword query. Examples of complex queries include verbose queries, natural language question queries and document-based queries. In this paper, we will focus on verbose queries.

A *query operation* (r) indicates a query processing technique. In this paper, we focus on two query operations, i.e. subset query selection and query substitution. Subset query selection [11, 12, 1, 30] selects a subset of query words from the original query. Query substitution [10, 26, 29] replaces the original query word with a new word. Other examples of query operations include query translation, query segmentation and so on.

A *reformulated query* (q_r) is the output of applying a query operation. A *reformulation sequence* is a sequence of reformulated queries by applying a series of query operations.

A *reformulation tree* (T) is a tree structure that organizes the reformulated queries generated by different query operations. Each path of T corresponds to a reformulation sequence.

4. REFORMULATION TREE

In this section, we first describe the framework for generating the reformulation tree T. Then, we compare this tree-based representation with previous query representations. Finally, the principle of the weight estimation is described.

4.1 Framework

Suppose that n query operations $\{r_1, r_2, ..., r_n\}$ are required to process a complex query q. Then, q is transformed into a n-level tree T. Each node of T represents a reformulated query q_r. From now on, if not explicitly indicated, we use q_r to represent both a node of T and the corresponding reformulated query. The root node of T represents the original query q, which can be considered as a special reformulated query. The ith level of T are generated by applying the ith operation r_i to the nodes at the $(i-1)$th level. An arc is added between the nodes at the $(i-1)$th level and the nodes at the ith level if the latter is the output of applying r_i to the former. Therefore, each path of T corresponds to a reformulation sequence. Furthermore, weight $w(q_r)$ is assigned to each node of T, which measures the importance of the corresponding reformulated query q_r.

When T is used for retrieval, the retrieval score of a document D is calculated using Eq. 1.

$$sc(T, D) = \sum_{q_r \in T} w(q_r) sc(q_r, D) \quad (1)$$

where $w(q_r)$ is the weight assigned to the node corresponding to the reformulated query q_r and $sc(q_r, D)$ is the retrieval score of using q_r to retrieve D. In general, $sc(T, D)$ is calculated by combining the retrieval score of using each

reformulated query q_r in T, where $w(q_r)$ is used as the combination weight. The calculation of $sc(q_r, D)$ depends on implementation.

4.2 Comparisons of Query Representations

In this subsection, we compare different query representations using the example in the introduction. Fig. 2 displays the "bag of words" representation, the "query distribution" representation and the "reformulation tree" representation.

In the "bag of words" representation, the basic unit is words or phrases. This representation may tell you that the important words in the original query are "reduction", "iraqs" and "debt", but how these words can be used together to form a new query is not clear.

The "query distribution" representation extends the "bag of words" representation by explicitly modeling a reformulated query. For example, this representation lists the top ranked subset queries such as "seek reduction iraqs" and "seek reduction iraqs debt". However, the relationships between the reformulated queries are not captured using this representation.

When a series of query operations are applied, we need a representation that models the reformulation sequences generated using these operations. The "reformulation tree" representation is designed to solve this problem. For example, in Fig. 2, the subset queries and the query substitutions are organized into a tree structure. This structure indicates that we need to first select subset queries and then conduct query substitution. It captures the relationships between "iraqs foreign debt" and "iraqs external debt", since the latter is the output of applying the query subsitution operation to the former.

4.3 Weight Estimation

The weight assigned to each node in the tree indicates the importance of the corresponding reformulated query. This weight should characterize both the features of this node itself and its relationships with other nodes. Therefore, the weight of q_r, i.e., $w(q_r)$, is calculated in Eq. 2.

$$w(q_r) = w(par(q_r)) \sum_k \lambda_k f_k(q_r) \quad (2)$$

where $par(q_r)$ denotes the parent node of q_r. f_k is the query feature extracted from q_r and λ_k is the parameter. Eq. 2 shows that the weight of q_r is not only decided by its own query features $\{f_k\}$ but also by the weight of its parent node $par(q_r)$. Intuitively, if q_r is important, its children should also receive high weights. In this way, the relationships between reformulated queries are incorporated into the weight estimation.

Note that Eq. 2 provides the principle of weight estimation. How to calculate $w(q_r)$ will depend on the implementation.

5. REFORMULATION TREE FOR VERBOSE QUERIES

In this section, we describe a two-level reformulation tree for verbose queries. We first describe the query operations used to construct the reformulation tree, i.e. subset query selection and query substitution. Then, we introduce a stage-based weight estimation method to assign weight to each node. Finally, the retrieval model is described.

5.1 Building Tree Structure

The construction of the reformulation tree for verbose queries consists of two steps: first, subset queries are selected from the original query; second, the subset queries generated in the previous step are further modified to generate query substitutions.

We follow Kumaran and Carvalho [12]'s method to generate subset queries. All query words from the original verbose query are considered. If the length of the verbose query is bigger than ten, we first rank all query words by their *idf* values and then pick the top ten words for the subset query generation. Then, all subset queries with length between three and six words are generated.

The passage analysis technique [29] is used to generate query substitutions. In order to replace one word from the original query, all the passages containing the rest of the query words are first extracted. Then, three methods are used to generate candidates for query substitution from these passages. **Morph** considers the morphologically similar words as candidates. **Pattern** considers the words matching the patterns extracted from the original query as candidates. **Wiki** considers the Wikipedia redirect pairs as the candidates. More details can be found in [29]. Finally, the top ranked candidates are used as query substitutions.

Given the above two query operations, the reformulation tree for the verbose query can be generated in this way. First, all subset queries with length between three to six are extracted from the original query. Each subset query is assigned a weight. How to estimate the weight will be described in the next subsection. According to this weight, we will pick the top ranked subset queries to construct the first level of the reformulation tree. *SubNum* is a parameter that controls the number of nodes at the first level. Second, among these *SubNum* subset queries, we further modify the top *ModNum* queries to generate query substitutions, which constructs the second level of the reformulation tree. *ModNum* is another parameter that controls the number of nodes that will be substituted.

For example, the reformulation tree used in the introduction (Fig. 2) can be constructed in two steps. This process is illustrated in Fig. 3. First, we pick the top two subset queries "reductions iraqs foreign debt" and "iraqs foreign debt" to construct the first level of the reformulation tree. Second, we modify these two subset queries respectively. For the first subset query, "reduce iraqs foreign debt" is generated by replacing "reduction" with "reduce". For the second subset query, two query substitutions, i.e. "iraqs foreign debt" and "iraqs external debt" are generated.

5.2 Weight Estimation

Eq. 2 indicates that the weight of a node in the reformulation tree depends on both its intrinsic features and the weight of its parent node. However, how to estimate the weight is still unclear. In this part, we describe a stage-based weight estimation method. The learning-to-rank based parameter estimation strategy [28] is used as the basis, which transforms a query feature into a corresponding retrieval feature.

In the initial stage, the root node (the original query q) is assigned the weight 1, i.e. $w(q) = 1$.

In **Stage I**, after the subset queries q_{sub} are generated, we calculate the weight of q_{sub} using Eq. 3.

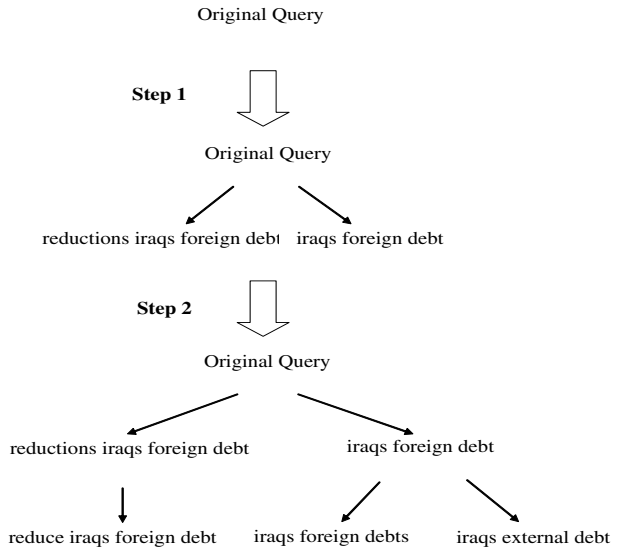


Figure 3: The process of constructing a reformulation tree

$$w(q_{sub}) = w(q)\sum_k \lambda_k^{sub} f_k^{sub}(q_{sub}) = \sum_k \lambda_k^{sub} f_k^{sub}(q_{sub}) \quad (3)$$

Eq. 3 instantiates Eq. 2 by focusing on the subset queries. f_k^{sub} is the query feature extracted from q_{sub} and λ_k^{sub} is the corresponding parameter. Since the root node is the parent of every subset query, its weight $w(q) = 1$, is used in Eq. 3.

In order to estimate $\{\lambda_k^{sub}\}$ by directly optimizing the retrieval performance, we transform each query feature f_k^{sub} into the corresponding retrieval feature F_k^{sub}, where F_k^{sub} is calculated in Eq. 4.

$$F_k^{sub}(\{q_{sub}\}, D) = \sum_{q_{sub}} f_k^{sub}(q_{sub}) sc(q_{sub}, D) \quad (4)$$

where $sc(q_{sub}, D)$ is the retrieval score of using q_{sub} to retrieve D. The calculation of $sc(q_{sub}, D)$ depends on the retrieval model. The retrieval feature F_k^{sub} combines the retrieval score of each subset query q_{sub} using their corresponding query feature $f_k^{sub}(q_{sub})$ as the combination weight. In general, F_k^{sub} indicates how well documents are ranked if f_k^{sub} is used as the weight to combine subset queries.

Now, we obtain a set of retrieval features $\{F_k^{sub}\}$. The problem of estimating $\{\lambda_k^{sub}\}$ to combine the query features $\{f_k^{sub}\}$ is transformed into the problem of combining the corresponding retrieval features $\{F_k^{sub}\}$ to achieve the best retrieval performance. The latter problem is typically solved using learning to rank techniques. In this paper, the ListNet method [5] is adopted to learn $\{\lambda_k^{sub}\}$ on the training set.

After obtaining $\{\lambda_k^{sub}\}$, we can assign the weight for each subset query according to Eq. 3.

In **Stage II**, we assign weights to the substituted queries. The weight of a substituted query q_{mod} is calculated using Eq. 5.

$$w(q_{mod}) = w(q_{sub})\sum_k \lambda_k^{mod} f_k^{mod}(q_{mod}) \quad (5)$$

Table 1: Summary of features

Features for Subset Queries $f_k^{sub}(q_{sub})$	
MI[11]	mutual information
SQLen[12]	sub-query length
QS[7]	query scope
QC[25]	query clarity score
SOQ[12]	similarity to original query
psg	count of passages containing q_{sub}
KeyCpt[2]	whether contains the key concept
Features for Substituted Queries $f_k^{mod}(q_{mod})$	
Morph	generated using Morph
Pattern	generated using Pattern
Wiki	generated using Wiki
psg	count of passages containing q_{mod}
seg-type	the number of possible segmentations

where q_{sub} is the parent node of q_{mod}. Compared with Eq. 3, the weights of the subset queries $w(q_{sub})$ generated in Stage I are incorporated in Eq. 5.

Similarly, in order to estimate $\{\lambda_k^{mod}\}$, we transform f_k^{mod} into the corresponding retrieval feature F_k^{mod} using Eq. 6.

$$F_k^{mod}(\{q_{mod}\}, D) = \sum_{q_{mod}} w(q_{sub}) f_k^{mod}(q_{mod}) sc(q_{mod}, D) \quad (6)$$

where q_{sub} is the parent node of q_{mod}. In general, F_k^{mod} tells how well the documents are ranked if f_k^{mod} is used as the weight to combine the substituted queries. Thus, the parameters $\{\lambda_k^{mod}\}$ are learned by combining these retrieval features $\{F_k^{mod}\}$ using ListNet.

5.3 Features

In this part, we describe the query features used to characterize the subset queries and the substituted queries.

The features used to characterize the subset queries are mainly query quality predictors. This type of features have been widely used in previous research [12][30]. Examples of query quality predictors include Mutual Information [11], Query Scope [7] and Query Clarity [25]. In addition, passage information is considered. The number of passages that contain a subset query provides strong evidence for the quality of a subset query. Whether a subset query contains key concepts is also considered as a feature. These key concepts were discovered by Bendersky et al [2].

The features used to characterize the substituted queries include the type of methods of generating substituted queries. As described in Section 5.1, "Morph" indicates using the morphologically similar words as candidates, "Pattern" indicates using the pattern based method and "Wiki" indicates using the Wikipedia redirect page. The passage information is also considered as one feature. Furthermore, the number of possible segmentations of a substituted query is used as another feature. This feature can be directly obtained using the passage analysis technique [29].

The details of features are summarized in Table 1.

5.4 Retrieval Model

The retrieval score of using a reformulation tree T can be calculated using Eq. 1. For example, given the reformulation tree shown in Fig. 2, the retrieval score can be calculated as follows:

$$
\begin{aligned}
sc(T, D) &= 0.36 \times sc(\text{Original Query}, D) \\
&= +0.2 \times sc(\text{“reductions iraqs foreign debt”}, D) \\
&= +0.12 \times sc(\text{“iraqs foreign debt”}, D) \\
&= +0.2 \times sc(\text{“reduce iraqs foreign debt”}, D) \\
&= +0.08 \times sc(\text{“iraqs foreign debts”}, D) \\
&= +0.04 \times sc(\text{“iraqs external debt”}, D)
\end{aligned}
$$

In this paper, the sequential dependency model (SDM) [19] is used to calculate $sc(q_r, D)$, which has been widely used in previous work [2, 30] as a state-of-the-art technique. Using SDM, the score of a document can be calculated as follows:

$$
\begin{aligned}
sc(q_r, D) &= \lambda_T \sum_{t \in T(q_r)} \log(P(t|D)) + \lambda_O \sum_{o \in O(q_r)} \log(P(o|D)) \\
&+ \lambda_U \sum_{u \in U(q_r)} \log(P(u|D)) \qquad (7)
\end{aligned}
$$

where $T(q_r)$ denotes a set of query words of q_r, $O(q_r)$ denotes a set of ordered bigrams extracted from q_r and $U(q_r)$ denotes a set of unordered bigrams extracted from q_r. λ_T, λ_O and λ_U are parameters controlling the weights of different parts and are usually set as 0.85, 0.1 and 0.05 [19]. $P(t|D)$, $P(o|D)$ and $P(u|D)$ are calculated using the language modeling approach [22, 31].

The SDM model can be easily implemented using the Indri query language [18]. Fig. 4 shows an example of Indri query for SDM model.

Figure 4: Example of Indri query.

```
q_r : iraqs foreign debt
SDM: #weight(
       0.85 #combine(iraqs foreign debt)
       0.10 #combine(#1(iraqs foreign) #1(foreign debt))
       0.05 #combine(#uw8(iraqs foreign) #uw8(foreign debt))
       )
```

6. EXPERIMENTS

Four TREC collections, Gov2, Robust04, ClueWeb (Category B) and Wt10g are used for experiments. Robust04 is a newswire collection, while the rest are web collections. The statistics of each collection are reported in Table 2. For each collection, two indexes are built, one not stemmed and the other stemmed using the Porter Stemmer[23]. Stemming transforms a word into its root form, which is conducted either during indexing or during query processing. The latter case treats stemming as a part of query reformulation, which has been shown effective for web search [21]. Both cases are considered in this paper using two types of indexes. No stopword removal is done during indexing. For each topic, the description part is used as the query. A short list of 35 stopwords and some frequent stop patterns (e.g., “find information”) are removed from the description query.

The query set of each collection is split into a training set and a test set. On the training set, the parameters λ_k are learned. On the test set, the learned parameters λ_k are used to assign weight to the reformulation tree generated from each test query. Specifically, ten-fold cross validation is used, where the query set is split into ten folds. Each time nine folds are used for training and one fold is used for test. This process repeats ten times.

Several baselines are compared. QL denotes the query likelihood language model [22, 31]. SDM denotes the sequential dependence model [19]. KC denotes the key concept method proposed by Bendersky et al [2]. Note that we do not report KC on ClueWeb, since the key concept query is not provided on ClueWeb in [2]. QL+SubQL and DM+SubQL [30] are the subset query distribution methods,

Table 2: TREC collections used in experiments

Name	Docs	Topics
Gov2	25,205,179	701-850
Robust04	528,155	301-450,601-700
Wt10g	1,692,096	451-550
ClueWeb	50,220,423	1-100

which combine the original query with a distribution of subset queries. QL+SubQL uses QL for both the original query and the subset queries, while DM+SubQL uses SDM for the original query and uses QL for the subset queries. In this paper, QL+SubQL and DM+SubQL are trained using the global features mentioned in [30]. SDM, KC, QL+SubQL and DM+SubQL are the state-of-the-art techniques for verbose queries. SDM and KC can be classified as the “bag of words” representation, while QL+SubQL and DM+SubQL can be considered as the “query distribution” representation. Therefore, the comparisons with these baselines help show the advantages of the “reformulation tree” representation.

The proposed reformulation tree approach is denoted as RTree, which uses SDM as the underlying retrieval model. Two parameters are used during the tree construction, *SubNum* and *ModNum*, where *SubNum* denotes how many subset queries are kept in the reformulation tree and *ModNum* denotes among those kept subset queries how many are further modified to generate query substitutions. In this paper, *SubNum* takes all subset queries generated and *ModNum* is set as 10. The effect of these parameters will be explored in the following part of this paper.

The standard performance measures, mean average precision (MAP), precision at 10 (P10) and the normalized discounted cumulative gain at 10 (NDCG10), are used to measure retrieval performance. In order to improve readability, we report 100 times the actual values of these measures. The two-tailed t-test is conducted for significance.

The *Lemur 4.10* toolkit is used to build the index and run the query. The *ireval* package provided in the toolkit is used for evaluation and significance test.

6.1 Example

In Table 3, we show some examples of the generated reformulation trees. As mentioned previously, some stopwords and stop patterns are removed from the original query. Those words are kept to improve readability. Note that they are not used for retrieval and subset query generation.

Table 3 shows that the subset queries and the substituted queries are effectively combined within the same framework. For example, given the original query “what allegations have been made about enrons culpability in the california energy crisis”, the reformulation tree first generates high quality subset queries “enrons culpability california energy crisis” and “california energy crisis” and then further modifies “california energy crisis” as two substituted queries “california gas prices” and “california electricity crisis”.

Table 4: Comparisons of retrieval performance. * denotes significantly different from the baseline.

	Gov2			Robust04			Wt10g			ClueWeb		
	MAP	P10	NDCG10	MAP	P10	NDCG10	MAP	P10	NDCG10	MAP	P10	NDCG10
Non-stemmed Index												
QL	22.46	49.13	35.94	22.40	39.12	39.63	16.44	28.97	27.86	10.97	21.63	14.10
SDM	23.98	51.01	38.81	23.30	40.04	40.60	16.76	31.65	29.82	11.53	22.76	15.04
KC	24.88	50.87	37.72	23.87	40.76	40.75	17.45	30.82	29.55	n/a	n/a	n/a
QL+SubQL	23.36	50.81	37.96	22.85	39.28	39.98	16.81	29.79	28.48	11.01	21.84	14.16
DM+SubQL	24.82	53.36	40.55	23.65	40.32	41.15	18.25	31.13	30.71	11.54	21.94	14.85
RTree	**26.70**	**53.96**	**40.78**	**25.07**	**42.33**	**42.64**	**19.44**	**34.02**	**32.80**	**12.94**	**26.43**	**18.05**
vs. QL	18.9%*	9.8%*	13.5%*	11.9%*	8.2%*	7.6%*	18.2%*	17.4%*	17.7%*	18.0%*	22.2%*	28.0%*
vs. SDM	11.3%*	5.8%*	5.1%*	7.6%*	5.7%*	5.0%*	16.0%*	7.5%*	10.0%*	12.2%*	16.1%*	20.0%*
vs. KC	7.3%*	6.1%*	8.1%*	5.0%*	3.9%*	4.6%*	11.4%*	10.4%*	11.0%*	n/a	n/a	n/a
vs. QL+SubQL	14.3%*	6.2%*	7.4%*	9.7%*	7.8%*	6.7%*	15.6%*	14.2%*	15.2%*	17.5%*	21.0%*	27.5%*
vs. DM+SubQL	7.6%*	1.1%	0.6%	6.0%*	5.0%*	3.6%*	6.5%	9.3%*	6.8%	12.1%*	20.5%*	21.5%*
Porter-stemmed Index												
QL	25.43	52.21	38.42	25.49	43.13	42.89	19.61	32.68	31.31	12.64	23.57	15.46
SDM	27.85	54.03	40.14	26.83	44.94	44.78	20.87	35.77	33.21	13.01	24.90	15.87
KC	27.52	53.83	39.10	25.97	41.65	42.29	21.01	34.02	32.29	n/a	n/a	n/a
QL+SubQL	26.19	53.36	39.51	25.81	43.01	43.23	20.11	32.06	30.93	12.94	25.00	16.37
DM+SubQL	28.49	55.91	**41.95**	26.99	44.86	44.94	21.98	35.05	33.45	13.29	24.08	15.58
RTree	**29.85**	**56.38**	41.84	**28.00**	**45.10**	**45.30**	**23.80**	**37.42**	**35.84**	**13.69**	**25.82**	**18.09**
vs. QL	17.4%*	8.0%*	8.9%*	9.8%*	4.6%*	5.6%*	21.4%*	14.5%*	14.5%*	8.3%	9.5%	17.0%
vs. SDM	7.2%*	4.3%*	4.2%*	4.4%*	0.4%	1.2%	14.0%*	4.6%	7.9%*	5.2%*	3.7%	14.0%*
vs. KC	8.5%*	4.7%	7.0%*	7.8%*	8.3%*	7.1%*	13.3%*	10.0%*	11.0%*	n/a	n/a	n/a
vs. QL+SubQL	14.0%*	5.7%*	5.9%*	8.5%*	4.9%*	4.8%*	18.3%*	16.7%*	15.9%*	5.8%	3.3%	10.5%
vs. DM+SubQL	4.8%*	0.8%	-0.3%	3.7%*	0.5%	0.8%	8.3%	6.8%*	7.1%*	3.0%	7.2%	16.1%*

Table 3: Examples of the reformulation tree. The top ranked nodes are displayed. In the original query Q, the stopwords and stop structures are italicized.

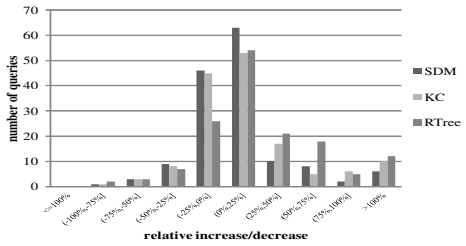
Q: *what* allegations *have been* made about enrons culpability *in the* california energy crisis
0.194 Q 0.133 enrons culpability california energy crisis 0.047 california energy crisis 0.009 california gas prices 0.008 california electricity crisis
Q: *give information on* steps *to* manage control *or* protect squirrels
0.148 Q 0.060 control protect squirrels 0.048 control squirrels 0.013 control of ground squirrels 0.012 control squirrel
Q: *what is the* state *of* maryland *doing to* clean up *the* chesapeake bay
0.089 Q 0.063 maryland chesapeake bay 0.015 md chesapeake bay 0.009 maryland chesapeake bay watershed 0.034 chesapeake bay 0.007 chesapeake bay watershed 0.006 chesapeake bay river

6.2 Retrieval Performance

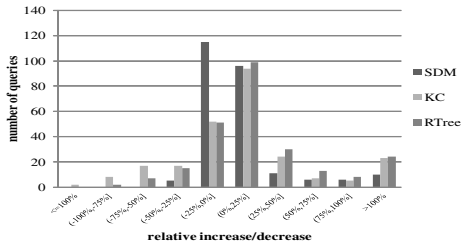
The first experiment is conducted to compare the retrieval performance of the proposed RTree method with the baselines. The baseline methods include QL, SDM, KC, QL+Sub QL and DM+SubQL. The results are shown in Table 4. The best performance is bolded.

Table 4 shows that RTree outperforms all the baseline methods. Specifically, RTree performs better than the "bag of word" representations, SDM and KC. Using the non-stemmed index, RTree significantly improves SDM and KC on all four collections with respect to all three performance measures. For example, on ClueWeb, RTree improves SDM by 12.2% and 20.0% with respect to MAP and NDCG10, respectively. On Wt10g, RTree improves KC by 11.4% and 11.0% with respect to MAP and NDCG10, respectively. On the Porter-stemmed index, similar results are also observed. These results show that the "reformulation tree" representation is more effective than the "bag of words" representation on modeling verbose queries, since the former explicitly models the reformulated query, while the latter only considers words and phrases.

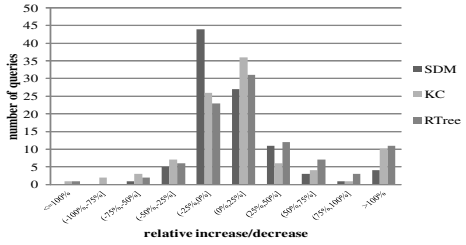
Furthermore, RTree also outperforms the "query distribution" representations, QL+SubQL and DM+SubQL. Using the non-stemmed index, RTree outperforms QL+SubQL and DM+SubQL on all four collections with respect to all three measures. Most of the improvements are significant. For example, on ClueWeb, RTree improves QL+SubQL by 17.5% and 27.5% with respect to MAP and NDCG10, respectively. Also, RTree improves DM+SubQL by 12.1% and 21.5% with respect to MAP and NDCG10, respectively. The results using the Porter-stemmed index are similar. These observations indicate that the "reformulation tree" representation is better than the "query distribution" representation,



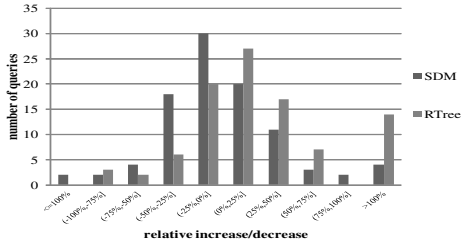
(a) Gov2



(b) Robust04



(c) Wt10g



(d) ClueWeb

Figure 5: Analysis of relative increases/decreases of MAP over QL.

since the former effectively combines different reformulation operations within the same framework.

It is not surprising that RTree brings more improvements over the baselines using the non-stemmed index than using the Porter-stemmed index, since some effect of query substitutions, especially those generated using the morphologically similar words, is already provided by the Porter stemmer.

6.3 Further Analysis

Table 4 shows that RTree significantly outperforms the baseline methods. In this part, we make detailed comparisons between RTree and the baseline approaches.

First, we compare RTree with SDM and KC. Specifically, we analyze the number of queries each approach increases or decreases over QL. Fig 5 shows the histograms of SDM, KC and RTree based on the relative increases/decreases of MAP over QL. The non-stemmed index is used in Fig. 5. Similar results are observed using the Porter-stemmed index.

Table 5: Comparisons with QL+SubQL and DM+SubQL. "+", "=" and "-" denote that RTree performs better, equal or worse than QL+SubQL and DM+SubQL with respect to MAP.

RTree	vs. QL+SubQL			vs. DM+SubQL		
	+	=	-	+	=	-
Gov2	71.81%	0.67%	27.52%	63.09%	0.67%	36.24%
Robust04	68.27%	0.00%	31.73%	63.05%	0.00%	36.95%
Wt10g	62.89%	2.06%	35.05%	62.89%	1.03%	36.08%
ClueWeb	65.31%	2.04%	32.65%	70.41%	2.04%	27.55%

Table 6: The effect of subset query selection and query substitution with respect to MAP

MAP	Gov2	Robust04	Wt10g	ClueWeb
SDM	23.98	23.30	16.76	11.53
KC	24.88	23.87	17.45	n/a
QL+SubQL	23.36	22.85	16.81	11.01
DM+SubQL	24.82	23.65	18.25	11.54
RTree-Subset	25.80	24.76	18.11	11.73
RTree	26.70	25.07	19.44	12.94

Fig. 5 shows that RTree improves more queries than SDM and KC. For example, on Gov2, RTree improves 110 queries out of the total 150 queries, while SDM and KC improve 89 and 91, respectively. On Robust04, RTree improves 174 queries out of the total 250 queries, while SDM and KC improve 129 and 153 queries, respectively. At the same time, RTree also hurts less queries than SDM and KC. These observations indicate that RTree is more robust than both SDM and KC.

Furthermore, we compare RTree with QL+SubQL and DM+SubQL. QL+SubQL and DM+SubQL only consider subset query selection, while RTree combines both subset query selection and query substitution. The comparisons between them indicate whether RTree effectively combines two query operations to improve verbose queries. Specifically, we analyze the percent of queries where RTree performs better than QL+SubQL and DM+SubQL, respectively. The results using the non-stemmed index are reported in Table 5.

Table 5 shows that RTree consistently outperforms QL+SubQL and DM+SubQL for 60%-70% queries on all four collections. These results indicate that RTree provides an effective way to combine different query operations, which significantly improves the retrieval performance of verbose queries.

6.4 Subset Selection vs. Query Substitution

RTree combines subset query selection and query substitution together using a two-level reformulation tree. Previous experiments have demonstrated the general effect of this approach. In this part, we split the effect of subset query selection and query substitution. Specifically, we propose a one-level reformulation tree, which only consists of subset queries. This one-level reformulation tree is denoted as RTree-Subset. The comparisons between RTree-Subset and other approaches using the non-stemmed index are shown in Table 6.

In Table 6, RTree-Subset outperforms the baseline methods, which indicates the effect of subset queries in the reformulation tree. When query substitutions are introduced, RTree further improves RTree-Subset. Thus, both subset selection and query substitution account for the performance

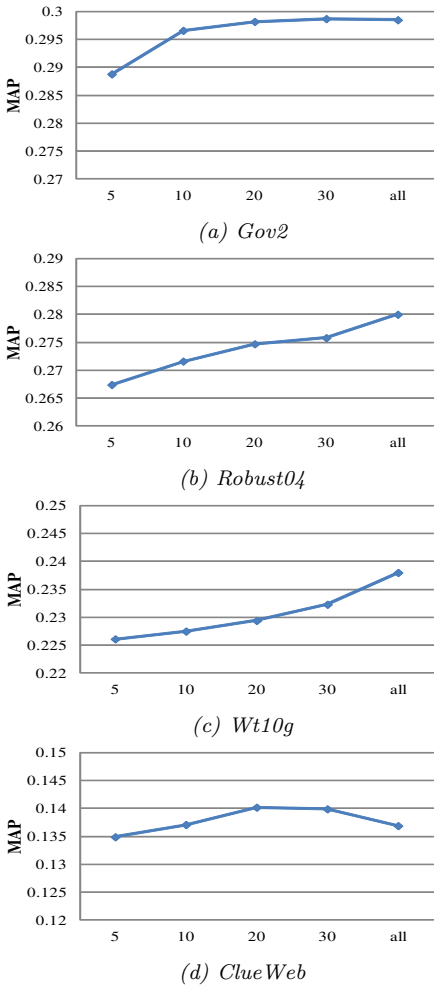


Figure 6: The effect of the parameter *SubNum*. x-axis denotes *SubNum* and y-axis denotes MAP.

of RTree. RTree-Subset also performs better than other subset query selection methods such as QL+SubQL and DM+SubQL.

6.5 Parameter Analysis

As described in Section 5.1, there are two parameters used during the process of constructing the reformulation tree, $SubNum$ and $ModNum$. $SubNum$ denotes the number of subset queries used in the reformulation tree and $ModNum$ denotes the number of subset queries that are modified to generate query substitutions. In this subsection, we explore the effect of these two parameters. The Porter-stemmed index is used. Fig. 6 shows the effect of the parameter $SubNum$, where $SubNum$ takes the values 5, 10, 20, 30 and "all", where $ModNum$ is fixed as 10. Here, "all" indicates using all subset queries generated.

Fig. 6 shows that the best number of subset queries used in the reformulation tree is inconsistent. On Gov2, the performance becomes stable after using the top 20 subset queries. On Robust04 and Wt10g, the performance keeps increasing when more subset queries are considered. On ClueWeb, the performance drops when more than the top 20 queries are used. One possible explanation for these observations is provided. Robust04 and Wt10g are relatively small collections, thus using more subset queries is likely to retrieve more relevant documents. However, when the size of the collection becomes very large such as Gov2, using more subset queries does not help much retrieval all relevant documents. If the collection is not only big but also contains much noise such as ClueWeb, using more subset queries even hurts the performance.

Table 7: The effect of the parameter $ModNum$ with respect to MAP

$ModNum$	Gov2	Robust04	Wt10g	ClueWeb
3	29.52	27.08	22.61	13.77
5	29.63	27.11	22.72	13.75
10	29.66	27.16	22.75	13.71

Table 7 displays the retrieval performance when $ModNum$ takes three different values, i.e. 3, 5 and 10, where $SubNum$ is set as 10.

Table 7 shows that there is not much performance change when $ModNum$ is bigger than 3, which indicates that modifying the top three subset queries is enough to achieve most of the performance of RTree.

7. EFFICIENCY

We now discuss the efficiency of using the reformulation tree model for retrieval. The online cost of this model comes from three aspects, i.e. reformulated query generation, query feature extraction, and retrieval.

The efficiency of the reformulated query generation depends on the reformulation operations involved. For example, generating the subset queries is very efficient. In contrast, generating query substitutions using the passage analysis is more time consuming, since it needs to analyze a lot of passages.

The efficiency of the query feature extraction also depends on the query features used. Some query features are expensive such as query clarity, while some features are relatively cheap such as the frequency in query logs.

Both of these steps can be optimized if large scale query logs are available. We can limit the reformulated queries to those appearing in query logs. In this way, instead of generating queries, we simply search the query logs, which can be efficiently implemented using the index. Also, all query features can be precomputed, which speeds up the query feature extraction.

In terms of the efficiency of retrieval, Eq. 1 shows that the retrieval score of each reformulated query ($sc(q_r, D)$) is required. At first glance, this appears to be inefficient, since we need to run multiple queries. However, this can be easily optimized. Eq. 7 shows that $sc(q_r, D)$ consists of the scores of the words and bigrams in q_r. Since all the reformulated queries in the reformulation tree are generated from the same original query, they share many words and bigrams. Thus, the scores of these words and bigrams can be reused by different reformulated queries. For example, the words and bigrams in the subset queries all come from the original query. Thus, we only need to calculate the scores for every word and bigram in the original query and then reuse these scores for each subset query. Further, although query substitutions may introduce new words and bigrams, the number of these new words and bigrams is limited. For example, Table 7 shows that query substitutions generated

from the top three subset queries are sufficient to achieve good retrieval performance.

8. CONCLUSION

Complex queries pose a new challenge to search systems. In order to combine different query operations and model the relationships between the reformulated queries, a new query representation is proposed in this paper, where the original query is transformed into a reformulation tree. A specific implementation is described for verbose queries, which combines subset query selection and query substitution within a principled framework. In the future, this query representation will be applied to other search tasks involving complex queries such as the cross-lingual retrieval and diversifying the search results.

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.

9. REFERENCES

[1] N. Balasubramanian, G. Kumaran, and V. Carvalho. Exploring reductions for long web queries. In *SIGIR10*, pages 571–578, 2010.

[2] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *SIGIR08*, pages 491–498, Singapore, 2008.

[3] M. Bendersky, D. Metzler, and W. B. Croft. Learning concept importance using a weighted dependence model. In *WSDM10*, New York City, NY, 2010.

[4] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: model and applications. In *CIKM08*, pages 609–618, 2008.

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML07*, pages 129–136, 2007.

[6] J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *SIGIR08*, pages 379–386, Singapore, 2008.

[7] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. In *SPIRE04*, pages 43–54, 2004.

[8] J. Jeon, W. B. Croft, and J. H. Lee. Finding similar questions in large question and answer archives. In *CIKM05*, pages 84–90, 2005.

[9] V. Jijkoun and M. de Rijke. Retrieving answers from frequently asked questions pages on the web. In *CIKM05*, pages 76–83, 2005.

[10] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW06*, pages 387–396, Ediburgh, Scotland, 2006.

[11] G. Kumaran and J. Allan. A case for shorter queries, and helping users creat them. In *ACL07*, pages 220–227, Rochester, New York, 2007.

[12] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *SIGIR09*, pages 564–571, Boston, MA, 2009.

[13] L. Larkey. A patent search and classification system. In *DL99*, pages 179–187, Berkeley, CA, 1999.

[14] V. Lavrenko and W. B. Croft. Relevance based language models. In *SIGIR01*, pages 120–127, New Orleans, LA, 2001.

[15] M. Lease. An improved Markov random field model for supporting verbose queries. In *SIGIR09*, pages 476–483, Boston, MA, 2009.

[16] M. Lease, J. Allan, and W. B. Croft. Regression rank: learning to meet the oppotunity of descriptive queries. In *SIGIR05*, pages 472–479, Salvador, Brazil, 2005.

[17] Q. Mei, K. Klinkner, R. Kumar, and A. Tomkins. An analysis framework for search sequences. In *CIKM09*, pages 1991–1994, 2009.

[18] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Information Processing and Management*, 40(5):735–750, 2004.

[19] D. Metzler and W. B. Croft. A Markov random field model for term dependencies. In *SIGIR05*, pages 472–479, Salvador, Brazil, 2005.

[20] D. Metzler and W. B. Croft. Latent concept expansion using markov random fields. In *SIGIR07*, pages 311–318, Amsterdam, the Netherlands, 2007.

[21] F. Peng, N. Ahmed, X. Li, and Y. Lu. Context sensitive stemming for web search. In *SIGIR07*, pages 639–646, Amsterdam, the Netherlands, 2007.

[22] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR98*, pages 275–281, Melbourne, Australia, 1998.

[23] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[24] S. Riezler, A. Vasserman, I. Tsochantaridis, V. Mittal, and Y. Liu. Statistical machine translation for query expansion in answer retrieval. In *ACL07*, pages 464–471, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[25] Y. Z. S. Cronen-Townsend and W. B. Croft. Predicting query performance. In *SIGIR02*, pages 299–306, 2002.

[26] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *CIKM08*, pages 479–488, Napa Valley, CA, 2008.

[27] X. Xue and W. B. Croft. Representing queries as distributions. In *SIGIR10 Workshop on Query Representation and Understanding*, pages 9–12, Geneva, Switzerland, 2010.

[28] X. Xue and W. B. Croft. Modeling subset distributions for verbose queries. In *SIGIR11*, pages 1133–1134, 2011.

[29] X. Xue, W. B. Croft, and D. A. Smith. Modeling reformulation using passage analysis. In *CIKM10*, pages 1497–1500, 2010.

[30] X. Xue, S. Huston, and W. B. Croft. Improving verbose queries using subset distribution. In *CIKM10*, pages 1059–1068, 2010.

[31] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR01*, pages 334–342, New Orleans, LA, 2001.