Pseudo-Query Reformulation

Fernando Diaz
Microsoft
fdiaz@microsoft.com

ABSTRACT

Automatic query reformulation refers to rewriting a user's original query in order to improve the ranking of retrieval results compared to the original query. We present a general framework for automatic query reformulation based on discrete optimization. Our approach, referred to as *pseudo-query reformulation*, treats automatic query reformulation as a search problem over the graph of unweighted queries linked by minimal transformations (e.g. term additions, deletions). This framework allows us to test existing performance prediction methods as heuristics for the graph search process. We demonstrate the effectiveness of the approach on several publicly available datasets.

1. INTRODUCTION

Most information retrieval systems operate by performing a single retrieval in response to a query. Effective results sometimes require several manual reformulations by the user [6, 25, 22] or semi-automatic reformulations assisted by the system [21, 36, 23]. Although the reformulation process can be important to the user (e.g. in order to gain perspective about the domain of interest), the process can also lead to frustration and abandonment [14].

In many ways, the core information retrieval problem is to improve the initial ranking and user satisfaction and, as a result, reduce the need for reformulations, manual or semi-automatic. While there have been several advances in learning to rank given a fixed query representation [29], there has been somewhat less attention, from a formal modeling perspective, given to automatically reformulating the query before presenting the user with the retrieval results. One notable exception is pseudo-relevance feedback (PRF), the technique of using terms found in the top retrieved documents to conduct a second retrieval [1, 9]. PRF is known to be a very strong baseline. However, it incurs a very high computational cost because it issues a second, much longer query for retrieval.

In this paper, we present an approach to automatic query reformulation which combines the iterated nature of human query reformulation with the automatic behavior of PRF. We refer to this process as *pseudo-query reformulation* (PQR). Figure 1 graphically illustrates the intuition behind PQR. In this figure, each query and its retrieved results are depicted as nodes in a graph. An edge exists between two nodes, q_i and q_j, if there is a simple reformulation from q_i to q_j; for example, a single term addition or deletion. This simulates the incremental query modifications a user might conduct during a session. The results in this figure are colored so that red documents reflect relevance. If we assume that a user is following a good reformulation policy, then, starting at q_0, she will select reformulations (nodes) which incrementally increase the number of relevant documents. This is depicted as the path of shaded nodes in our graph. We conjecture that a user navigates from q_i to q_j by using insights from the retrieval results of q_i (e.g. q_j includes a highly discriminative term in the results for q_i) or by incorporating some prior knowledge (e.g. q_j includes a highly discriminative term in general). PQR is an algorithm which behaves in the same way: issuing a query, observing the results, inspecting possible reformulations, selecting a reformulation likely to be effective, and then iterating.

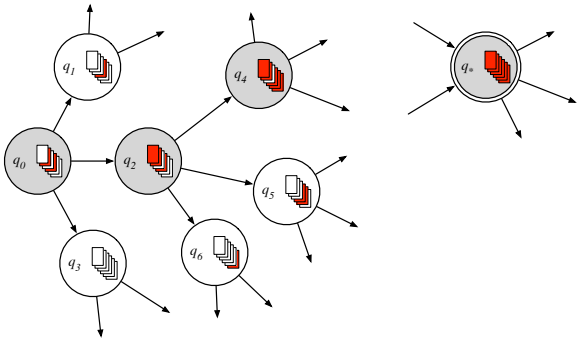


Figure 1: Query reformulation as graph search. Nodes represent queries and associated retrieved results. Relevant documents are highlighted in red. Edges exist between nodes whose queries are simple reformulations of each other. The goal of pseudo-query reformulation is to, given a seed query q_0 by a user, automatically navigate to a better query.

Several properties make PQR attractive. First, PQR directly optimizes performance for short, unweighted keyword interaction. This is important for scenarios where a searcher, human or artificial, is constrained by an API such as those found in many search services provided by general web search engines or social media sites. This constraint prevents the use of massive query expansion techniques such as PRF. Even if very long queries were supported, most modern systems are optimized (in terms of efficiency and effectiveness) for short queries, hurting the performance of massive query expansion. Second, our experiments demonstrate that PQR significantly outperforms several baselines, including PRF. Finally, PQR provides a framework in which to evaluate performance prediction methods in a grounded retrieval task.

2. RELATED WORK

Pseudo-query reformulation draws together three areas of information retrieval: pseudo-relevance feedback, iterative query rewriting, and performance prediction. Previous research has combined elements of these, but not in the way described in our work.

Kurland *et al.* present several heuristics for iteratively refining a language model query by navigating document clusters in a retrieval system [24]. The technique leverages specialized data structures storing document clusters derived from large scale corpus analysis. While related, the solution proposed by these authors violates assumptions in our problem definition. First, their solution assumes weighted language model style queries not supported by backends in our scenario. Second, their solution assumes access to the entire corpus as opposed to a search API.

Using performance predictors in order to improve ranking has also been studied previously, although in a different context. Sheldon *et al.* demonstrate how to use performance predictors in order to better merge result lists from pairs of reformulated queries [40]. This is, in spirit, quite close to our work and is a special case of PQR which considers only two candidate queries and a single iteration instead of hundreds of candidates over several iterations. In the context of learning to rank, performance predictors have been incorporated as ranking signals and been found to be useful [32]. From the perspective of query weighting, Lv and Zhai explored using performance predictors in order to set the optimal interpolation weight in pseudo-relevance feedback [31]. Similarly Xue and Croft have demonstrated how to use performance predictors in order to improve concept weighting in an inference network model[44, 43]. Again, while similar to our work in the use of performance predictors for query reformulation, we focus on the discrete, iterated representation. The work of Xue and Croft focuses on a single iteration and a weighted representation. More generally, there has been some interest in detecting the importance of query terms in a long queries or in expanded queries [3, 7, 27, 46, 2].

Representing related queries as graphs has been studied extensively. Early work by Mooers proposed treating the entire space of unweighted queries (i.e. length $|\mathcal{V}|$ boolean vectors) as a lattice [35]. In the context of web search, Boldi *et al.* studied within-session query reformulations as a graph [5]. Other work, such as spreading activation and inference networks as well as term-only graphs are less related although they use a similar formalism.

3. MOTIVATION

As mentioned earlier, users often reformulate an initial query in response to the system's ranking [6, 25, 41, 22]. Reformulation actions include adding, deleting, and substituting query words, amongst other transformations. There is evidence that manual reformulation can improve the quality of a ranked list for a given information need [22, Table 5]. However, previous research has demonstrated that humans are not as effective as automatic methods in this task [15, 33, 39].

In order to estimate an upper bound on the potential improvement from reformulation, we propose a simulation of an optimal user's reformulation behavior. Our simulator is based on query-document relevance judgments, referred to as qrels. Previous research has used similar techniques to examine the optimality of human reformulation behavior [15, 33, 39]. In this section, we revisit these results with contemporary test collections and retrieval methods. Unlike this prior work, though, we are not interested in determining the human (in)ability to achieve optimal performance but in gauging the upper bound for PQR.

We sketch our query reformulation simulator in Figure 2. The simulator is inspired by a model of optimal human search behavior and should not be considered model of any real user. Our recursive search algorithm uses as input: a reference query q (e.g. a TREC 'title' query), a set of qrels, r for q, a current depth, d, and a maximum depth, $d_{\max}$. The process can be considered a depth-limited graph search by a oracle on the space of queries depicted in Figure 1. The simulated search begins by generating a set of candidate reformulations, $\mathcal{Q}^q$, from an initial query, q.

The next step in our simulation selects the best reformulation from this set of candidates. We assume that the oracle can measure the performance μ of the set of candidate reformulations by running each query against the retrieval system and compute a metric such as NDCG with r. After selecting this query, we rerun the process on the best reformulation, q^*. Our search terminates after it reaches a specified depth, $d_{\max}$. We introduce $d_{\max}$ in order to limit computation and resource usage.

Before describing this experiment and results in more detail, we want to make the assumptions of our model clear. First, the effectiveness of the query found by this simulation is constrained by the query representation. For example, if our query is an unweighted term vector, then, even if we could exhaustively evaluate all $2^{|\mathcal{V}|}$ possible queries, we may not find a query achieving the upper bound of the metric (i.e. 1 for most information retrieval metrics). Therefore, we refer to the *representational upper bound* as the best performance possible using a fixed query representation. The upper bound found by this simulation is also constrained by the fact that we are performing a local search. As such, we assume that a better query is reachable from q_0 through a series of query reformulations. We do not want to claim that the representational upper bound is reachable or even that a very good query is reachable, only that a better query than q_0 is reachable. Fortunately, the previously cited work in human and automatic query reformulation supports this claim. More subtly, we assume that these 'better queries' are reachable through a series of reformulations with increasing performance. If the better queries are reachable but cannot be navigated to by progressively getting better results, then we will not be able to attain better performance using relevance information. Unfortunately, this assumption has less justification and we must take it as is. Note that this assumption does not claim that *all* reformulations $\mathcal{Q}_{q_0}$ are better than q_0; only that there exists a better query that is 'closer' to even better queries. Because of these added constraints, we refer to the outcome of this process as the *search-restricted representational upper bound.*

For a random sample of 50 judged training queries, we ran the simulator described in Figure 2 using the following methods. The set of candidates consists of all one word deletions and 10 one word additions taken from the 10 ten most frequent words in the retrieval results for q. We considered two implementations of ScoreQueries: oracle prediction and random prediction. Oracle prediction computes NDCG@30. We select this high-precision measure for two

```
QRSim(q, d, d_max, r)
    q          ▷ current query
    d          ▷ current depth
    d_max      ▷ maximum depth
    r          ▷ relevance judgments
 1  if d = d_max
 2     then
 3          return q
 4  Q^q ← GenerateCandidateReformulations(q)
 5  μ ← ScoreQueries(Q^q, r)
 6  q* ← argmax_{q_i ∈ Q^q} μ_{q_i}
 7  if (q* = q)
 8     then
 9          return q
10     else
11          return QRSim(q*, d + 1, d_max, r)
```

Figure 2: Reformulation simulator. Given a query q and query-document relevance judgments r, this algorithm will perform gradient ascent on query performance, μ, over the space of query reformulations, $\mathcal{Q}$. The oracle policy uses r to compute true reformulation performance in ScoreQueries. The random policy uses a random number generator for this function.

	trec12	robust	web
QL	0.4011	0.4260	0.1628
RM3	0.4578	0.4312	0.1732
random	0.3162	0.2765	0.0756
PQR*	0.6482	0.6214	0.3053

Table 1: NDCG@30 for random (random) and optimal (PQR*) pseudo-query reformulation compared to query likelihood (QL) and relevance model (RM3). Datasets are described in Section 7.1.

reasons. First, our simulation needs to operate quickly and retrieving shorter lists is much more efficient. Second, NDCG is superior at distinguishing high precision runs compared to other measures such as mean average precision [37]. Random prediction scores reformulation candidates using a random scalar in the unit range. Starting at q_0, we search up to a depth of four. Further details of our corpora and queries can be found in Section 7.

The results of these experiments (Table 1) demonstrate the range of performance for PQR. Our oracle simulator performs quite well, even given the limited depth of our search. Performance is substantially better than the baseline, with relative improvements greater than those in published literature. To some extent this should be expected since the oracle can leverage relevance information. Surprisingly, though, the algorithm is able to achieve this performance increase by adding and removing a small set of up to four terms. The poor performance of the random policy suggests that oracle is not just using the terms selected by the initial retrieval to get its boost in performance.

Keeping this search-restricted representational upper bound in mind, we would like to develop algorithms that can approximate the behavior of our optimal policy *without having access to any qrels or an oracle*. The closer our automatic reformulation is to oracle, the better our performance.

4. PROBLEM DEFINITION

Let $\mathcal{Q}$ be the entire set of queries submittable to a retrieval system. In the case of unweighted keyword queries, this is all boolean vectors of dimension $|\mathcal{V}|$. For each query q, we define a set of reformulation candidates, $\mathcal{Q}_q$, consisting of all queries reachable by a single term addition or deletion. For example, the reformulation candidate set for the query `[hello world]` would include `[hello]`, `[world]`, `[hello world program]`, `[hello world song]`, amongst the $O(|\mathcal{V}|)$ other queries resulting from a single term addition. Our problem can be stated as follows: given an initial query, q_0, and access to the candidate generation function, find a query q_+ that performs better than q_0. Performance here is measured by submitting a query to a fixed retrieval system and evaluating results according to a fixed metric (e.g. NDCG@30). As mentioned earlier, this can be considered a graph search problem where queries are nodes and edges exist between q and $\mathcal{Q}_q$. Importantly, our algorithm has access to the unweighted keyword retrieval system in order to generate features, but *it never has access to any true relevance information or performance metric*. Such retrieval services can be found in search APIs such as those provided by major search engines, social media sites, and distributed information retrieval services.

5. ALGORITHMS

Conceptually, PQR follows the framework of the simulator from Figure 2. That is, the algorithm recursively performs candidate generation and candidate scoring within each recursion. In this section, we will describe candidate set generation (Section 5.1) and candidate scoring (Section 5.2) along with the graph search algorithm (Section 5.3).

5.1 Generating Candidates

Our entire search space can be represented by a very large lattice of queries. Even if we were performing local graph search, the $O(|\mathcal{V}|)$ edges incident to any one node would make a single iteration computationally intractable. As a result, we need a method for pruning the full set of reformulation candidates to a smaller set that we can analyze in more detail. Fortunately, in many cases, we can establish heuristics so that we only consider those reformulations likely to improve performance. For example, reformulating the query `[Master theorem]` into `[Master theorem yak]` seems unlikely to improve performance if we believe `yak` is unlikely to occur in documents relevant to `[Master theorem]`. In our case, given q_t, we consider the following candidates, *a*) all single term deletions from q_t, and *b*) all single term additions from the n terms with the highest probability of occurring in relevant documents. Since we do not have access to the relevant documents at runtime, we approximate this distribution using the terms occurring in the retrieval for q_t. Specifically, we select the top n terms in the relevance model, $\theta_{\mathcal{R}_t}$, associated with q_t [26]. The relevance model is the retrieval score-weighted linear interpolation of retrieved document languages models. We adopt this approach for its computational ease and demonstrated effectiveness in pseudo-relevance feedback.

5.2 Scoring Candidates

The candidate generation process described in Section 5.1 provides a crude method for pruning the search space. Based

on our observations with the random and oracle policies in Section 3, we know that inaccurately scoring reformulation candidates can significantly degrade the performance of a scoring algorithm. In this section, we model the oracle using established performance prediction signals.

5.2.1 Performance Prediction Signals

Performance prediction refers to the task of ordering a set of queries without relevance information so that the better performing queries are ordered above worse performing queries. With some exception, the majority of work in this area has focused on ranking queries coming from different information needs (i.e. one query per information need). We are interested in the slightly different task of ranking many queries for a single information need. Despite the difference in problem setting, we believe that, with some modifications discussed in Section 5.2.2, performance predictors can help model the oracle or, more accurately, the true performance of the reformulation. A complete treatment of related work is beyond the scope of this paper but details of approaches can be found in published surveys (e.g. [16]).

The set of performance predictors we consider can be broken into three sets: query signals, result set signals, and drift signals. Throughout this section, we will be describing signals associated with a candidate query q.

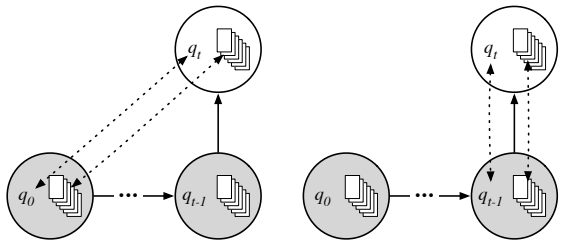
Query signals refer to properties of the terms in q alone. These signals are commonly referred to as 'pre-retrieval' signals since they can be computed without performing a costly retrieval. Previous research has demonstrated that queries including non-discriminative terms may retrieve non-relevant results. The inverse document frequency is one way to measure the discrimination ability of a term and has been used in previous performance prediction work [18]. Over all query terms in q, we consider the mean, maximum, and minimum IDF values. In addition to IDF, we use similarly-motivated signals such as Simplified Clarity (SC) and Query Scope (QS) [19].

Result set signals measure the quality of the documents retrieved by the query. These signals are commonly referred to as 'post-retrieval' signals. These features include the well-known Query Clarity (QC) measure, defined as the Kullback-Leibler divergence between the language model estimated from the retrieval results, $\theta_{\mathcal{R}_t}$, and the corpus language model, θ_C [10]. In our work, we use $\mathcal{B}(\mathcal{R}_t, \theta_C)$, the Bhattacharyya correlation between the corpus language model and the query language model [4], defined as

$$\mathcal{B}(\theta_i, \theta_j) = \sum_{w \in \mathcal{V}} \sqrt{p(w|\theta_i) \times p(w|\theta_j)} \quad (1)$$

This measure is in the unit interval and with low values for dissimilar pairs of language models and high values for similar pairs of language models. The Bhattacharyya correlation has been used effectively other other retrieval tasks [12]. We use the Bhattacharyya correlation between these two distributions instead of the Kullback-Leibler divergence because the measure is bounded and, as a result, does not need to be rescaled across queries. We also use the score autocorrelation (SA), a measure of the consistency of scores of semantically related documents [11]. In our implementation, we again use the Bhattacharyya correlation to measure the similarity between all pairs of documents in $\mathcal{R}_t$, as represented by their maximum likelihood language models.

Drift signals compare the current query q_t with its parent q_{t-1} and the initial query q_0 (Figure 3). These signals can serve to anchor our prediction and avoid query drift, situations where a reformulation candidate appears to be high quality but is topically very different from the desired information need. One way to measure drift is to compute the difference in the query signals for these pairs. Specifically, we measure the aggregate IDF, SC, and QS values of the deleted, preserved, and introduced keywords.



(a) Initial Query (b) Parent Query

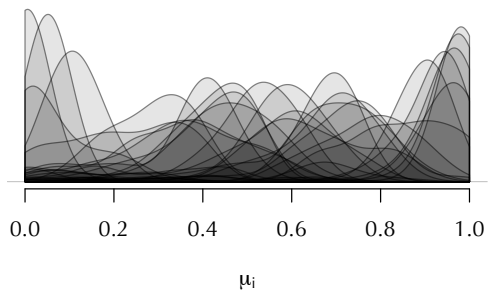
Figure 3: Drift signal classes. Signals for q_t include comparisons with reference queries q_{t-1} and q_0 to prevent query drift.

We also generate two signals comparing the results sets of these pairs of queries. The first measures the similarity of the ordering of retrieved documents. In order to do this, we compute the τ-AP between the rankings [45]. The τ-AP computes a position-sensitive version of Kendall's τ suitable for information retrieval tasks. The ranking of results for a reformulation candidate with a very high τ-AP will be indistinguishable from those of the reference query; the ranking of results for a reformulation candidate with a very low τ-AP will be quite different from the reference query. Our second result set signal measures drift by inspecting the result set language models. Specifically, it computes $\mathcal{B}(\theta_{\mathcal{R}_{t-1}}, \theta_{\mathcal{R}_t})$, the Bhattacharyya correlation between the result sets.

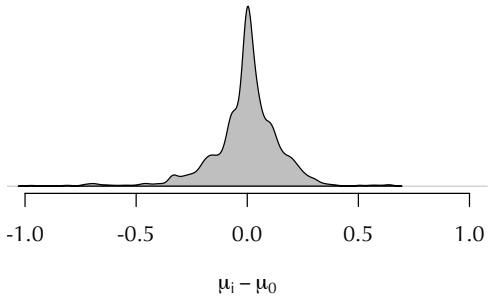
5.2.2 Performance Prediction Model

With some exception, the majority of performance prediction work has studied predictors independently, without looking at a combinations of signals. Several approaches to combine predictors focus on regressing against the the absolute performance for a set of training queries [13, 17]. This is appropriate when the task is to rank queries from different information needs but it may not be when the task is to predict the performance for reformulation candidates related to the same information need.

In order to demonstrate the problem with regressing against the uncalibrated performance metric for all queries, it is worth inspecting the training data for such an algorithm. In Figure 4a, we overlay the distributions of performance metric values for 28 information needs. Each distribution is a kernel density estimate based on the performance metric values observed when following the graph search algorithm in Section 3. The figure shows that the relative importance of a reformulation candidate depends strongly on the information need. Different information needs–as represented by different initial queries–have different mean performance values and, at times, variances. In fact, the diversity of performance ranges varies dramatically based on the information need, its representation in the corpus, and its complexity;

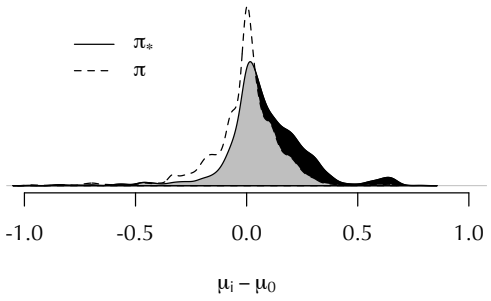


(a) uncentered scores

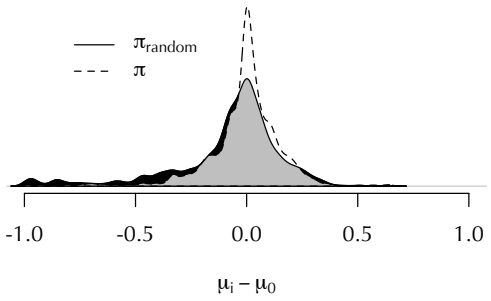


(b) centered scores

Figure 4: Distribution of NDCG@30 values for queries visited by the oracle policy for 28 training information needs. Note that the data for the first plot comes from the oracle policy while the data for the second plot comes from a pseudo-query reformulation policy.



(a) oracle



(b) random

Figure 5: Score distribution for different data-gathering policies. The shaded area reflect the distribution with respect to the exploration policy. The dashed line reflects the distribution with respect to an example solution. The black area reflects the over-representation by the exploration policy.

a good value for one information need may be terrible for another.

Consider the situation where we need to rank a set of reformulation candidates. The actual value of the metric is less important than the relative value. One way to address the poorly-calibrated values is to center all performance metric values by subtracting the value of the original query. The result, a distribution over the relative improvements over q_0, is presented in Figure 4b. This transform is reasonable for our task since it simplifies the regression problem to one of predicting a relative improvement over the baseline as opposed to wasting modeling effort on predicting the absolute performance metric value. In addition, if the model is accurate, it could provide a convenient method for pruning large areas of the search space predicted to be inferior to q_0.

Inspecting Figure 4b, though, also suggests why a regression against relative performance which minimizes the mean squared error may be undesirable. The distribution is very peaked around the center and a model will be penalized for poor predictions of reformulation candidates with little or no impact on performance. In the worst case, the model will predict values close to zero for all reformulation candidates.

Although binning or other techniques can be used to address this situation, we can address this unbalance by simplifying our problem further. Recall that we really only need a relative ordering of reformulation candidates. Therefore, we treat this as an ordinal regression problem. That is, we estimate a model which learns the correct ordering of reformulation candidates for a given information need. In practice, we train this model using true performance values of candidates encountered throughout a search process started at q_0; running this process over a number of training q_0's results in a large set of training candidates. Precisely how this training set is collected will be described in the next section.

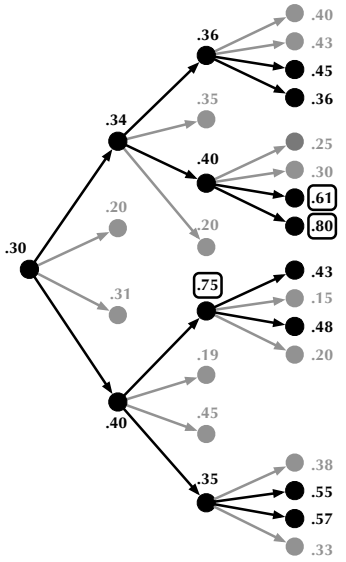
Even though we are interested in finding high-performing queries, we will not be biasing our pairwise loss toward the top of the ranked list of candidate queries. This is because our search algorithm is iterative and observes batches of reformulation candidates at a time, perhaps including highly performing queries, but often not. We need a model which is accurate for all reformulation candidates, not just the top performing ones. We are agnostic about the precise functional form of our model and opt for a linear ranking support vector machine [28] due to its training and evaluation speed, something we found necessary when conducting experiments at scale.

5.3 Searching Candidates

Considering the reformulation graph in Figure 1, the previous two sections explained how to represent the edges (candidate generation) and predict the value of nodes (candidate scoring). We still need to describe a process for searching for queries starting from q_0. We approach this process as a heuristic search problem, using the predicted performance as our heuristic. Unfortunately, algorithms such as A* cannot be reliably used because our heuristic is not admissible. Similarly, the noise involved in our performance prediction causes greedy algorithms such as beam search or best first search to suffer from local maxima.

```
QuerySearch(q, d, b, d_max, m)
    q          ▷ current query
    d          ▷ current depth
    b          ▷ search breadth
    d_max      ▷ maximum depth
    m          ▷ number of return reformulations
 1  if d = d_max
 2     then
 3          return q
 4  Q^q ← GenerateCandidates(q)
 5  μ̃ ← PredictPerformance(Q^q)
 6  Q̃^q ← TopQueries(Q^q, μ̃, b)
 7  Q̂^q ← TopQueries(Q^q, μ̃, m)
 8  for q_i ∈ Q̃^q
 9        do
10          Q̂^q ← Q̂^q ∪ QuerySearch(q_i, d + 1, b, d_max, m)
11  μ̂ ← PredictPerformance(Q̂^q)
12  return TopQueries(Q̂^q, μ̂, m)
```

(a) Query reformulation procedure.



(b) Illustration of the search process.

Figure 6: The search procedure recursively explores the reformulation graph and returns the top m highest scoring reformulations inspected. In the illustration, numbers reflect a query's predicted score. The bold nodes represent those nodes selected for expansion. The highlighted numbers represent the top m candidates visited throughout the search.

Motivated by our search simulator (Figure 2), we propose an algorithm that recursively inspects n reformulation candidates at each q_i up to a certain depth, $d_{\max}$. We present this algorithm in Figure 6a. The algorithm differs from our simulation insofar as it executes several reformulation sessions simultaneously, keeping track of those reformulations with the highest predicted effectiveness. One attractive aspect of our algorithm is the broad coverage of the reformulation space unlikely to be visited in greedier algorithms.

At termination, the algorithm selects a small number (m) of candidate queries visited for final retrieval. These m retrievals are merged using a Borda count algorithm with constituent rankings weighted by predicted performance. This process allows the algorithm to be more robust to errors in performance prediction.

The total number of candidates evaluated (Line 4 of Figure 6a) throughout the search process is approximately,

$$|\mathcal{C}| \approx \left\lfloor \frac{b^{d_{\max}} - 1}{b - 1} \right\rfloor n \tag{2}$$

where the approximation error comes from varying initial query length.

6. TRAINING

The effectiveness of the search algorithm (Section 5.3) critically depends on the reliability of the performance predictor (Section 5.2.2). Conversely, the distribution of instances supplied to the performance predictor depends on the decisions made by the search algorithm. Therefore, in order to train the performance prediction model, we need to gather example instances by executing a search and visiting nodes. Note that, for practical reasons, we cannot possibly gather training signals and targets for *all* queries in our search space. Even if we could, this set would probably not be representative of the instances the performance prediction model would observe in practice. For the same reason, we cannot use an arbitrary search policy in order to gather a smaller sample of instances. To see why this is the case, consider gathering instances for every reformulation candidate inspected by the oracle algorithm described in Section 3. Even though there will be poorly performing queries in this set of examples, the distribution would over-represent effective queries because the oracle is guiding the search towards those reformulations. We demonstrate in Figure 5a where we plot the distribution of centered performance metric values of queries inspected by the oracle compared to a distribution of those inspected by a model used in our experiments. As expected, the oracle visits a larger number of effective queries on average compared to our example solution. A model trained on unrepresentative data may be less performant than one trained on data more representative of the queries it will encounter during testing. At the same time, although sampling with a random policy seems attractive, the distribution of queries inspected here will have the opposite problem. As shown in Figure 5b, these queries are will be overrepresent less effective than those visited by the example solution.

The solution is to make gather a set of training instances for the performance prediction model which are representative of those visited by the search at test time. We accomplish this by gathering training instances using a data-gathering policy that approximates the behavior of our final graph search. The training operates as follows. We first partition our training queries into several subsets, $\{t_0, \ldots, t_5\}$; we also partition our validation queries into two subsets $\{v_0, v_1\}$; *our testing queries are left aside for evaluation* (Figure 7). We then iterate through the training subsets in order. For each subset, t_i, we execute the search algorithm

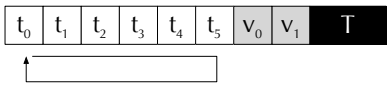


Figure 7: Partitioning of training (t_i), validation (v_i), and testing data (T).

in Figure 6a using the existing performance prediction model (or the oracle policy if $i = 0$). During the search, we record the feature vector and *true* performance of any encountered query. This set of $|\mathcal{C}| \times |t_i|$ instances from t_i, can then be used to train a performance prediction model. The regularization parameter of the SVM is tuned to select the model with the best performance on the validation set, v_0. After this step, we move on to the next training subset, t_{i+1}, using newly trained performance prediction model. As a result of this process, we iteratively accumulate a large set of training instance for the performance predictor representative of instances encountered during the search. Throughout the process we monitor performance on our second validation partition v_1. This method of gathering training representative training data has previously been used in robotics [38, Algorithm 3.1] and natural language processing [20, Algorithm 2].

We found that making several passes over the training splits improved the model performance on v_1. Therefore, we made several passes over the training splits and selected the model which performed best on v_1 for final evaluation. However, reformulating exactly the same queries in t_i may result in overfitting. To address this, after the first pass over t_i, we deformed the queries using the following procedure. With equal probability, terms were randomly added or dropped from the original query. The source of added terms was the true relevance model for the training query. We applied these perturbations until the Jaccard correlation between the top ten results of the perturbed and unperturbed queries was less than 0.50 and while performance was no less than 75% of the performance of the unperturbed query. These conditions ensured that the query was different (in terms of results) but still comparably performant with the unperturbed query. Similar perturbation processes have been used for computing query-dependent term similarity [8] and expanding digit recognition data [30].

7. METHODS

7.1 Data

We use three standard retrieval corpora for our experiments (Table 2). Two news corpora, trec12 and robust, consist of large archives of news articles. The trec12 dataset consists of the Tipster disks 1 and 2 with TREC *ad hoc* topics 51-200. The robust dataset consists of Tipster disks 4 and 5 with TREC *ad hoc* topics 301-450 and 601-700. Our web corpus consists of the Category B section of the Clue Web 2009 dataset with TREC Web topics 1-200. We tokenized all corpora on whitespace and then applied Krovetz stemming and removed words in the SMART stopword list.[1] We further pruned the web corpus of all documents with a Waterloo spam score less than 70.[2] We use TREC title queries

[1] `ftp://ftp.cs.cornell.edu/pub/smart/english.stop`
[2] `https://plg.uwaterloo.ca/~gvcormac/clueweb09spam/`

	documents	queries
trec12	469,949	51-200
robust	528,155	301-450,601-700
web	29,038,227	1-200

Table 2: Experiment corpora and query sets. Documents marked as spam removed from web before indexing.

in all of our experiments.

We randomly partitioned the queries into three sets: 60% for training, 20% for validation, and 20% for testing. We repeated this random split procedure five times and present results averaged across the test set queries.

7.2 Implementation

All indexing and retrieval was conducted using indri 5.7.[3] Our SVM models were trained using liblinear 1.95.[4] We evaluated final retrievals using NIST trec_eval 9.0.[5] In order to support large parameter sweeps, each query reformulation in PQR performed a *re-ranking* of the documents retrieved by q_0 instead of a *re-retrieval* from the full index. Pilot experiments found that the effectiveness of re-retrieval was comparable with that of re-ranking though re-retrieval incurred *much* higher latency.

7.3 Parameters

Aside from the performance prediction model, our algorithm has the following free parameters: the number of term-addition candidates per query (n), the number of candidates to selection per query (b), and the maximum search depth ($d_{\max}$). Combined, the automatic reformulation and the multi-pass training resulted in computationally expensive processes whose runtime is sensitive to these parameters. Consequently, we fixed our parameter settings at relatively modest numbers ($n = 10, b = 3, d_{\max} = 4$) and leave a more thorough analysis of sensitivity for an extended manuscript. Although these numbers may seem small, we remind the reader that this results in roughly $|\mathcal{C}| \approx 800$ reformulations considered within the graph search for a single q_0 (Equation 2). The number of candidates to merge (m) is tuned throughout training on the validation set v_0 and ranges from five to twenty.

The query likelihood baseline used Dirichlet smoothing with parameter tuned on the full training set using a range of values from 500 through 5000. The parameters of the relevance model baseline (RM3) were also tuned on the full training set. The range of feedback terms considered was $\{5, 10, 25, 50, 75, 100\}$; the range of feedback documents was $\{5, 25, 50, 75, 100\}$; the range of λ was $[0, 1]$ with a step size of 0.1.

All runs, including baselines, optimized NDCG@30.

8. RESULTS

We present the results for our experiments in Table 3. Our first baseline, query likelihood (QL) reflects the performance of q_0 alone and represents an algorithm which is representationally comparable with PQR insofar as it also

[3] `http://www.lemurproject.org/indri/`
[4] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large_scale_ranksvm`
[5] `http://trec.nist.gov/trec_eval/`

Table 3: Comparison of PQR to query likelihood (QL) and relevance model (RM3) baselines for our datasets. Statistically significant difference with respect to QL (■: better; □: worse) and RM3 (♦: better; ◊: worse) using a Student's paired t-test ($p < 0.05$ with a Bonferroni correction). The best performing run is presented in bold. All runs have parameters tuned for NDCG@30 on the validation set.

	NDCG@5	NDCG@10	NDCG@20	NDCG@30	NDCG	MAP
trec12						
QL	0.5442	0.5278	0.5066	0.4835	0.5024	0.2442
RM3	**0.6465**$^{■}$	**0.6113**$^{■}$	**0.5796**$^{■}$	**0.5627**$^{■}$	**0.5300**$^{■}$	**0.2983**$^{■}$
random	0.5690 $^{◊}$	0.5563 $^{◊}$	0.5257 $^{◊}$	0.5089 $^{◊}$	0.5120$^{■◊}$	0.2653$^{■◊}$
PQR	0.6112$^{■◊}$	0.5907$^{■}$	0.5630$^{■}$	0.5419$^{■◊}$	0.5216$^{■◊}$	0.2819$^{■◊}$
robust						
QL	0.4874	0.4559	0.4306	0.4172	0.5419	0.2535
RM3	0.4888	0.4553	0.4284	0.4176	0.5462	0.2726$^{■}$
random	0.4240$^{□◊}$	0.3967$^{□◊}$	0.3675$^{□◊}$	0.3588$^{□◊}$	0.5143$^{□◊}$	0.2352$^{□◊}$
PQR	**0.5009**	**0.4713**$^{■♦}$	**0.4438**$^{■♦}$	**0.4315**$^{■♦}$	**0.5498**$^{■}$	**0.2736**$^{■}$
web						
QL	0.2206	0.2250	0.2293	0.2315	0.3261	0.1675
RM3	0.2263	0.2273	0.2274	0.2316	**0.3300**$^{■}$	**0.1736**$^{■}$
random	0.1559$^{□◊}$	0.1562$^{□◊}$	0.1549$^{□◊}$	0.1537$^{□◊}$	0.2790$^{□◊}$	0.1157$^{□◊}$
PQR	**0.2528**$^{■♦}$	**0.2501**$^{■♦}$	**0.2493**$^{■♦}$	**0.2435**$^{■}$	**0.3300**	0.1690

retrieves using a short, unweighted query. Our second baseline, the relevance model (RM3) reflects the performance of a strong algorithm that also uses the retrieval results to improve performance, although with much richer representational power (the optimal number of terms often hover near 75-100). As expected, RM3 consistently outperforms QL in terms of MAP. And while the performance is superior across all metrics for trec12, RM3 is statistically indistinguishable from QL for higher precision metrics on our other two data sets. The random policy, which replaces our performance predictor with random scores, consistently underperforms both baselines for robust and web. Interestingly, this algorithm is statistically indistinguishable from QL for trec12, suggesting that this corpus may be easier than others.

Next, we turn to the performance of PQR. Across all corpora and across almost all metrics, PQR significantly outperforms QL. While this baseline might be considered low, it is a representationally fair comparison with PQR. So, this result demonstrates the ability of PQR to find more effective reformulations than q_0. The underperformance of the random algorithm signifies that the effectiveness of PQR is attributable to the performance prediction model as opposed to a merely walking on the reformulation graph. That said, PQR is statistically indistinguishable from QL for higher recall metrics on the web corpus (NDCG and MAP). In all likelihood, this results from the optimization of NDCG@30, as opposed to higher recall metrics. This outcome is amplified when we compare PQR to RM3. For the robust and web datasets, we notice PQR significantly outperforming RM3 for high precision metrics but showing weaker performance for high recall metrics. We point out that PQR performs weaker than RM3 for trec12. This might be explained by the easier nature of the corpus combined with the richer representation of the RM3 model.

We can inspect the coefficient values to determine the importance of individual signals in performance prediction. In Table 4, we present the most important signals for each of our experiments. Because our results are averaged over several runs, we selected the signals most often occurring amongst the highest weighted in these runs, using the final selected model (see Section 6). Interestingly, many of the top ranked signals are our drift features which compare the language models and rankings of the candidate result set with those of its parent and the first query. This suggests that the algorithm is successfully preventing query drift by promoting candidates that retrieve results similar to the original and parent queries. On the other hand, the high weight for Clarity suggests that PQR is simultaneously balancing ranked list refinement with ranked list anchoring.

Table 4: Top five highest weighted signals for each experiment. For each run in each experiment, we ranked all signals by the magnitude of their associated weight in the linear model. We aggregated these rankings and present the signals ranked by frequency in the top five signals across runs.

trec12	robust	web
$\mathcal{B}(\theta_{\mathcal{R}_0}, \theta_{\mathcal{R}_t})$	$\mathcal{B}(\theta_{\mathcal{R}_0}, \theta_{\mathcal{R}_t})$	$\tau_{\text{AP}}(\mathcal{R}_0, \mathcal{R}_t)$
$\mathcal{B}(\theta_{\mathcal{R}_{t-1}}, \theta_{\mathcal{R}_t})$	$\mathcal{B}(\theta_{\mathcal{R}_{t-1}}, \theta_{\mathcal{R}_t})$	$\mathcal{B}(\theta_{\mathcal{R}_0}, \theta_{\mathcal{R}_t})$
$\tau_{\text{AP}}(\mathcal{R}_0, \mathcal{R}_t)$	Clarity	$\tau_{\text{AP}}(\mathcal{R}_{t-1}, \mathcal{R}_t)$
$\tau_{\text{AP}}(\mathcal{R}_{t-1}, \mathcal{R}_t)$	$\tau_{\text{AP}}(\mathcal{R}_{t-1}, \mathcal{R}_t)$	$\mathcal{B}(\theta_{\mathcal{R}_{t-1}}, \theta_{\mathcal{R}_t})$
Clarity	maxIDF	Clarity

9. DISCUSSION

Although QL is the appropriate baseline for PQR, comparing PQR performance to that of RM3 helps us understand where improvements may be originating. The effectiveness of RM3 on trec12 is extremely strong, demonstrat-

ing statistically superior performance to PQR on many metrics. At the same time, the absolute metrics for QL on these runs is also higher than on the other two collections. This suggests that part of the effectiveness of RM3 results from the strong initial retrieval (i.e. QL). As mentioned earlier, the strength of the random run separately provides evidence of the initial retrieval's strength. Now, if the initial retrieval uncovered significantly more relevant documents, then RM3 will estimate a language model very close to the true relevance model, boosting performance. Since RM3 allows a long, rich, weighted query, it follows that it would outperform PQR's constrained representation. That said, it is remarkable that PQR achieves comparable performance to RM3 on many metrics with at most $|q_0| + d_{\max}$ words.

The weaker performance for high-recall metrics was somewhat disappointing but should be expected given our optimization target (NDCG@30). Post-hoc experiments demonstrated that optimizing for MAP boosted the performance of PQR to 0.1728 on web, resulting in statistically indistinguishable performance with RM3. Nevertheless, we are not certain that human query reformulation of the type encountered in general web search would improve high recall metrics since users in that context rarely inspect deep into the ranked list.

One of the biggest concerns with PQR is efficiency. Whereas our QL baseline ran in a 100-200 milliseconds, PQR ran in 10-20 *seconds*, even using the re-ranking approach. However, because of this approach, our post-retrieval costs scale modestly as corpus size grows, especially compared to massive query expansion techniques like RM3. To understand this observation, note that issuing a long RM3 query results in a huge slowdown in performance due to the number of postings lists that need to be evaluated and merged. We found that for the web collection, RM3 performed quite slow, often taking *minutes* to complete long queries. PQR, on the other hand, has the same overhead as RM3 in terms of an initial retrieval and fetching document vectors. After this step, though, PQR only needs to access the index for term statistic information, not a re-retrieval. Though even with our speedup, PQR is unlikely to be helpful for realtime, low-latency retrieval. However, there are several situations where such a technique may be permissible. For example, 'slow search' refers to search situations where users tolerate latency in order to receive better results [42]. Another situation is document filtering, where the user has a standing query for a certain topic and the system can optimize its query representation during indexing lulls. More generally, this technique is also valuable for any distributed information retrieval problem with APIs constrained to unweighted queries.

10. CONCLUSION

The positive results on three separate corpora provide evidence that PQR is a framework worth investigating further. In terms of candidate generation, we considered only very simple word additions and deletions while previous research has demonstrated the effectiveness of applying multiword units (e.g. ordered and unordered windows) [34]. Beyond this, we can imagine applying more sophisticated operations such as filters, site restrictions, or time ranges. While it would increase our query space, it may also allow for more precise and higher precision reformulations. In terms of candidate scoring, we found that our novel drift signals allowed for effective query expansion. We believe that PQR provides a framework for developing other performance predictors in a grounded retrieval model. In terms of graph search, we believe that other search strategies might result in more effective coverage of the space.

We would like to return to our original motivation: mimicking human reformulation. We have developed framework for learning reformulation behavior from an oracle. In many situations, as with production web search engines, we have access to human reformulation behavior. Given such data, we could train a PQR model directly on human behavior. Although prior work demonstrates the poor performance of human reformulation, we are interested in exploring the effects on our trained models.

11. REFERENCES

[1] R. Attar and A. S. Fraenkel. Local feedback in full-text retrieval systems. *J. ACM*, 24(3), 1977.

[2] M. Bendersky. *Information Retrieval with Query Hypergraphs*. PhD thesis, University of Massachusetts Amherst, 2012.

[3] M. Bendersky and W. B. Croft. Discovering key concepts in verbose queries. In *Proceedings of the 31st SIGIR Conference*, 2008.

[4] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by probability distributions. *Bull. Calcutta Math. Soc.*, 35, 1943.

[5] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: Model and applications. In *Proceedings of the 17th CIKM*, 2008.

[6] P. Bruza and S. Dennis. Query reformulation on the internet: Empirical data and the hyperindex search engine. In *Recherche d'Information et ses Applications*, 1997.

[7] G. Cao, J.-Y. Nie, J. Gao, and S. Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st SIGIR Conference*, 2008.

[8] K. Collins-Thompson. Robust word similarity estimation using perturbation kernels. In L. Azzopardi, G. Kazai, S. Robertson, S. Rüger, M. Shokouhi, D. Song, and E. Yilmaz, editors, *Advances in Information Retrieval Theory*, volume 5766. 2009.

[9] W. B. Croft and D. J. Harper. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35(4), 1979.

[10] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of the 25th SIGIR Conference*, 2002.

[11] F. Diaz. Performance prediction using spatial autocorrelation. In *Proceedings of the 30th SIGIR Conference*, 2007.

[12] F. Diaz and J. Arguello. Adaptation of offline vertical selection predictions in the presence of user feedback. In *Proceedings of the 32nd SIGIR Conference*, 2009.

[13] F. Diaz and R. Jones. Using temporal profiles of queries for precision prediction. In *Proceedings of the 27th SIGIR Conference*, 2004.

[14] H. A. Feild, J. Allan, and R. Jones. Predicting searcher frustration. In *Proceedings of the 33rd SIGIR Conference*, 2010.

[15] D. Harman. Towards interactive query expansion. In *Proceedings of the 11th SIGIR Conference*, 1988.

[16] C. Hauff. *Predicting the Effectiveness of Queries and Retrieval Systems*. PhD thesis, University of Twente, 2010.

[17] C. Hauff, L. Azzopardi, and D. Hiemstra. The combination and evaluation of query performance prediction methods. In *Proceedings of ECIR*, 2009.

[18] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. In *Proceeding of CIKM*, 2008.

[19] B. He and I. Ounis. Inferring Query Performance Using Pre-retrieval Predictors. In *The Eleventh Symposium on String Processing and Information Retrieval (SPIRE)*, 2004.

[20] H. He, H. Daumé III, and J. Eisner. Dynamic feature selection for dependency parsing. In *Proceedings of EMNLP*, 2013.

[21] C.-K. Huang, L.-F. Chien, and Y.-J. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *J. Am. Soc. Inf. Sci. Technol.*, 54, 2003.

[22] J. Huang and E. N. Efthimiadis. Analyzing and evaluating query reformulation strategies in web search logs. In *Proceedings of the 18th CIKM*, 2009.

[23] E. Kharitonov, C. Macdonald, P. Serdyukov, and I. Ounis. User model-based metrics for offline query suggestion evaluation. In *Proceedings of the 36th SIGIR Conference*, 2013.

[24] O. Kurland, L. Lee, and C. Domshlak. Better than the real thing?: iterative pseudo-query processing using cluster-based language models. In *Proceedings of the 28th SIGIR Conference*, 2005.

[25] T. Lau and E. Horvitz. Patterns of search: Analyzing and modeling web query refinement. In *Proceedings of the Seventh International Conference on User Modeling*, 1999.

[26] V. Lavrenko and W. B. Croft. Relevance based language models. In *Proceedings of the 24th SIGIR Conference*, 2001.

[27] M. Lease. An improved markov random field model for supporting verbose queries. In *Proceedings of the 32nd SIGIR Conference*, 2009.

[28] C.-P. Lee and C.-J. Lin. Large-scale linear ranksvm. *Neural Computation*, 26(4), 2014.

[29] T.-Y. Liu. *Learning to Rank for Information Retrieval*. 2009.

[30] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. 2007.

[31] Y. Lv and C. Zhai. Adaptive relevance feedback in information retrieval. In *Proceedings of the 18th CIKM*, 2009.

[32] C. Macdonald, R. L. Santos, and I. Ounis. On the usefulness of query features for learning to rank. In *Proceedings of the 21st CIKM*, 2012.

[33] M. Magennis and C. J. van Rijsbergen. The potential and actual effectiveness of interactive query expansion. In *Proceedings of the 20th SIGIR Conference*, 1997.

[34] D. Metzler and W. B. Croft. Latent concept expansion using markov random fields. In *Proceedings of the 30th SIGIR Conference*, 2007.

[35] C. N. Mooers. A mathematical theory of language symbols in retrieval. In *Proceedings of the International Conference on Scientific Information*, 1959.

[36] U. Ozertem, O. Chapelle, P. Donmez, and E. Velipasaoglu. Learning to suggest: a machine learning framework for ranking query suggestions. In *Proceedings of the 35th SIGIR Conference*, 2012.

[37] F. Radlinski and N. Craswell. Comparing the sensitivity of information retrieval metrics. In *Proceedings of the 33rd SIGIR Conference*, 2010.

[38] S. Ross, G. Gordon, and J. A. D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of AISTATS*, 2011.

[39] I. Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *Proceedings of the 26th SIGIR Conference*, 2003.

[40] D. Sheldon, M. Shokouhi, M. Szummer, and N. Craswell. Lambdamerge: Merging the results of query reformulations. In *Proceedings of the Fourth WSDM Conference*, 2011.

[41] C. Silverstein, M. Henzinger, J. Marais, and M. Moricz. Analysis of a very large altavista query log. Technical Report SRC-TN-1998-014, HP Labs Technical Report, 1998.

[42] J. Teevan, K. Collins-Thompson, R. W. White, and S. Dumais. Slow search. *Commun. ACM*, 57(8), 2014.

[43] X. Xue and W. B. Croft. Generating reformulation trees for complex queries. In *Proceedings of the 35th SIGIR Conference*, 2012.

[44] X. Xue, W. B. Croft, and D. A. Smith. Modeling reformulation using passage analysis. In *Proceedings of the 19th CIKM*, 2010.

[45] E. Yilmaz, J. A. Aslam, and S. Robertson. A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st SIGIR Conference*, 2008.

[46] L. Zhao and J. Callan. Term necessity prediction. In *Proceedings of the 19th CIKM*, 2010.