Can Short Queries Be Even Shorter?

Peilin Yang, Hui Fang
{franklyn,hfang}@udel.edu
Department of Electrical and Computer Engineering
University of Delaware

ABSTRACT

It is well known that query formulation could affect retrieval performance. Empirical observations suggested that a query may contain extraneous terms that could harm the retrieval effectiveness. This is true for both verbose and title queries. Given a query, it is possible that using its subqueries can generate more satisfying search results than using the original query. Although previous studies proposed method to reduce verbose queries, it remains unclear how we could reduce title queries given the short length of the title queries. In this paper, we focus on identifying the best performed subqueries for a given query. In particular, we formulate this problem as a ranking problem, where the goal is to rank subqueries of the query based on its predicted retrieval performance. To tackle this problem, we propose a set of novel post-retrieval features that can better capture relationships among query terms, and apply a learning-to-rank algorithm based on these features. Empirical results over TREC collections show that these new features are indeed useful in identifying the best subqueries.

CCS CONCEPTS

•Information systems →Query representation; Query reformulation;

ACM Reference format:
Peilin Yang, Hui Fang. 2017. Can Short Queries Be Even Shorter?. In *Proceedings of ICTIR'17, October 1–4, 2017, Amsterdam, The Netherlands, ,* 8 pages.
DOI: https://doi.org/10.1145/3121050.3121056

1 INTRODUCTION

The retrieval performance is closely related to the quality of a query. Not all terms in a query are equally important. Given a query, it is possible that its *subqueries*, i.e., the ones generated by removing terms from the query, can lead to better search results.

The problem of query reduction has been studied intensively for *verbose queries* (usually long, more than 6 terms, e.g., queries that are formulated based on the description of TREC topics) [1, 8, 13]. Previous studies showed that although a subquery does not always perform as well as the original query, the best subquery could be much better – 23% improvement in terms of MAP for verbose queries [1, 13]. However, reducing *keyword queries* (usually short, 2-6 terms in total, e.g. queries that are formulated based on the title of TREC topics) has drawn less attention than its counterpart. Previous study [8] showed that *title queries can also be reduced to obtain better performances* on ClueWeb collection. We made similar observations based on the results on other TREC collections too. Table 1 compares the performance when using the original keyword queries with those using the best performed subqueries. Although the table only contains the queries with length 3, it can be seen that the performance of using the best subqueries has more than 10% of gain in terms of the effectiveness. It is clear that *reducing keyword queries could lead to better performance*, but the problem is how to identify the best-performed subquery for a given query when we do not have any information about the relevance judgments.

Reducing keyword queries is a challenging task. Given a keyword query is already very short, how can we ever remove terms from that? One simplest solution would be to remove the terms based on their IDF values. Unfortunately, it does not work well. Let us consider query "pheromone scents work" (from WT10G). Among all the query terms, "work" has the lowest IDF. However, removing "work" from the query would not achieve our goal since the best-performed subquery for this query is "pheromone work". Similarly, other features, such as mutual information and clarity score [13] are not as useful as what they are supposed to be (more details in Section 3).

In this paper, we focus on the problem of identifying best-performed subquery for a given keyword query. In particular, we formulate the problem as ranking all the subqueries of a keyword query based on their predicted performance. To tackle this problem, we propose a set of novel features that can better capture the relations among query terms, and then apply a learning-to-rank algorithm to rank the subqueries based on these new features as well as some existing ones.

All the proposed new features are post-retrieval ones, meaning that they are computed based on the retrieval results. These features are designed to capture different relationships among query terms from different aspects: query term proximity, the aggregated ranking scores of query terms, and the compactness and position of term tensors. Specifically, *term proximity based features* are designed to capture the intuition that some query terms should be viewed as phrases as opposed to individual terms. Let us consider query "family leave law". Its best subquery is "family leave", which is a law code. And only when the two terms occur next to each other and in the right order in a document, we are sure that the document is relevant. To capture this intuition, we propose to compute the statistics of the ranking scores that are computed based on term dependency model [14] for each subquery. We also leverage the correlations between these ranking scores with the ranking scores of the original query for this category of features. Furthermore, we

Table 1: Comparison of the MAP between using original queries and optimal subqueries. Only queries of length 3 are shown and the ranking function is BM25

Collection	Original	Upper Bound	Diff.
Disk12	0.2597	0.2880	+10.9%
Disk45	0.2399	0.2772	+15.5%
AQUAINT	0.2107	0.2426	+15.1%
WT2G	0.3285	0.3580	+9.0%
WT10G	0.1720	0.2051	+19.2%
GOV2	0.3060	0.3221	+5.3%

proposed another set of *term score based* features that are designed to measure the balance between TF and IDF weighting [7]. These features are computed based on different ways of aggregating the term scores of individual query terms. The assumption is that these statistics could capture the key properties of the best subquery at the term score level and thus are useful. Finally, we proposed a set of features based on the *compactness and positions of the term score tensors.* For this set of features , we investigate the spatial properties of the term scores. We view the term scores from top ranked documents as tensors in the multi-dimensional space and then compute the compactness and the position of the tensors cluster.

Empirical results show that the proposed new features are effective in identifying the best-performed subquery. Moreover, we intensively analyze the important features by comparing the performance difference between the subset of features and all features. The results validate the utility of the proposed new features.

2 IDENTIFYING BEST SUBQUERY

We now discuss how to identify the best-performed subquery for a given keyword query.

2.1 Problem Formulation

Given a query, it is possible that some of the terms in the query are not informative and including them in the query could harm the performance. Thus, the goal of best-performed subquery identification is to identify the best query representation by using the terms in the original query. The solution could be the original query or part of the original query.

The best-performed subquery identification problem can be formally defined as follows. Given an arbitrary query $Q = \{t_1, t_2, ..., t_{|QL|}\}$ where t_i is the ith term in Q and $|QL|$ is the length (number of terms) of the query, let P^Q denote the power set of Q, which includes all possible subqueries of Q. Let f be a retrieval function used for ranking the documents in the collection for any query in P. Let $m(P, f)$ denote a metric for the ranking effectiveness of retrieval function f using query P. The best-performed subquery identification problem aims at finding a subquery $P^* = \arg\max_{P \in P^Q} m(P, f)$.

We formulate the best-performed subquery identification problem as a subqueries ranking problem. The ranking is based on the *predicted* performance of the subqueries without prior knowledge of relevance. More specifically, given a query, we will generate all the subqueries and rank the subqueries based on their predicted retrieval performance. We leverage existing learning-to-rank algorithm such as LambdaMART [3] and focus on feature identification in our study.

Table 2: Notations and Explanations

Notations	Explanations
$Q = \{t_1, t_2, ..., t_{\lvert QL \rvert}\}$	The original query and its terms
$\lvert QL \rvert$	Query length
$P^Q = \{q_1, q_2, ..., q_i, ...\}$	The power set of Q which contains all subqueries
q	The general notation for any subquery including the original query.
c	Ranking list cutoff position
d_i	Document i in the ranking list
$ds_{q,i}$	Ranking score of document i for query q
$L_{q,c}(f) = \{d_1, ..., d_c\}$	Ranking list of q using model f cutoff at c.
$\vec{SL}_{q,c}(f) = \{ds_{q,1}, ..., ds_{q,c}\}$	Ranking scores in $L_{q,c}(f)$.
$\vec{t}_{q,d_i}(f) = \{f_{t_1,d_i}, ..., f_{t_n,d_i}\}$	Terms scores of d_i for query q computed by model f. $n = \lvert QL \rvert$.
$\vec{TL}_{t_i,c}(f) = \{f_{t_i,d_1}, ..., f_{t_i,d_c}\}^T$	Column term scores for t_i.
$ML_{q,c}(f) = \{\vec{t}_{q,d_1}(f), ..., \vec{t}_{q,d_c}(f)\}$	Terms scores matrix of $L_{q,c}(f)$.
$g(\vec{x}), h(\vec{x}) \in \Re$	Feature function. One of **MIN**, **MAX**, **MAX-MIN** (difference), **MAX/MIN** (division), **SUM**, **MEAN**, **STD** (standard deviation), **GMEAN** (geometric mean)

2.2 Subquery Ranking Features

2.2.1 Terms Relationship Features. We introduce the newly proposed features that can better capture the relations among query terms – the motivation behind them as well as the detailed steps to compute them. These features are designed to capture different relationships among query terms from different aspects: query term proximity, the aggregated ranking scores of query terms, and the compactness and position of term tensors. The above mentioned features are *post-retrieval features* where we explore the scores in the ranking list of the subquery and generate the features from that. The variables and the notations that will be used in the following sections are summarized in Table 2.

Term Proximity Based Features (PXM) When identifying useful subqueries, it is important to consider the relations among terms in the subqueries. One of the important term relations is phrases. Intuitively, a subquery containing a phrase makes more sense than the combination of a few random terms. Let us consider an example query "family leave law". It is clear that its subquery "family leave" and "family law" are better choices than "leave law".

To distinguish subqueries with phrases from those without, we proposed to utilize the statistics of the ranking results when using the term dependency model [14]. When ranking documents using dependency model, the documents that have exact or close matching with the subquery will be favored. Thus, when a subquery is a phrase, the scores of the top-ranked results could have larger variance since some documents have the exact matching while others do not. On the contrary, when a subquery is not a phrase, the score variance is often small since few documents would have the exact matching. Clearly, we should use the statistics of ranking results based on the term dependency model as features for the

subquery ranking. More specifically, we first rank the documents in the collection using one of the following term proximity models: unordered window model (UW), ordered window model (OW) and the combination of the two models (UWOW). The window parameter (i.e. terms must appear with at most how many terms between each) wd is set to $4 \cdot (|QL| - 1)$. For example, a sample query of UWOW using Indri query language is
`#combine(#uw4(family leave) #ow4(family leave))`.
After getting the results, we extract high level statistics from the document scores at cutoff c as the features by applying the feature functions $h(\vec{x})$ to the scores. The feature function h is defined in Table 2 and it consists of a set of statistical functions that can be applied to a vector of values such as summation and standard deviation. The use of feature function was shown to be beneficial in order of aggregating the raw values in the previous studies [2, 5]. Formally, the features can be computed as follows:

$$PXM(w)_h = h(\vec{SL_{q,c}}(w)) \tag{1}$$

where $w \in UW, OW, UWOW$ and $\vec{SL_{q,c}}(w)$ is the documents scores vector of the term proximity model.

Another way to identify the phrase-based subquery is to examine how similar the search results are when using the term dependency model and when using the basic retrieval models. By comparing the the scores in the two ranking lists we might have more insights about the subqueries. Presumably, if a specific subquery performs much better than the original query because of the subquery is the key phrase in the original query, the scores of the two ranking lists should be different from each other. Take our previous subquery "family leave" for example, our method assumes that this subquery (actually the term proximity model) should have different ranking scores from the original query "family leave law" ranking scores and we are expected to capture such feature. Based on the above reasoning we measure the correlation between the documents scores of the term proximity model of the subquery and the regular ranking scores of the original query Q using Kendall's Tau (τ_B) and Pearson's r as two additional features. Formally, the correlation-based PXM features can be computed as:

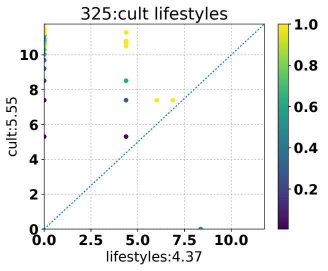
$$PXM(w)_{corr} = Corr(\vec{SL_{Q,c}}(w), \vec{SL_{q_i,c}}(w)) \tag{2}$$

where $Corr \in \{\tau_B, \rho\}$.

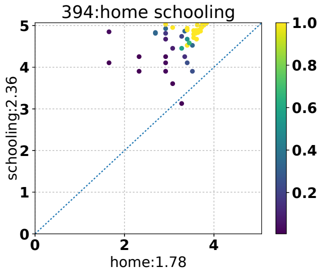
Term Score Based Features (TS)

For this set of features, we continue to explore the ranking list of the subqueries – the scores of individual query terms instead of the score of the document. The intuition of TS is originated from the term frequency constraint and the term discrimination constraint from previous work [7]. The constraint essentially introduces the balance between document term frequency (TF) and inverted document frequency (IDF). This really inspires us that there should be some interesting properties in the term score of top ranked documents. Instead of separately considering the TF and IDF, we choose to directly look at the individual term score computed by any ranking function that has reasonable TF and IDF components (e.g. BM25) for two reasons: (1) the ranking list is determined by the scores computed by the ranking function, and (2) the ranking function has the TF and IDF components and thus it already naturally adopts the TF-IDF constraint [7]. We wonder, for instance, do the top ranked documents have more balanced term scores or do they have highly skewed term scores? Or is the performance of subquery related to the minimum of the term scores in the top ranked documents? Figure 1 illustrates the intuition: the two subfigures are the term scores computed by BM25 model for the two queries for TREC keyword topics. From the figures we find two distinct patterns: for query "cult lifestyle" its relevant documents have higher probability along the y-axis indicating that "cult" is more important than "lifestyle" for this query. But for query "home schooling" the term scores are more balanced.

Figure 1: Individual term scores. Term scores are computed using BM25 model. Colors of the dots are the probability of relevant document at that point. Axis labels show the *IDF* values computed by $log\frac{N}{df}$.



(a) AQUAINT QID:325



(b) AQUAINT QID:394

The TS features are then computed as follows: we first generate the ranking list $\vec{L_{q,c}}(f)$ for subquery q using any ranking function that has reasonable TF and TD components. For each document d_i in $\vec{L_{q,c}}(f)$, we compute the score for each individual term. This would generate the term scores vector $\vec{t_{q,d_i}}(f)$ for d_i. We then apply the feature function h to $\vec{t_{q,d_i}}(f)$ to get the aggregated statistics for d_i as $h(\vec{t_{q,d_i}}(f))$. The result of this step is a list of statistics with each element corresponding to one document. We then apply the feature function g again to each column of the previously generated list $h(\vec{t_{q,d_i}}(f))$ to generate the final TS features. Formally, TS is computed as:

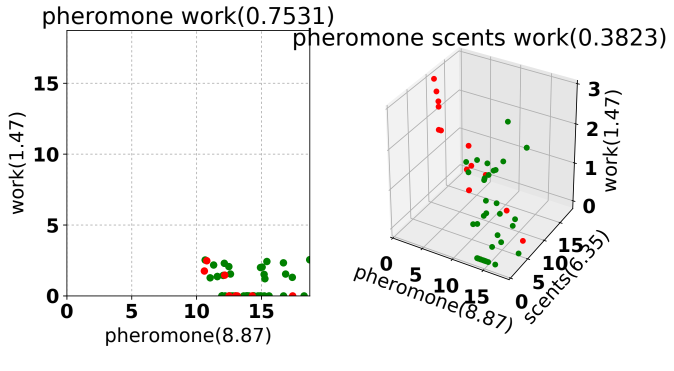
$$TS(f, h, g) = g(h(\vec{t_{q,d_i}}(f))) \tag{3}$$

For example, $TS(BM25, MEAN, STD)$ for query "home schooling" first rank the documents in the collection using BM25 function and then we compute the terms scores for "home" and "schooling" for each document in the ranking list again using BM25 function. The average value of terms scores for each document in the ranking list is then calculated and this results in a list of average values. Finally the standard deviation of the average values is computed and the value is served as the feature.

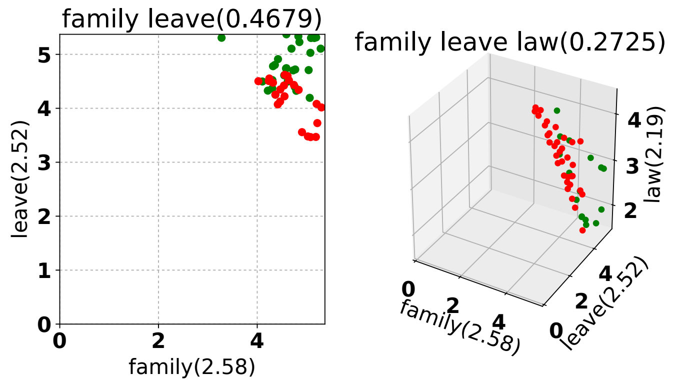
Compactness and Positions of Term Score Tensors (TCP)

Using document scores in the ranking list as query prediction features has been proposed in the previous studies [17]. In their work, the feature Normalized Query Commitment (NQC), which is essentially defined as the standard deviation of document scores in the ranking list, was used as a post-retrieval feature to predict the query performance. Larger deviation values were correlated with potentially lower query drift, and thus indicating the better

Figure 3: Terms scores (computed by BM25) of the top 50 ranked documents in the list. The numbers in the titles are the Average Precision of the corresponding subquery. Green dots are relevant documents and red dots are non-relevant documents. For each query only the optimal subquery and the original query are shown.



(a) WT10G QID:530



(b) ROBUST04 QID:648

effectiveness [17]. We also find the deviation and other statistics of ranking scores are indeed useful. However, we focus on the term level scores instead of document level scores.

Figure 3 shows two example queries from WT10G and ROBUST04 respectively. The x-axis and y-axis are the term scores computed by BM25 model and only the top 50 ranked documents are included in the figures. For both queries, the best subqueries are the queries with fewer query terms, i.e. not the original query. We find the similarity and difference for the chosen queries. First, it can be seen that for both best subqueries the term scores from top ranked documents are more compactly clustered. Second, the two queries are different in the sense that the term scores clusters are located at the different position in the two dimensional space. Such difference indicates that for the best subquery the ranking model has its own preference among query terms. For WT10G-530 "pheromone" receives much higher score. But for ROBUST04-648 both query terms receive similar scores. We name this category of features as compactness and positions of term score tensors since we intensively compute the all kinds of distances in the multi-dimensional term space and the term scores from the documents are essentially N dimensional vectors. We formally define three types of features in this category as follows:

- Tensor Compactness (**TCP(TC)**): The average and the standard deviation of the distances for the tensors to their centroid. This feature captures the compactness of the tensors cluster.

$$TCP(TC)_\mu = \frac{\sum_{d \in L_{q,c}(f)} ||\vec{t_{q,d_i}}(f), \vec{t_{q,d_\mu}}(f)||}{c} \tag{4}$$

$$TCP(TC)_\sigma = \sqrt{\frac{1}{c} \sum_{d \in L_{q,c}(f)} ||\vec{t_{q,d_i}}(f), TCP(TC)_\mu||^2} \tag{5}$$

where f is BM25 ranking model, $||T_A, T_B||$ is the distance between tensor A and tensor B, $\vec{t_{q,d_\mu}}$ is centroid of all the tensors in the list which is essentially

$$\vec{t_{q,d_\mu}}(f) = \left(\frac{\sum \vec{TL_{t_1,c}(f)}}{c}, \frac{\sum \vec{TL_{t_2,c}(f)}}{c}, \ldots \right) \tag{6}$$

- Tensor Closeness to Diagonal (**TCP(CDG)**): The distance from the tensors centroid to the diagonal line in multi-dimensional space, the average and the standard deviation of the distances from the tensors to the diagonal line in multi-dimensional space. These features capture part of the position information of the tensors.

$$TCP(CDG)_c = ||\vec{t_{q,d_\mu}}(f), l_{dg}|| \tag{7}$$

$$TCP(CDG)_\mu = \frac{\sum_{d \in L_{q,c}(f)} ||\vec{t_{q,d_i}}(f), l_{dg}||}{c} \tag{8}$$

$$TCP(CDG)_\sigma = \sqrt{\frac{1}{c} \sum_{d \in L_{q,c}(f)} ||\vec{t_{q,d_i}}(f), TCP(CDG)_\mu||^2} \tag{9}$$

where l_{dg} is the diagonal line in the multi-dimensional space and $||T, l||$ is the distance from tensor T to line l.

- Tensor Closeness to Nearest Axis (**TCP(CNA)**): We compute the distance from the tensors centroid to its nearest axis, the average/standard deviation distance from the tensors to the nearest axis in multi-dimensional space. TCNA and TCD together define the position property of the tensors.

$$TCP(CNA)_c = ||\vec{t_{q,d_\mu}}(f), l_{na}|| \tag{10}$$

$TCP(CNA)_\mu$ and $TCP(CNA)_\sigma$ can be computed similarly with Equation 8 and 9 with replacement of l_{dg} to l_{na} where l_{na} is the nearest axis to the centroid of all tensors and is computed as:

$$l_{na} = \min_{1 \le i \le N} ||\vec{t_{q,d_\mu}}, l_i|| \tag{11}$$

where l_i is ith-axis and N is the number of the dimensions.

We first computed the tensor closeness related features for the terms in the subquery q_i. Later on we found that it is beneficial to compute the tensor closeness related features for all the terms in the original query Q. We apply this in all our experiments.

2.2.2 Basic Features. Besides the aforementioned features (PXM, TS, TCP) we also applied other features proposed by others which we will further refer as "basic features".

Mutual Information We compute the mutual information by first counting the co-occurrence of pairwise terms within N terms window in the matching documents. The value is then normalized by the product of the document frequencies of the two terms. Finally we apply all possible feature functions h to the the pairwise terms list. The formula [13] is shown as follows:

$$MI = h(I(x, y)) = h\left(\frac{\frac{\sum O(x,y)}{T}}{\frac{O(x)}{T} \cdot \frac{O(y)}{T}}\right) \tag{12}$$

where $O(x, y)$ is the number of times term x and term y co-occur within a window of 50 terms in each matched documents, $O(t)$ is the total occurrence of term t in the collection and T is the total number of terms in the collection.

Collection Term Frequency (CTF)

The collection level term frequency of term t. Then we apply feature function h to the list of CTFs as $h(CTF_q)$.

Document Frequency (DF)

This is simply the document frequency for each term in the subquery $q_{i,j}$. Then we apply feature function h to the terms DFs as $h(DF_q)$.

Inversed Document Frequency (IDF)

The IDF here is the modified $log(IDF)$ component used in the modified BM25 model [6]:

$$IDF_t = log\frac{N+1}{DF_t}$$

where N is the number of documents in the collection. We then apply the feature function h to the list of IDF_t as $h(IDF_q)$.

Min Document Term Frequency (MINTF) and Max Document Term Frequency (MAXTF)

MINTF is the minimum term frequency in the collection and is computed as:

$$MINTF_t = \min_{1 \le i \le DF_t} TF_{t,d_i}$$

Similarly

$$MAXTF_t = \max_{1 \le i \le DF_t} TF_{t,d_i}$$

Final features are masked using feature function as $h(MINTF_q)$ and $h(MAXTF_q)$.

Average Document Term Frequency (AVGTF) and Standard Deviation Document Term Frequency (STDTF)

This AVGTF applies to each individual term as:

$$AVGTF_t = \frac{\sum_{i=1}^{DF_t} TF_{t,d_i}}{DF_t}$$

The STDTF is the standard deviation of $AVGTF_t$. We apply feature function masks to both features as $h(AVGTF_q)$ and $h(STDTF_q)$.

Average Document Term Frequency with IDF (AVGTFIDF)

We also incorporate the average document term frequency with IDF to capture the term salience in the collection. Formally we have:

$$AVGTFIDF_t = AVGTF_t \cdot IDF_t.$$

Table 3: Collections and Queries

Collection	#qry	$\lvert QL\rvert = 2$	$\lvert QL\rvert = 3$	$\lvert QL\rvert = 4$
Disk12	150	30(20%)	37(25%)	41(27%)
Disk45	250	75(33%)	147(59%)	17(7%)
AQUAINT	50	21(42%)	27(54%)	1(2%)
WT2G	50	24(48%)	23(46%)	0(0%)
WT10G	100	30(30%)	25(25%)	20(20%)
GOV2	150	44(29%)	65(43%)	35(23%)

Average Document Term Frequency with Collection Occurrence Probability (AVGTFCOP)

Similar to AVGTFIDF, we can also leverage the average collection occurrence probability to capture the term salience in the collection. Formally we have:

$$AVGTFCOP_t = AVGTF_t + \mu \cdot p(t|C)$$

where $p(t|C)$ is the probability of term t occurred in the whole collection and is computed as $p(t|C) = \frac{CTF_t}{|C|}$. $|C|$ is the total number of terms in the collection. We choose $\mu = 1000$ based on the preliminary results.

Simplified Clarity Score (SCS)

This feature was firstly proposed by He and Ounis [9] to reduce the computational cost of original query clarity and it was used as a pre-retrieval query performance predicator. It is computed as:

$$SCS_q = \sum_{t \in q} p(t|q) \cdot log_2 \frac{p(t|q)}{p(t|C)}$$

where $p(t|q)$ is the probability of term occurred in the query q.

3 EXPERIMENTS AND RESULTS

In this section we test our subquery ranking method using TREC collections and topics. We will describe the details of the experiment setup as well as the analysis of the results.

3.1 Experiment Setup

We use six TREC Ad-hoc/Web collections in our experiments: Disk12, Disk45 with ROBUST04 query set, AQUAINT News Collection with ROBUST05 query set (ROBUST04 hard queries), WT2G, WT10G and GOV2. The title part of the query topics is used to test the proposed subquery ranking method. Stopwords are removed from both the collections and the queries and porter stemmer is applied to the indexes. Table 3 lists the details of the collections and the corresponding queries. As we can see that for most title topics their lengths are within 2 to 4.

Since we are targeting the subquery ranking, single term queries will not be included [1]. Lots of features can not be directly applied to the queries of length 2 such as MI since the subqueries are single term query (other than the original query) and thus we separate the queries by their lengths. In our experiments we focus on queries of length 2 and 3 since even for queries of length 4 AQUAINT and WT2G do not have enough queries for both training and testing. When tested on one collection, queries from other 5 collections are used together as training examples. All the features are normalized to the range [0, 1] before being fed to the learning algorithm.

[1] [20] provided the performance upper bound for single term queries

LambdaMART is leveraged to rank the subqueries based on their features and the average precision of the subquery is used as the labels. Since LambdaMART favors the scalar labels we convert the AP values to integers based on the relative AP values distribution. The relative AP values are the differences between the AP of a subquery and the AP of the best subquery. The mapping from AP to integer should reflect the relative importance of a query whose best subquery performs much better than the rest of its subqueries. We do not show the actual distribution due to space limit and the rule of the mapping is:

$$Label(q) = \begin{cases} 4, & \text{if } AP_q = MaxAP_{OQ(q)} \\ 3, & \text{if } MaxAP_{OQ(q)} - AP_q \leq 0.1 \\ 2, & \text{if } MaxAP_{OQ(q)} - AP_q > 0.1 \cap MaxAP_{OQ(q)} - AP_q \leq 0.3 \\ 1, & \text{if } MaxAP_{OQ(q)} - AP_q > 0.3 \cap MaxAP_{OQ(q)} - AP_q \leq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

where q is a subquery, $OQ(q)$ denotes the original query of q, AP_q denotes the average precision of the subquery q, and $MaxAP_Q$ denotes the best AP of all the subqueries of Q. Basically we only care about which subquery should be labeled as the best-performed subquery thus the metric for LambdaMART is set to nDCG@1. The number of leaves for each tree is chosen from [2, 10] and the best performance averaged among all collections is reported. When generating the ranking list or compute the features like TS and TCP where ranking function is needed we always apply BM25 [2] with optimal parameter b ($k_1 = 1.2$ always) set based on the results reported in [21]. For performance metrics we report the accuracy and MAP. We also compare the MAP of best-performed subqueries identified by our method with the theoretical upper bound.

3.2 Results of Subquery Ranking

The results of subquery ranking (SR) using all features (mentioned in Section 2.2 and normalized) for queries of length 2 and 3 are listed in Table 4. The accuracy is computed as the number of queries whose best-performed subqueries are correctly identified divided by the total number of queries. MAP is computed by first picking the best subqueries identified by our model and then taking the average precision for these best subqueries. It can be seen that our SR method is better than using original query for most of the collections. The only exception is the GOV2 with queries of length 2 where the upper bound of the optimal performance is almost the same with the performance of using original queries. We also find that in general the our model performs better with queries of length 3 than the queries of length 2 in terms of percentage improvement. This is mainly because some features such as MI and PXM simply can not be applied to the queries of length 2.

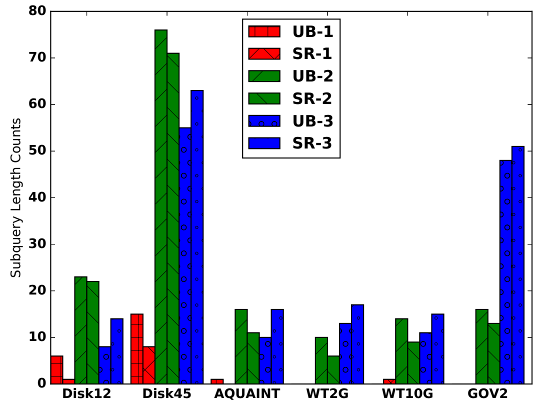
Figure 5 shows the length distribution of the best subqueries for the queries of length 3. Basically it shows how many queries of which its best subquery of length N. For example, for Disk12 there are 6 queries whose best subquery are of length 1 and 23 of the queries have their optimal subquery length of 2. From the figure we can see that in general our model has balanced number of best subqueries in different length and the numbers are very close to the upper bound. We also find that our model slightly favors the original queries as the number of best queries that have three terms in our model is always larger than the values of the upper bound.

[2] We also tried using Dirichlet language model and found using BM25 leads to slightly better performance.

Table 4: Results of using all features. *OG* represents the original query. *SR* represents our subquery ranking model. *UB* represents the upper bound where the optimal subquery for each original query is picked.

$\lvert QL \rvert$	Collection	Accuracy	MAP		
			OG	SR	UB
2	Disk12	90%	0.3216	0.3309 +2.89%	0.3372 +4.85%
	Disk45	82%	0.2506	0.2566 +2.39%	0.2662 +6.23%
	AQUAINT	76%	0.2063	0.2091 +1.36%	0.2184 +5.87%
	WT2G	83%	0.2983	0.2983 +0.00%	0.3083 +3.35%
	WT10G	83%	0.2544	0.2663 +4.68%	0.2738 +7.63%
	GOV2	96%	0.2912	0.2911 -0.03%	0.2913 +0.03%
3	Disk12	92%	0.2597	0.2833 +9.09%	0.2880 +10.90%
	Disk45	89%	0.2399	0.2643 +10.17%	0.2772 +15.55%
	AQUAINT	88%	0.2107	0.2323 +10.25%	0.2426 +15.14%
	WT2G	90%	0.3285	0.3380 +2.89%	0.3580 +8.98%
	WT10G	94%	0.1720	0.1949 +13.31%	0.2051 +19.24%
	GOV2	95%	0.3060	0.3113 +1.73%	0.3221 +5.26%

Figure 5: Optimal Subqueries Lengths of queries with 3 terms. *UB-1* denotes the number of ground truth best subqueries that has 1 term. *SR-1* denotes the number of subquery ranking model ranked best subqueries that has 1 term. *UB-2, UB-3, SR-2, SR-3* follow the same notation.



3.3 Feature Importance Analysis

In order to quantify the feature importance we set up another experiment in which a subset of features are taken off from the

the feature space and the performance difference between using all features and using the subset of features is compared. The results are in Table 5 and we only show the results of queries of length 3 due to space limit.

In Table 5 there are two main sections: the top important features from basic features are on the left and the detailed feature importance of terms relationship features on the right. For basic features the features with the largest performance drops if they were removed from the feature space are listed. For terms relationship features we present the details of PXM, TS and TCP by showing the importance of the sub-features. Sub-features are the features like $PXM(w)_h$ and $TCP(CDG)$ which essentially reflect specific intuitions of the newly proposed features. First, we notice that in general the terms relationship features have the performance drop larger than -13%. Comparing to the top basic features where two of them have the performance drop below than -13% it validates the utility of these features. Second, Detailed collection-wise performance drop indicates different features contribute differently for the collections. For example, AVGTFCOP is important to AQUAINT while TS(SUM,SUM) is specifically useful for WT2G. Third, detailed analysis on terms relationship features reveal that: for PXM $PXM(w)_h$ is better than $PXM(w)_corr$. For TS features TS(SUM,SUM) which is actually the sum of the top ranked documents scores (the first sum of all query terms equals to the document score) is more vital. For TCP features the TCP(CNA) which captures the position of the terms scores tensors and the TCP(TC) which captures the tensors compactness are all important with performances drops larger than -14% and this validates the utility of such features.

4 RELATED WORK

Reducing verbose queries to shorter queries has been intensively studied in recent years [1, 13].

Most previous work involves in generating the features for either the original query Q, the subquery q or groups of terms. Basically there are several features categories:

Statistical Features TF-IDF based features are the most widely used set of statistical features [2, 5, 10, 13, 16, 18] which include various statistics such as collection TF, IDF, residual IDF, TF in matching Wikipedia titles, count of passages containing the subquery etc. Other popular features include simplified clarity score [5, 13] and mutual information (MI) between query terms [11, 12, 19], domain specific dictionaries based features such as whether the term indicating a brand [19].

Query Features Query features are only based on the query itself and no collection context is involved. Similarity Original Query measures the cosine similarity between TF-IDF vectors of each subquery and the original query [13]. Presence of Stop Words [1, 16] computes the ratio of stop words in subquery. IsRightMost and IsLeftMost [19] are the features that capture the position of the subquery in the original query.

Term Dependency Features These features capture the dependencies between query words. Park et al. [16] proposed four types of dependencies among query terms: parent-child, ancestor-descendant, siblings and c-commanding. The final features include the number of dependent clauses in the query; the ratio of the dependent term pairs which have parent-child; ancestor-dependent, siblings, and c-commanding in the query.

Post Retrieval Features These features are based on the ranking results of subqueries. Typically, these features are expensive to compute but they have been proven to be effective. Query-document Relevance Scores [1, 4] are the LambdaRank and BM25 scores of top K documents, the click through counts of top K documents and the page-rank scores of top K documents. Query scope [13] of a subquery is the size of the retrieved document set relative to the size of the collection. Weighted Information Gain [2] is the difference of the retrieval quality by comparing the state where only the average document is retrieved to the state where the actual results are observed. Query drift among results [5] is a set of features which include the standard deviation of the ranking scores at 100 documents, the maximum standard deviation in the ranking list, etc.

Our proposed features are all *post-retrieval features* and some of them share the similar ideas with previous studies, e.g. term proximity based features are inspired by the term dependency features mentioned above. Different from the previous work, we are interested in the document score of the proposed term proximity models and various properties of the term scores.

A large number of research efforts have been made towards combining the features using a classification or a regression model. The classification problem is equivalent to pick the best subquery and it typically decides whether a term in the original query should be included in the best subquery. The regression problem is to learn a weight for each term denoting its importance score or to learn a weight for a sub-query; the top terms or the subquery with highest weight is then chosen. RankSVM [1, 13, 15], decision trees, AdaBoost, logistic regression [19] are popular classification models while random forests [1] is the most popular regression model. We adopt the classification model in our work where we apply the LambdaMART to the features from all subqueries and the model learns which subquery should be the best subquery.

5 CONCLUSIONS AND FUTURE WORK

This paper focuses on the problem of identifying the best-performed subquery for a given keyword query. Our main contribution lies in the identification of new features that can be used to predict the performance of a subquery. These new features are designed to capture different types of term relations based on the retrieval results, i.e., the proximity among query terms, the aggregated relevance score of query terms, and the compactness and positions of the term score tensors. The newly proposed features together with other basic features are used to train a LambdaMART model to identify the best subqueries. Experiment results show that by using the terms relationship features together with other popular basic features could lead to promising performance in terms of both accuracy and MAP. Detailed analysis on feature importance validates the usefulness of our proposed features.

One of the limitations of the proposed features is related to the computational cost. Since these features are post-retrieval, it would inevitably increase the processing time and make it less practical to apply this method to reduce the keyword queries online.

Table 5: Feature importance analysis for queries of length 3, the lower the better. The lowest value of each collection is bolded.

Collection	Top Basic Features			Terms Relationship Features							
	AVGTFCOP	SCS	CTF	$PXM(w)_h$	$PXM(w)_{corr}$	TS1	TS2	TS3	TCP(TC)	TCP(CDG)	TCP(CNA)
Disk12	0.2399	0.2497	0.2445	0.2535	0.2497	0.2532	0.2536	0.2536	**0.2374**	0.2492	0.2498
0.2833	-15.3%	-11.9%	-13.7%	-10.5%	-11.9%	-10.6%	-10.5%	-10.5%	**-16.2%**	-12.0%	-11.8%
Disk45	0.2292	0.2293	**0.2224**	0.2306	0.2354	0.2294	0.2313	0.2313	0.2330	0.2337	0.2267
0.2643	-13.3%	-13.2%	**-15.9%**	-12.8%	-10.9%	-13.2%	-12.5%	-12.5%	-11.8%	-11.6%	-14.2%
AQUAINT	**0.1882**	0.1999	0.2003	0.1974	0.1970	0.2029	0.1949	0.1949	0.1884	0.2005	0.1999
0.2323	**-19.0%**	-13.9%	-13.8%	-15.0%	-15.2%	-12.7%	-16.1%	-16.1%	-18.9%	-13.7%	-13.9%
WT2G	0.2561	0.2756	0.2853	0.2785	0.2760	0.2641	**0.2191**	0.2191	0.2828	0.2826	0.2795
0.3380	-24.2%	-18.5%	-15.6%	-17.6%	-16.3%	-21.9%	**-35.2%**	-35.2%	-16.4%	-17.3%	-18.3%
WT10G	0.1587	0.1643	0.1663	0.1454	0.1510	0.1671	0.1617	0.1617	0.1589	**0.1435**	0.1441
0.1949	-18.6%	-15.7%	-14.7%	-25.4%	-22.5%	-14.3%	-17.0%	-17.0%	-18.5%	**-26.4%**	-26.1%
GOV2	0.3040	0.2990	0.3029	0.2989	0.3025	**0.2857**	0.2927	0.2947	0.3046	0.3087	0.2990
0.3113	-2.3%	-4.0%	-2.7%	-4.0%	-2.8%	**-8.2%**	-6.0%	-5.3%	-2.2%	-0.8%	-4.0%
AVG	0.2294	0.2363	0.2370	0.2341	0.2364	0.2337	**0.2256**	0.2259	0.2342	0.2359	0.2329
0.2707	-15.5%	-12.9%	-12.7%	-14.2%	-13.3%	-13.5%	**-16.2%**	-16.1%	-14.0%	-13.6%	-14.7%

TS1: TS(MAX/MIN,SUM); TS2: TS(SUM,SUM); TS3: TS(GMEAN,MEAN)

We acknowledge this limitation and plan to study more efficient ways of computing features in our future work. However, we also want to emphasize that one benefit of the proposed new features is to help us better understand the impact of term relations on the retrieval performance. And these features might shed some lights on developing more effective retrieval functions.

As for future work, there are two interesting directions. The first one is to continue on the features so that the identification of the best subquery can be further improved. For example, the semantic features would be the promising perspective. Incorporating the outside resources as the potential feature source would be another choice. The second direction is to leverage the terms relationship features and the experimental results presented in this paper to do more theoretical studies. For example, we could get inspiration from the features and try to prove the performance upper bound of multiple-terms queries.

Acknowledgments. This research was supported by the U.S. National Science Foundation under IIS-1423002.

REFERENCES

[1] N. Balasubramanian, G. Kumaran, and V. R. Carvalho. Exploring reductions for long web queries. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 571–578, New York, NY, USA, 2010. ACM.

[2] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 491–498, New York, NY, USA, 2008. ACM.

[3] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, June 2010.

[4] Y. Chen and Y.-Q. Zhang. A query substitution-search result refinement approach for long query web searches. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, pages 245–251, Washington, DC, USA, 2009. IEEE Computer Society.

[5] R. Cummins, M. Lalmas, C. O'Riordan, and J. M. Jose. Navigating the user query space. In *Proceedings of the 18th International Conference on String Processing and Information Retrieval*, SPIRE'11, pages 380–385, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 49–56, New York, NY, USA, 2004. ACM.

[7] H. Fang, T. Tao, and C. Zhai. Diagnostic evaluation of information retrieval models. *ACM Trans. Inf. Syst.*, 29(2):7:1–7:42, Apr. 2011.

[8] H. Fang and H. Wu. An exploration of query term deletion. 2011.

[9] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. In *In Proc. Symposium on String Processing and Information Retrieval*, pages 43–54. Springer Verlag, 2004.

[10] S. Huston and W. B. Croft. Evaluating verbose query processing techniques. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 291–298, New York, NY, USA, 2010. ACM.

[11] G. Kumaran and J. Allan. A case for shorter queries, and helping users create them. In *HLT-NAACL*, pages 220–227, 2007.

[12] G. Kumaran and J. Allan. Effective and efficient user interaction for long queries. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 11–18, New York, NY, USA, 2008. ACM.

[13] G. Kumaran and V. R. Carvalho. Reducing long queries using query quality predictors. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 564–571, New York, NY, USA, 2009. ACM.

[14] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 472–479. ACM, 2005.

[15] J. H. Park and W. B. Croft. Query term ranking based on dependency parsing of verbose queries. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 829–830, New York, NY, USA, 2010. ACM.

[16] J. H. Park, W. B. Croft, and D. A. Smith. A quasi-synchronous dependence model for information retrieval. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 17–26, New York, NY, USA, 2011. ACM.

[17] A. Shtok, O. Kurland, D. Carmel, F. Raiber, and G. Markovits. Predicting query performance by query-drift estimation. *ACM Trans. Inf. Syst.*, 30(2):11:1–11:35, May 2012.

[18] X. Xue, S. Huston, and W. B. Croft. Improving verbose queries using subset distribution. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1059–1068, New York, NY, USA, 2010. ACM.

[19] B. Yang, N. Parikh, G. Singh, and N. Sundaresan. A study of query term deletion using large-scale e-commerce search logs. In *Proceedings of the 36th European Conference on IR Research on Advances in Information Retrieval - Volume 8416*, ECIR 2014, pages 235–246, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[20] P. Yang and H. Fang. Estimating retrieval performance bound for single term queries. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 237–240, New York, NY, USA, 2016. ACM.

[21] P. Yang and H. Fang. A reproducibility study of information retrieval models. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 77–86, New York, NY, USA, 2016. ACM.