Pseudo-Query Reformulation

Fernando Diaz(✉)

Microsoft Research, New York, USA
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 [11] or semi-automatic reformulations assisted by the system [10]. 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.

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 [15], 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 [4]. 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

N. Ferro et al. (Eds.): ECIR 2016, LNCS 9626, pp. 521–532, 2016.
DOI: 10.1007/978-3-319-30671-1_38

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.

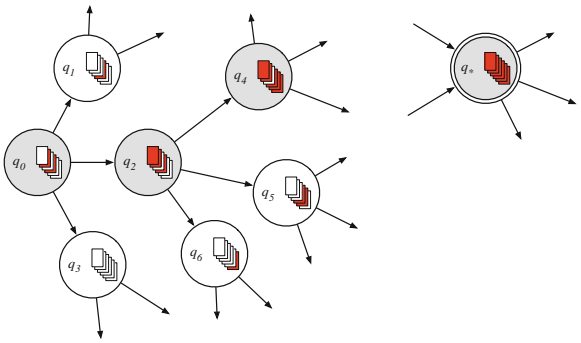


Fig. 1. Query reformulation as graph search. Nodes represent queries and associated retrieved results. Relevant documents are highlighted in red. Edges exist between nodes whose queries are simple reformulations of each other. The goal of pseudo-query reformulation is to, given a seed query q_0 by a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

PQR is a fundamentally new query reformulation model with several attractive properties. 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

Kurland *et al.* present several heuristics for iteratively refining a language model query by navigating document clusters in a retrieval system [13]. 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 [19]. 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 [17]. 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 [16]. Similarly Xue and Croft have demonstrated how to use performance predictors in order to improve concept weighting in an inference network model [21]. 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 [1,3].

The work of Kumaran and Carvahlo examines using performance predictors for *query reduction*, the task of selecting the best reformulation from the set of all $\mathcal{O}(2^n)$ possible subqueries of the original query [12]. Because of the size of this set, the authors use several heuristics to prune the search space. While related, our approach is substantially more scalable (in terms of number of reformulations considered) and, as a result, flexible (in terms of query transformations).

3 Motivation

As mentioned earlier, users often reformulate an initial query in response to the system's ranking [11]. 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 [11, Table 5]. However, previous research has demonstrated that humans are not as effective as automatic methods in this task [18].

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 models an omniscient user based on query-document relevance judgments, known as qrels. Given a seed query, the user first generates a set of candidate reformulations based on one-word additions and deletions. The user then selects the query whose retrieval has the highest performance based on the qrels. The process then iterates up to some search depth. If we consider queries

as nodes in a graph and edges between queries with single word additions or deletions, then the process can be considered a depth-limited graph search by an oracle.

We ran this simulation on a sample of queries described in Sect. 6. 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.

Table 1. NDCG@30 for random and optimal (PQR*) pseudo-query reformulation compared to query likelihood (QL) and relevance model (RM3).

	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

4 Pseudo-Query Reformulation

We would like to develop algorithms that can approximate the behavior of our optimal policy *without having access to any qrels or an oracle*. As such, PQR follows the framework of the simulator from Sect. 3. The algorithm recursively performs candidate generation and candidate scoring within each recursion up to some depth and returns the highest scored query encountered.

4.1 Generating Candidates

Recall that 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. 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 [14]. 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.

4.2 Scoring Candidates

The candidate generation process described in Sect. 4.1 provides a crude method for pruning the search space. Based on our observations with the random and oracle policies in Table 1, we know that inaccurately scoring reformulation candidates can significantly degrade the performance of an algorithm. In this section, we model the oracle by using established performance prediction signals.

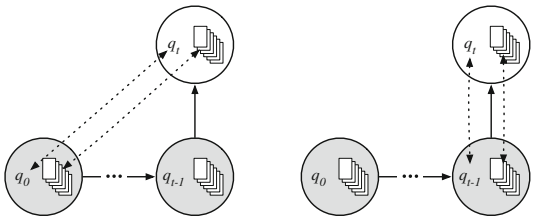
Performance Prediction Signals. Performance prediction refers to the task of ordering a set of queries without relevance information so that the better performing queries are ordered above worse performing queries. With some exception, the majority of work in this area has focused on ranking queries 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 Sect. 4.2, 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. [7]).

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 [7]. 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) [9].

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 [5]. 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 [2]. 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. 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 [6]. 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.



(a) Initial Query (b) Parent Query

Fig. 2. Drift signal classes. Signals for q_t include comparisons with reference queries q_{t-1} and q_0 to prevent query drift.

Drift signals compare the current query q_t with its parent q_{t-1} and the initial query q_0 (Fig. 2). 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. 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 [22]. 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.

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 absolute performance for a set of training queries [8]. Instead, 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. We are agnostic about the precise functional form of our model and opt for a linear ranking support vector machine [15] due to its training and evaluation speed, something we found necessary when conducting experiments at scale. Precisely how this training set is collected will be described in the next section.

4.3 Searching Candidates

Considering reformulation as a graph, 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.

Motivated by our search simulator (Sect. 3), 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 Fig. 3. 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 scoring.

```
QuerySearch(q, d, b, d_max, m)
 1  if d = d_max
 2     then return q
 3  Q^q ← GenerateCandidates(q)
 4  μ̃ ← PredictPerformance(Q^q)
 5  Q̃^q ← TopQueries(Q^q, μ̃, b)
 6  Q̂^q ← TopQueries(Q^q, μ̃, m)
 7  for q_i ∈ Q̃^q
 8        do Q̂^q ← Q̂^q ∪ QuerySearch(q_i, d + 1, b, d_max, m)
 9  μ̂ ← PredictPerformance(Q̂^q)
10  return TopQueries(Q̂^q, μ̂, m)
```

Fig. 3. Query reformulation procedure. The search procedure recursively explores a reformulation graph and returns the top m highest scoring reformulations.

5 Training

The effectiveness of the search algorithm (Sect. 4.3) critically depends on the reliability of the performance predictor (Sect. 4.2). We can gather training data for this model by sampling batches of reformulations from the query graph. In order to ensure that our sample includes high quality reformulations, we run the oracle algorithm (Sect. 3) while recording the visited reformulations and their true performance values. Any bias in this data collection process can be addressed by gathering additional training reformulations with suboptimal performance predictor models (i.e. partially trained models).

6 Methods

We use three standard retrieval corpora for our experiments. 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. We further pruned the web corpus of all documents with a Waterloo spam score less than 70. We use TREC title queries 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.

All indexing and retrieval was conducted using indri 5.7. Our SVM models were trained using liblinear 1.95. We evaluated final retrievals using NIST trec_eval 9.0. 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. 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 800 reformulations considered within the graph search for a single q_0. The number of candidates to merge (m) is tuned throughout training on the validation set v_0 and ranges from five to twenty. All runs, including baselines, optimized NDCG@30. We used QL and RM3 as baselines. QL used Dirichlet smoothing with parameter tuned on the full training set using a range of values from 500

through 5000. We tuned the RM3 parameters 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.

7 Results

We present the results for our experiments in Table 2. Our first baseline, QL, reflects the performance of q_0 alone and represents an algorithm which is representationally comparable with PQR insofar as it also retrieves using a short, unweighted query. Our second baseline, 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.

Table 2. Comparison of PQR to QL and 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.

	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

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. The weaker performance of PQR compared to RM3 for trec12 might be explained by the easier nature of the corpus combined with the richer representation of the RM3 model. Nevertheless, we stress that, for those metrics we optimized for and for the more precision-oriented collections, PQR dominates all baselines.

We can inspect the coefficient values in the linear model to determine the importance of individual signals in performance prediction. In Table 3, 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 Sect. 5). 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 3. Top five highest weighted signals for each experiment.

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

8 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, demonstrating 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.

Despite the strong performance for high-precision metrics, 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 ms, PQR ran in 10–20 s, 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, including 'slow search' scenarios where users tolerate latency in order to receive better results [20], offline result caching, and document filtering.

9 Conclusion

We have presented a new formal model for information retrieval. 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). 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.

References

1. Bendersky, M.: Information Retrieval with Query Hypergraphs. Ph.D. thesis, University of Massachusetts Amherst (2012)
2. Bhattacharyya, A.: On a measure of divergence between two statistical populations defined by probability distributions. Bull. Calcutta Math. Soc. **35**, 99–109 (1943)
3. Cao, G., Nie, J.Y., Gao, J., Robertson, S.: Selecting good expansion terms for pseudo-relevance feedback. In: SIGIR (2008)
4. Croft, W.B., Harper, D.J.: Using probabilistic models of document retrieval without relevance information. J. Documentation **35**(4), 285–295 (1979)
5. Cronen-Townsend, S., Zhou, Y., Croft, W.B.: Predicting query performance. In: SIGIR (2002)
6. Diaz, F.: Performance prediction using spatial autocorrelation. In: SIGIR (2007)
7. Hauff, C.: Predicting the Effectiveness of Queries and Retrieval Systems. Ph.D. thesis, University of Twente (2010)
8. Hauff, C., Azzopardi, L., Hiemstra, D.: The combination and evaluation of query performance prediction methods. In: Boughanem, M., Berrut, C., Mothe, J., Soule-Dupuy, C. (eds.) ECIR 2009. LNCS, vol. 5478, pp. 301–312. Springer, Heidelberg (2009)
9. He, B., Ounis, I.: Inferring query performance using pre-retrieval predictors. In: SPIRE (2004)
10. Huang, C.K., Chien, L.F., Oyang, Y.J.: Relevant term suggestion in interactive web search based on contextual information in query session logs. JASIST **54**, 638–649 (2003)
11. Huang, J., Efthimiadis, E.N.: Analyzing and evaluating query reformulation strategies in web search logs. In: CIKM (2009)
12. Kumaran, G., Carvalho, V.: Reducing long queries using query quality predictors. In: SIGIR (2009)
13. Kurland, O., Lee, L., Domshlak, C.: Better than the real thing?: iterative pseudo-query processing using cluster-based language models. In: SIGIR (2005)
14. Lavrenko, V., Croft, W.B.: Relevance based language models. In: SIGIR (2001)
15. Liu, T.Y.: Learning to Rank for Information Retrieval. Springer, Heidelberg (2009)
16. Lv, Y., Zhai, C.: Adaptive relevance feedback in information retrieval. In: CIKM (2009)
17. Macdonald, C., Santos, R.L., Ounis, I.: On the usefulness of query features for learning to rank. In: CIKM (2012)
18. Ruthven, I.: Re-examining the potential effectiveness of interactive query expansion. In: SIGIR (2003)
19. Sheldon, D., Shokouhi, M., Szummer, M., Craswell, N.: Lambdamerge: merging the results of query reformulations. In: WSDM (2011)
20. Teevan, J., Collins-Thompson, K., White, R.W., Dumais, S.: Slow search. Commun. ACM **57**(8), 36–38 (2014)
21. Xue, X., Croft, W.B., Smith, D.A.: Modeling reformulation using passage analysis. In: CIKM (2010)
22. Yilmaz, E., Aslam, J.A., Robertson, S.: A new rank correlation coefficient for information retrieval. In: SIGIR (2008)