LambdaMerge: Merging the Results of Query Reformulations

Daniel Sheldon*
Oregon State University
sheldon@eecs.oregonstate.edu

Milad Shokouhi, Martin Szummer and Nick Craswell
Microsoft
{milads, szummer, nickcr}@microsoft.com

ABSTRACT

Search engines can automatically reformulate user queries in a variety of ways, often leading to multiple queries that are candidates to replace the original. However, selecting a replacement can be risky: a reformulation may be more effective than the original or significantly worse, depending on the nature of the query, the source of reformulation candidates, and the corpus. In this paper, we explore methods to mitigate this risk by issuing several versions of the query (including the original) and merging their results. We focus on reformulations generated by random walks on the click graph, a method that can produce very good reformulations but is also variable and prone to topic drift. Our primary contribution is LambdaMerge (λ-Merge), a supervised merging method that is trained to directly optimize a retrieval metric (such as NDCG or MAP) using features that describe both the reformulations and the documents they return. In experiments on Bing data and GOV2, λ-Merge outperforms the original query and several unsupervised merging methods. λ-Merge also outperforms a supervised method to predict and select the best single formulation, and is competitive with an oracle that *always* selects the best formulation.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Query formulation*

General Terms

Experimentation, Measurement

Keywords

LambdaMerge, Query reformulation, Query expansion, Learning to rank

*Work performed while at MSR Cambridge.

WSDM'11, February 9–12, 2011, Hong Kong, China.

1. INTRODUCTION

Consider the user query 'jupiters mass'. A search engine has a variety of ways to generate possible reformulations of this query, for example, by using information from past user sessions [16, 17], co-click analysis [6, 32] or anchor text analysis [9]. In this case, a co-click random walk suggests the reformulations 'jupiter mass', 'mass of jupiter' and 'jupiter facts'.

Slightly different query formulations may yield very different top-ranked results when issued to a search engine, and these results may vary significantly in quality and relevance to the user's information need. For example, the query 'jupiters mass' is missing an apostrophe, so it may yield poor results depending on the retrieval algorithms employed. The query 'mass of jupiter' is a good reformulation in this case, because it uses a more common phrasing and is more likely to exactly match the text in an expository page that contains the desired information. However, results depend in a complex way on the algorithms, corpus and index used for retrieval, so it is extremely challenging to know ahead of time which formulation will yield the best results.

Topic drift is a significant risk when attempting to choose a query reformulation. The query 'jupiter facts' has a different (broader) meaning than the original query that pertained specifically to the mass of Jupiter. Even if the search engine returns very good results for 'jupiter facts', the user may not be satisfied by the results. It is important for a reformulation method to avoid topic drift whenever possible.

This discussion highlights the fact that it can be difficult even for a human to judge the quality of formulations and to diagnose topic drift. However, in the context of a particular retrieval system one may evaluate the quality of a reformulation by retrieval performance: issue the query, judge the relevance of the results with respect to the *original* query, and then measure the quality of the results list using an evaluation measure such as NDCG [15] or MAP. Reformulations with high evaluation scores can be considered good reformulations.

Using Reformulations. Once multiple reformulations have been generated, the conventional approach would be to select one, issue that query formulation to the search engine, and return the results. The previous discussion highlights the risks inherent in this approach: issuing only the original query forgoes the relevance improvements from reformulations such as 'mass of jupiter'. However, if one chooses to abandon the user's original query this may degrade rele-

vance in some cases (e.g., topic drift from reformulation like 'jupiter facts'). Bad reformulations may give results that seem inexplicable to the user.

This paper presents methods to mitigate the aforementioned risks by issuing multiple formulations of the query to the search engine and merging the top-k results of each to produce a final ranked list. There are a number of advantages to this approach. First, properties of the results lists convey important information about the quality of a reformulation that is not available prior to retrieval, such as score distribution (to predict retrieval performance) and overlap with the other results lists (to diagnose drift). This information can guide the merging process. Second, results are always obtained for the original query, so it is possible to fall back to those if the reformulations are detected to be bad. Finally, merging is a strictly more powerful method for combining the results from multiple queries than selecting of a single query, so it is theoretically capable of greater performance improvements. For example, consider a set of five query formulations that each returns a single distinct relevant document: a perfect single-query system can only return one relevant document, while a merging system can potentially return all five.

Our main contribution is a supervised merging approach called λ-Merge that learns a scoring function to rank documents from multiple reformulations by combining features that indicate document quality (such as retrieval score) with features that indicate the quality of the reformulation and its results (such as score distribution or overlap with other results lists). We demonstrate that this approach outperforms a number of baselines and in some cases performs as well as a selection oracle that chooses between the original query and its reformulations. The main advantages of λ-Merge are that: (1) it is trained to directly optimize a retrieval metric such as NDCG or MAP, (2) the flexible architecture of the scoring function can use both query-document and results-list based performance prediction features, and (3) λ-Merge can handle reformulations that are worse than the original query and still improve performance significantly.

Our experiments utilize reformulations from two-step random walks on the click graph [6]. These reformulations vary greatly in quality. Although some are significantly better than the original query, they are *worse* than the original query on average. Because of this, simple unsupervised merging baselines that treat all reformulations equally — such as the well-known CombSUM algorithm [23] — degrade performance. However, we devise a simple unsupervised modification to CombSUM called CombRW that is able to outperform the original query (in NDCG@5) by using the random walk probabilities of the reformulations as merging weights. This suggests that merging can be successful, but it is important to take into account some measure of reformulation quality, and it motivates the architecture of λ-Merge presented in Section 3.2.

We also compare λ-Merge to a supervised baseline [2] that uses linear regression to predict the quality of each reformulation, and then selects the *single* formulation with the highest predicted performance. We show that λ-Merge significantly outperforms both CombRW and this supervised approach, and is even competitive with an *oracle* selection method that *always* chooses the single best formulation. These observations demonstrate that supervision alone is not enough in our setting, and that merging has significant advantages over a system that selects a single query.

In the remaining sections, we demonstrate how blending the results of different query reformulations can improve retrieval effectiveness. In particular, we discuss the limitations of current baselines and show that they are consistently outperformed by our supervised merging technique (λ-Merge).

2. RELATED WORK

Our work draws on multiple areas of information retrieval (IR) research, including query reformulation, learning to rank, result merging, and performance prediction. This section briefly introduces each of these areas and discusses relevant literature from each.

Query reformulation. Reformulation methods modify the user's query to boost the quality of search results. Traditional query reformulation methods in IR are based on pseudo-relevance (blind) feedback, which expands a given query by first retrieving its top-k documents, and then extracting document terms to add to the query [20, 26]. However, query reformulation is not limited to expansion. Studies show that performance improvement can be achieved by dropping [16] or substituting terms based on previous queries [9, 17, 29]. It is also possible to perform multiple refinements to a query (e.g. expansion and spell-correction) [13].

Candidate reformulations for a query can be collected from sources such as user session rewrites [16, 17], anchor-text [18, 9], word co-occurrence in documents [27, 30], click graphs [6, 32] or combination of multiple sources [13]. In this work, we follow the methodology of Craswell and Szummer [6] to generate reformulation candidates from click logs. The authors proposed a random walk model on the bipartite Query-URL click graph. This graph has a node for each query and URL that appears in a large usage log, with edges connecting a query to each URL that was clicked by some user in response to that query. For any query or URL node in the graph, the model outputs a probability distribution over other "nearby" nodes (including both queries and URLs). Craswell and Szummer focused on retrieval, and hence used the procedure to find a probability distribution over URLs given a query in order to rank the URLs. The resulting ranking has enhanced recall relative to a direct click-based ranking, because the results may include URLs that were not directly connected to the given query. Empirically the model was also shown to be robust to noise and to maintain good precision.

The same click-based random walk algorithm can also be used to find Query-Query, URL-Query and URL-URL relationships. Here we use a 2-step random walk to find Query-Query reformulations. For example, starting from the query 'jupiters mass', the first step of the walk gives a distribution over URLs that were clicked for that query. The second step moves the probability mass back to the queries. From [6] we choose a forward walk using their standard edge weighting based on click counts.

In general, the random walk assigns high probability to queries that share many clicked URLs with the original query, and hence these queries tend to be on the same topic as the original query, so that topic drift is avoided. The random walk may also assign high probability to formulations that have greater retrieval effectiveness, because these queries have higher click counts, which increases the probability that

the random walk steps to those queries under the model's edge weighting scheme. However, despite these advantages, the random walk distribution must always assign 100% of the probability mass to *some* set of queries, so in the case where there are no good reformulations within two steps it will still propose a reformulation. Thus, it is important for merging methods that use random walk reformulations to be robust to bad candidates.

Learning to rank. Traditional IR rankers such as BM25 [22] have few parameters, so that it is possible to tune them by hand. However, in a setting such as web search, rankers are based on a rich and diverse set of features that may include any desired feature of a query-document pair. When features are sufficiently numerous and noisy, it is practically impossible to find a good combination of features by hand.

Instead, learning-to-rank algorithms use a judged training set of query-document pairs and apply machine learning techniques to learn complex combinations of features. Recent algorithms, such as LambdaRank [4], directly optimize a desired IR metric (such as NDCG or MAP) on the training set, and aim to generalize well on novel test data.

The learning to merge problem is an instance of learning to rank, in that we employ a rich set of query and document features, and learn a ranking function from training data. However, merging has special structure beyond regular learning to rank, because it combines multiple lists of results. We add a *gating* component to LambdaRank [4], which can take into account features of a results list (*gating features*), to adjust the overall weight of documents in that list. The model can then learn, for example, to down-weight scores from an unpromising list, in order to optimize the effectiveness of the final merged list.

Metasearch and data fusion. We propose merging the results of different query reformulations; hence our work is related to metasearch and data fusion. A core concern in those areas is the normalization and combination of scores [21]. CombSUM and CombMNZ [23] are two methods that aggregate the scores from multiple lists that serve as well-known baselines in our setting. Our techniques also aggregate scores, but the contribution from each reformulation varies per query.

We explore methods that merge the results of different reformulations on a single collection. Some related work on supervised federated search [24, 34] merges the results of a single query on multiple collections. In both settings, the merging can be weighted according to the quality of the reformulation/collection. However, the available features differ. Moreover, these previous methods consist of two separate steps: first, quality is predicted, and then merging is performed based on predicted quality. In such a scheme it is unclear how a predicted quality metric (for example, P@10 or MAP in [33]) best translates into a merging weight. We instead include quality-indicating gating features as part of a joint training process to optimize the NDCG of the final merged list.

Query difficulty prediction. Performance (difficulty) prediction can play an important role in merging the results of query reformulations. Ideally, the merging system should be able to downgrade (or, in extreme cases, ignore) the results of reformulations that are less likely to contain relevant documents.

An early performance prediction feature was *clarity* [7], which measures the distance between the search results and the background distribution of the general corpus; good search results should differ from the background. This approach has been followed by methods that consider perturbation and score distributions. Perturbation-based methods [28, 33] measure changes when documents, queries or the ranker is perturbed; a ranking should be robust to such perturbations. Spatial autocorrelation methods [11] compare the score distribution of similar documents for performance prediction.

The query difficulty predictor of Yom-Tov et al. [33] is based on the overlap between a query's results and the results of all its one-word sub-queries. Easier queries tend to have more overlap. Their metasearch experiments demonstrated that the two stage merging method described above improves performance. However, they did not experiment with query reformulation, and their sub-query performance prediction method may not adapt well to expanded queries that have many terms.

Several recent works have utilized supervised learning to *select* query reformulations. Kumaran and Carvalho [19] performed query reduction by ranking sub-queries of the original query in a learning to rank framework that used content-based performance prediction features such as clarity and mutual information gain. Balasubramanian et al. [2] extended this methodology to the problem of query reduction for long web queries by developing performance prediction features that are more effective in the web setting and can be efficiently computed at the time of ranking. These features combine document-level features and retrieval scores from the top-k list for each reduced query, and are similar to our features described in Section 4.3. They consider several learning approaches for selecting a reduced query and find that regression to predict NDCG change of the reformulated query is the most effective; they also show that interleaving results from the original query with those from the reduced query further improves performance. Dang et al. [10] presented a supervised technique that successfully ranked query reformulations based on their retrieval performance. Xue et al. [31] used a conditional random field model to rank different subsets of query words according to their predicted performance, and showed that their approach can improve the performance on long queries by dropping some of the query terms.

Balasubramanian and Allan [1] use the regression approach in [2] to learn to select between different rankers for each query, and show that selecting the better of two rankers according to predicted performance outperforms individual rankers on the LETOR 3.0 GOV2 collection. Balasubramanian, Kumaran and Carvalho [3] also show that regression techniques using similar features constructed from the top-k lists are effective for predicting query performance in a web collection.

3. MERGING QUERY REFORMULATIONS

One approach to avoid the risk of bad query reformulations is to merge the results of several different reformulations. In this section, we first review CombSUM and CombMNZ [23], two unsupervised merging methods that can be applied to this task. We also introduce CombRW,

a variant of CombSUM that takes into account the random walk probabilities of each rewrite. We then introduce λ-Merge: a supervised, end-to-end merging method. Unlike the previous two-stage supervised merging methods [24, 33], the quality-prediction and merging components of λ-Merge are jointly trained to directly optimize a retrieval effectiveness measure such as NDCG.

3.1 Unsupervised merging

CombSUM and CombMNZ are unsupervised merging methods that are simple, effective and well-studied. Both methods are based on document retrieval scores, rewarding documents that have high scores and appear in multiple results lists. CombSUM sums a document's scores from all lists where it was present. CombMNZ additionally multiplies the CombSUM score by the number of lists that contained the document. Score normalization is important when applying these methods since different results lists may be scored on different scales (e.g. a long query might have generally lower scores than a short query). Also, a document that is absent from a list contributes nothing to the sum, so it is important for the documents that *are* present to have non-negative scores. For these reasons, it is common to normalize scores to the $[0, 1]$ interval before combination. Several of the features that we will use with λ-Merge rely on similar normalization ideas.

We define some notation here. For a query q, let $q^{(1)}, q^{(2)}, \ldots, q^{(K)}$ be the different formulations (including the original query q), and let $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(K)}$ be the corresponding results lists. We use a standard normalization method that shifts and scales the scores for each $\mathcal{D}^{(k)}$ into the interval $[0, 1]$. We refer to the resulting value as $\text{NormScore}(q^{(k)}, d)$. With this notation, CombSUM and CombMNZ are defined as follows:

$$\text{CombSUM}(d) = \sum_{k\,:\,d \in \mathcal{D}^{(k)}} \text{NormScore}(q^{(k)}, d), \tag{1}$$

$$\text{CombMNZ}(d) = \text{CombSUM}(d) \times |\{k\,:\,d \in \mathcal{D}^{(k)}\}|. \tag{2}$$

These simple combination rules are among the most effective merging methods [21]. In our experiments, we observed that CombSUM and CombMNZ had similar effectiveness, and hence we restrict our attention to the simpler CombSUM method for the rest of this paper.

CombRW. CombSUM and CombMNZ treat all result lists equally. They do not give special status to the user's original query, and do not take into account any information about the reformulations, e.g., how highly the reformulation scored under the random walk. We suggest a weighted version of CombSUM to consider such information.

$$\text{CombRW}(d) = \sum_{k\,:\,d \in \mathcal{D}^{(k)}} \text{NormScore}(q^{(k)}, d) \times W(\mathcal{D}^{(k)}) \tag{3}$$

Here, $W(\mathcal{D}^{(k)})$ denotes the weight assigned to the result list returned for the k-th reformulation. Weights may be assigned according to predicted performance [19], expected reformulation similarity (e.g. random-walk probability [6]), or any other measure of quality. We choose to set the weight $W(\mathcal{D}^{(k)})$ to be the probability assigned to $q^{(k)}$ by a random walk in the Query-URL graph starting at q (see the previous section for details), and refer to this variant of CombSUM as CombRW. Note that this type of two-stage score normalization is analogous to some federated search blending techniques where the rescaled server-specific document scores are normalized according to server quality scores [5].

3.2 Supervised merging

We now present the details of λ-Merge, a general *learning to merge* system that learns a ranking function to merge multiple results lists. λ-Merge builds on the basic ideas of CombSUM and related algorithms by drawing from the more flexible class of gated neural-network score-combination functions that (1) utilize multiple query-document features instead of just the retrieval score, (2) weight contributions from each reformulation according to multiple features, such as those predicting list quality and query drift, so that contributions vary depending on the query and reformulation, and (3) are trained to optimize a retrieval metric. As far as we know, λ-Merge is the first merging method that is trained using relevance judgments to directly optimize desired IR metrics such as NDCG or MAP.

The algorithm is general and can be applied in merging scenarios beyond the reformulation work of this paper, for example, the scenario of a single query issued to multiple collections. However, we leave such applications for future research.

λ-Merge uses two types of features: (1) *query-document* features that describe the match between a query formulation and the document (e.g. language modeling score), and (2) *gating* features that describe the quality of the reformulation and its results list (e.g. query clarity score). In general, the gating features can adjust the scores of an entire results list. For example, each document returned for 'mass of jupiter' and 'jupiter facts' has different query-document features, while gating features belong to the entire list and might cause the method to give more weight to the results for 'mass of jupiter', since it is a higher quality rewrite.

Model. Figure 1 gives an overview of λ-Merge. Let $\boldsymbol{x}_d^{(k)}$ be the vector of query-document features for document d and the k-th query reformulation $q^{(k)}$. These features typically include the score and rank information from a base ranker. If document d does not appear in the results list $\mathcal{D}^{(k)}$ for the k-th reformulation, default values must be assigned. In our experiments, we typically assign default values such that document d inherits the score and rank features of the lowest-ranked document that does appear in $\mathcal{D}^{(k)}$.

Let $f(\boldsymbol{x}; \boldsymbol{\theta})$ be the **scoring function**, with parameters $\boldsymbol{\theta}$. A given document d thus receives a score $f(\boldsymbol{x}_d^{(k)}; \boldsymbol{\theta})$ for the k-th reformulation.

Let $\boldsymbol{z}^{(k)}$ be the vector of gating features for reformulation $q^{(k)}$. These describe qualities of the k-th reformulation and its results list $\mathcal{D}^{(k)}$ as a whole.

The **gating network** determines the contribution of each reformulation to the final score. It outputs a mixing weight

$$\alpha_k = \text{softmax}(\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(K)}; \boldsymbol{\pi}) = \frac{\exp(\boldsymbol{\pi}^T \boldsymbol{z}^{(k)})}{\sum_p \exp(\boldsymbol{\pi}^T \boldsymbol{z}^{(p)})}$$

for the k-th results list. The mixing weights are non-negative and are normalized to sum to one, thus giving a relative weight to each reformulation. However, a linear gating net-

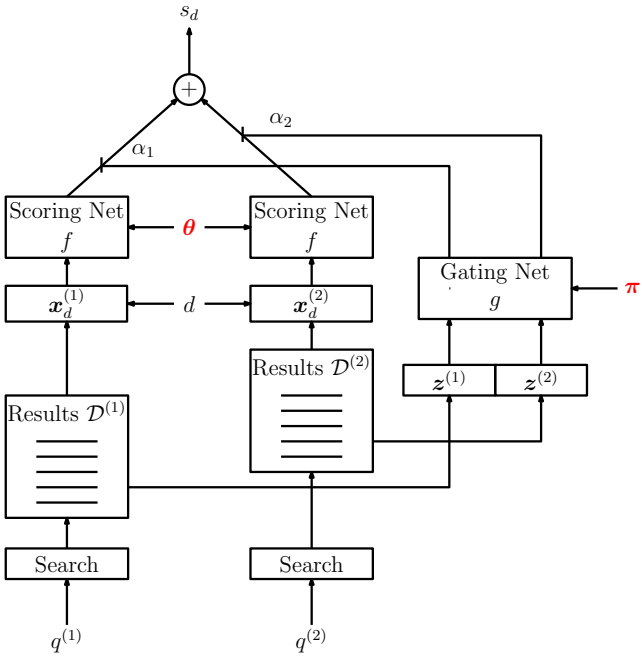

Figure 1: The architecture of λ-Merge for blending the results of multiple query reformulations.

work is also likely to work, which we suggest for future experiments.

The final ranker score s_d for the document is given by weighting the individual reformulation scores $f(\boldsymbol{x}_d^{(k)};\boldsymbol{\theta})$ by the mixing weights from the gating network:

$$s_d = \sum_k \alpha_k \cdot f(\boldsymbol{x}_d^{(k)};\boldsymbol{\theta}). \quad (4)$$

The scoring function f can be implemented by any differentiable function, such as a linear function, a neural network, or a set of boosted decision trees. The function should be flexible enough to normalize scores from the different reformulations so that they share a common scale suitable for combination. We choose f to be a fully connected two-layer neural network with four hidden units, each having a tanh activation function, and with an output that is a linear combination of the hidden units.

Related Architectures. The λ-Merge ranking function is a mixture of scoring functions. This is related to mixture of experts architectures [14], where the experts are neural networks [25]. Traditionally, each expert is specialized to handle a different part of the same input space; whereas we employ a single scoring function that receives different inputs (based on different reformulations).

Training. The scoring parameters $\boldsymbol{\theta}$ and gating weights $\boldsymbol{\pi}$ of λ-Merge are trained to optimize NDCG using a method based on LambdaRank [4]. Alternatively, the method can optimize Precision@k, MAP, or ERR. A key feature of all these retrieval metrics is that they depend on the ranks of documents, which are discontinuous with respect to the document scores computed by retrieval systems. (The ranks are obtained by sorting the scores.) Hence, gradient-based optimization is challenging. LambdaRank sidesteps this problem by using a smoothed version of the objective. It has empirically been proven to reach optima of NDCG successfully [12].

Denote the smoothed objective by C. To use LambdaRank, it is sufficient to implement its gradients with respect to score and gating parameters, namely $\partial C/\partial\theta_\ell$ and $\partial C/\partial\pi_m$. By the chain rule, these can be decomposed, for example, $\partial C/\partial\theta_\ell = \sum_d (\partial C/\partial s_d)\cdot(\partial s_d/\partial\theta_\ell)$. For NDCG, MAP, and some other IR metrics, LambdaRank derivatives are

$$\partial C/\partial s_d = \sum_e |\Delta_{de}|\,(\mathbb{I}_{d\succ e} - 1/(1+\exp(s_e - s_d))). \quad (5)$$

Here, $|\Delta_{de}|$ is the absolute change in the metric if items d and e were swapped in the current ranking. For NDCG, this change is $|2^{l_d}-2^{l_e}||1/\log_2(r_d+1)-1/\log_2(r_e+1)|/\mathrm{DCG}_{\max}$ for relevance labels l_d and l_e, and ranks r_d and r_e, respectively. $\mathrm{DCG}_{\max}$ is the DCG value achived by an ideal ranking for the query. The indicator $\mathbb{I}_{d\succ e}$ is 1 when document d is judged more relevant than e, and 0 otherwise. See also Section 6.2 of [4].

The remaining derivatives are computed by doing backpropagation separately in each of the scoring and gating networks, where the calculation for either network requires only the current output values from the other network. Mathematically, this is because differentiating (4) yields

$$\frac{\partial s_d}{\partial\theta_\ell} = \sum_k \alpha_k \cdot \frac{\partial}{\partial\theta_\ell} f(\boldsymbol{x}_d^{(k)};\boldsymbol{\theta}), \quad \frac{\partial s_d}{\partial\pi_m} = \sum_k \frac{\partial\alpha_k}{\partial\pi_m}\cdot f(\boldsymbol{x}_d^{(k)};\boldsymbol{\theta}).$$

Hence, the scoring function parameters $\boldsymbol{\theta}$ are updated by using standard backprop to compute $\frac{\partial}{\partial\theta_\ell} f(\boldsymbol{x}_d^{(k)};\boldsymbol{\theta})$. The gating net parameters $\boldsymbol{\pi}$ are updated by backprop through the softmax function. Let $\beta_k = \exp(\boldsymbol{\pi}^T\boldsymbol{z}^{(k)})$, so that $\alpha_k = \beta_k/\sum_p \beta_p$. Then

$$\frac{\partial\alpha_k}{\partial\pi_m} = \Big(\sum_p \beta_p\Big)^{-2}\Big(\beta_k z_m^{(k)}\cdot\sum_p \beta_p - \beta_k\cdot\sum_p \beta_p z_m^{(p)}\Big).$$

We train by stochastic gradient descent and consider queries in random order during each training epoch. We batch parameter updates by query for faster training [4]. In all experiments, we fix a step size of 10^{-3} and train for 25 epochs.

4. EXPERIMENTS

We conduct our primary experiments using ranking scores, query logs, and click data from the Bing search engine. Our techniques can use any source of reformulation candidates; here we have generated reformulations via a two-step random walk on a click graph based on several months of query logs. At least one random walk reformulation is generated for roughly 40% of all queries in the search engine's workload.[1] From this subset, we sample 4552 queries and split them into a training set (2303 queries) and a testing set (2249 queries).

Unless otherwise specified, we only consider one reformulation candidate for each query. The top-ranked random-walk reformulation candidates are diverse; 24% have words removed, 26% have words added, and 48% have words both added and removed.[2] We compare the retrieval effectiveness

[1]Coverage may improve by using a larger query log, or a longer random walk.

[2]The remaining 2% have the original words in a different order, for example 'movies 2008' and '2008 movies'.

Table 1: List of features.

Query-document features	Score, Rank, $\text{NormScore}_{[0,1]}$, $\text{NormScore}_{\mathcal{N}(0,1)}$, IsTopN
Gating features (difficulty)	ListMean, ListStd, ListSkew, Clarity, RewriteLen, RAPP
Gating features (drift)	IsRewrite, RewriteRank, RewriteScore, Overlap@N

of different methods by measuring NDCG with 5 grades of relevance judgments (Bad, Fair, Good, Excellent, and Perfect). We also use NDCG as our optimization metric for λ-Merge.

4.1 Features

A significant strength of λ-Merge is that one may employ many different features that together are predictive of document relevance or results-list quality. Table 1 lists the features we employed in our Bing experiments. In the following discussion we describe each feature in more details.

Query-document features. These are features for a given query-document pair $(q^{(k)}, d)$ and correspond to the $\boldsymbol{x}^k_{(d)}$ vectors in the architecture described in Figure 1.

1. Score: the original search engine ranker score. On the Bing data, it is the Bing ranker score. On GOV2, we used the default ranker of Indri.
2. Rank: the position of d in ranked list $\mathcal{D}^{(k)}$.
3. $\text{NormScore}_{[0,1]}$: the score of d in $\mathcal{D}^{(k)}$ after the top-10 scores are shifted and scaled into the interval $[0, 1]$ using min-max normalization.
4. $\text{NormScore}_{\mathcal{N}(0,1)}$: the score of d in $\mathcal{D}^{(k)}$ after the top-10 scores are shifted and scaled to have mean zero and unit variance (indicated by $\mathcal{N}(0, 1)$.)
5. IsTopN: A binary feature to indicate if this document is within the top-N results, where $N \in \{1, 3, 5, 10\}$.

Gating Features (difficulty & drift). Gating features correspond to $\boldsymbol{z}$ in Figure 1, and we have a total of 13 of them. Some gating features are designed to be predictive of the overall quality of the results list $\mathcal{D}^{(k)}$. Other features are designed to detect drift.

1. IsRewrite: A binary flag to distinguish between the user's original query and its reformulations.
2. RewriteScore: The query reformulation score (random walk probabilities in our case). This feature is defined to take value 1 for the original query.
3. RewriteRank: 0 for the original query, then n for the n-th random walk reformulation as sorted by random walk probability.
4. ListMean, ListStd, ListSkew: Respectively, mean, standard deviation and skew of raw ranking scores over each result list. The distribution of document scores has been cited as an important feature for query performance prediction [28].
5. Clarity: This feature is a modified version of the clarity score [8], designed to measure the coherence of language usage in documents from $\mathcal{D}^{(k)}$. It is defined as the Kullback-Leibler divergence between a query language model and the collection language model. We construct the query language model using the snippets of the top-10 results and build the background language model using a reference query log.
6. Overlap@N: The number of documents that appear in both the top-N results for the query reformulation and the top-N results for the original query. This feature is defined to take value N for the original query.
7. RewriteLen: The number of words in the reformulation. Long queries tend have worse retrieval performance [2], so this is a query difficulty feature.
8. RAPP [3]: The output of a linear regression model using the above features that is trained to predict NDCG. This is described in more details next.

4.2 Baselines

We compare the results of our two blending techniques (CombRW and λ-Merge) against four different baselines:

- **ORG:** The results for the original query with no merging or reformulation selection.
- **CombSUM:** Unsupervised baseline, described in Section 3.1.
- **RAPP-L:** Balasubramanian et al. [2, 3] proposed *rank-time performance prediction*, which uses linear regression or random forests to predict NDCG or ΔNDCG between a query and a reformulation. We implemented RAPP using linear regression to predict ΔNDCG, and denote this RAPP-L to emphasize the choice of linear regression, which performed as well as random forests in [3]. A difference in our setting is that we use the post-retrieval features described earlier (for λ-Merge) to train the regressor, while they used raw ranking features (e.g. BM25 and clicks).
- **RAPP(Ω):** The oracle for *selection* techniques such as RAPP, which always chooses the result list with the highest NDCG@5. Despite setting an upper bound for methods that select a single query, this baseline can be outperformed by merging methods.

4.3 Retrieval effectiveness

Table 2 presents overall NDCG results of the original query alone, reformulation alone, and then the various reformulation selection and merging techniques. All differences between different systems are statistically significant ($p < 0.01$) by the t-test (the only exception is for NDCG@5 between λ-Merge and RAPP (Ω)). The reformulation candidates (RW1) are on average much worse than the user's original query (ORG). Of the merging techniques, CombSUM is not robust to bad reformulations and has NDCG between that of ORG and RW1, while CombRW and λ-Merge both achieve gains via merging. The query selection techniques do not perform as well as λ-Merge; even the oracle RAPP(Ω) is unable to consistently perform better.

Table 2: The performance of different techniques averaged over the testing set (2303 queries). For each original query (ORG), the top-ranked random-walk candidate (RW1) is used as the reformulation candidate. All differences (except for NDCG@5: RAPP(Ω) vs. λ-Merge) are statistically significant ($p < 0.01$).

	NDCG@5	NDCG@10
ORG	0.538	0.524
RW1	0.422	0.387
CombSUM	0.510	0.486
CombRW	0.542	0.516
RAPP-L	0.534	0.524
λ-Merge	0.555	0.539
RAPP(Ω)	0.556	0.530

Robustness analysis. We use bubble plots to study the robustness of our retrieval methods (Figures 2 and 3). Each query is a assigned a location on the 2-D plot to indicate both the quality of the reformulation candidate for that query (on the x-axis, measured by difference in NDCG@5 compared to the original query) and the performance of the final results list the retrieval system produces for that query (on the y-axis, measured in overall gain or loss in NDCG@5). The size of the bubble indicates the number of queries that fall into that category. For example, bubbles in the upper right correspond to queries where both the reformulation candidate and the overall system output are better than the original query, and a large circle at $(0, 0)$ means that for many queries, both the reformulation candidate and the overall system produce output lists with the same NDCG@5 as the original query. In our experiments, the number of queries at 0,0 is usually around 1200 ± 100.

Figure 2 shows bubble plots for the unsupervised methods. As might be expected, the unsupervised CombSUM method produces gains when given high quality reformulations, and losses when given lower quality reformulations, as evidenced by points in the upper right and lower left quadrant. Overall CombSUM produces losses for many queries and is poor at recovering from bad reformulations. In contrast, CombRW achieves reasonable gains with very few losses, as evidenced by few points in the lower half of the plot. Hence, weighting by random walk score can reduce losses due to bad reformulations.

Figure 3 shows bubble plots for the supervised approaches. The RAPP(Ω) oracle always picks the best reformulation, and is therefore neutral on the left side of the plot and positive on the right side. When compared with RAPP(Ω), the bubble plots for the methods CombSUM, CombRW and λ-Merge clearly illustrate that merging can outperform query selection; all three approaches have points above the envelope of RAPP(Ω), indicating performance that is better than the maximum NDCG of ORG and RW1 on those queries.

RAPP-L bubbles that differ from RAPP(Ω) indicate mistakes of the system: the lower left ones are false positives, choosing bad reformulations, and the horizontal right ones are false negatives, failing to choose a good reformulation. Compared to these two, λ-Merge has a greater variety of behavior. The lack of points in the lower left quadrant demonstrates robustness to bad reformulations; in fact for many bad reformulations λ-Merge still achieves *gains* in retrieval

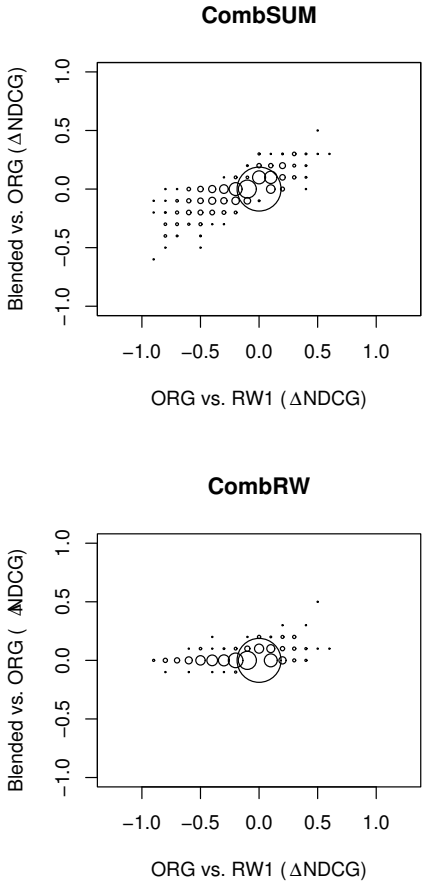

Figure 2: The robustness and effectiveness analysis of different unsupervised techniques with respect to the reformulation quality. The x-axis shows the ΔNDCG@5 between the original query (ORG) and the reformulation candidate (RW1). The y-axis is the ΔNDCG of the final list with respect to the original query.

effectiveness. This is evidence in favor of our supervised approach and for directly optimizing NDCG. We also see that merging methods in general (CombRW, λ-Merge) have much greater upside than methods that pick a single query (RAPP-L).

5. DISCUSSION

We showed that merging the results of different query reformulation methods could improve both robustness and overall effectiveness relative to the baseline methods. In this section, we provide further insights into λ-Merge. We also investigate the impact of increasing the number of query reformulations on different methods. Finally, we report results on the publicly available GOV2 dataset for comparison purposes.

Multiple query reformulations. In all the experiments reported so far we considered only one reformulation per query. In this section, we repeat the experiments with 5 query refor-

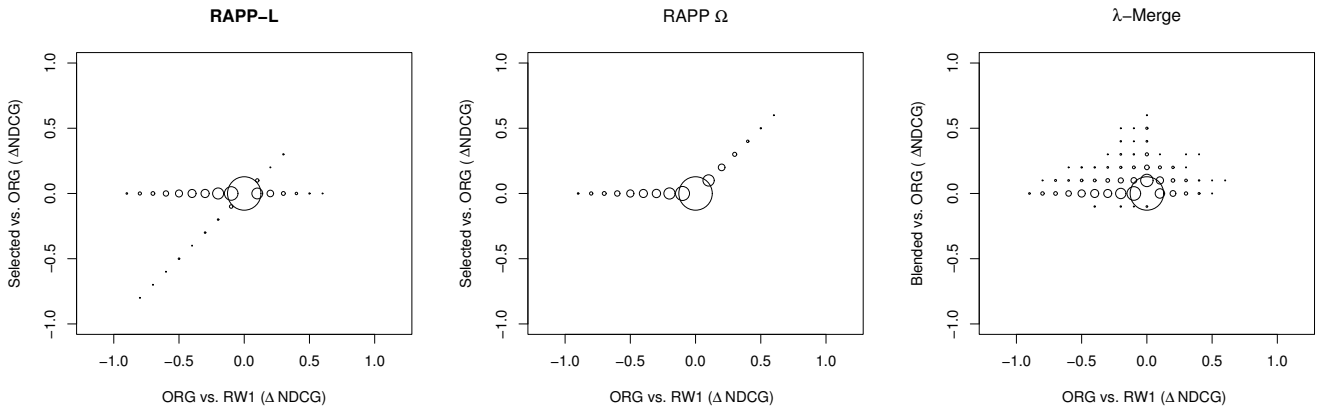

Figure 3: The robustness and effectiveness analysis of different unsupervised techniques with respect to the reformulation quality. The x-axis shows the ΔNDCG@5 between the original query (ORG) and the reformulation candidate (RW1). The y-axis is the ΔNDCG of the final list with respect to the original query.

Table 3: The performance of different techniques averaged over a subset of queries with at least five random-walk candidates (1872 queries).

	1 Reformulation		5 Reformulations	
	NDCG@5	NDCG@10	NDCG@5	NDCG@10
ORG	0.550	0.535	0.550	0.535
CombSUM	0.521	0.496	0.497	0.455
CombRW	0.554	0.527	0.555	0.524
RAPP-L	0.546	0.531	0.550	0.535
λ-Merge	0.566	0.550	0.563	0.548
RAPP(Ω)	0.567	0.541	0.587	0.545

mulations and investigate the impact on different methods. We only include queries that have at least 5 random walk candidates, which reduces the size of training and testing sets by about 16%. Therefore, when comparing the results with the previous experiments, we report all the results on this subset. The results are shown in Table 3. Firstly, compared to the previous experiments in Table 2, the original queries (ORG) have better NDCG on this new subset. This is because the retained queries have more clicks and richer usage data that improve the quality of the baseline ranking. The NDCG of CombSUM drops significantly ($p < 0.01$) with 5 compared to 2 reformulations, due to the diminished quality of lower-ranked random walk candidates (lower-ranked candidates are occasionally good, as demonstrated by the increased performance of RAPP(Ω)). The RAPP-L classifier learns to almost never trust any query reformulation which effectively leads it to match ORG (we note that this is a different behavior than [3] which merged rankers of similar effectiveness). CombRW and λ-Merge remain relatively robust, and λ-Merge continues to significantly outperform all the alternatives ($p < 0.01$) and surpass the selection oracle for NDCG@10.

We note that an important feature of this experiment setup is the fact that the expanded pool of reformulations is significantly lower in quality, so the experiment primarily demonstrates the robustness of CombRW and λ-Merge. In a scenario with a large pool of reformulations of uniform quality, one would expect performance of the merging methods to improve as more reformulations are made available. We leave exploration of this scenario for future work.

Inside the black box. λ-Merge's gating net is used to appropriately weight scores from each reformulation. The problem is one of trading off (1) good reformulations (in the query difficulty sense), (2) reformulations that introduce new documents over the original query, and (3) reformulations that are faithful to the original query (avoiding topic drift). λ-Merge training tunes the gating network to balance these factors for the best NDCG value.

We examined the effect of training λ-Merge with different gating features (Figure 4). As the baseline (corresponding to ΔNDCG being 0 in the diagram), we picked an empty set of gating features, yielding an averaging gating network that simply averages the scores from reformulations. We compared this against gating functions of individual and multiple gating features.

Some individual features had almost no benefit: in particular, traditional performance prediction features including *Clarity*, *ListMean*, *ListStd*, *ListSkew* showed only tiny improvements over the averaging gating network.

Features that express query drift were more effective; individual features capturing overlap between original and reformulation results were the better the more they focused on the top ranks, with *Overlap@1* excelling among these.

The best individual features were variations on distinguishing the original from the reformulated query. The *IsRewrite* feature was remarkably effective; when used by itself, it corresponds to a gating network that gives the original query a constant weight, and all reformulations of it some other constant weight, tuned by λ-Merge. Tuning gave weights (0.8, 0.2) when merging the original with one random walk candidate, and (0.63, 0.07, 0.07, 0.07, 0.07, 0.07) when merging with five candidates. The *RewriteRank* feature was equivalent to IsRewrite for one reformulation candidate, but for multiple candidates it could distinguish the order of the candidates. Similarly, the *RewriteScore* incorporated the random walk score. These refinements performed slightly better than *IsRewrite*, but the dominant effect seems

to be in distinguishing the original query from the reformulations.

Combining multiple weak features proved beneficial: for example, the *ListStats* includes *ListMean*, *ListStd*, *ListSkew*, and gave good results even though the individual features were weak. This result is very promising for other merging applications because it uses only information about score distributions and no features specific to the query reformulation application. *Overlap@** included all overlap features and also performed better than any of its constituents. Including all thirteen features gave the best overall result; the gating network then weighted the original query by 0.77 on average across the test queries, with a standard deviation of 0.12. However, the gain of using all features over using the best individual features was small. A possible limitation here is the linear gating network — a promising avenue of future work is to explore whether a non-linear gating function is better able to make use of combinations of features. Overall, λ-Merge proved quite robust to combinations of strong and weak gating features. An experimental combination of 28 features, many of which proved to be individually poor, did not degrade performance from combinations containing only individually good features.

We also examined the weights of the scoring network in λ-Merge (not shown). Of the two groups of features, normalizations of base ranker scores and rank position features, our main observation was that the scoring network utilized all score features as well as the *IsTop1* feature to give an extra boost to documents that appear first in a results list (appearing in the top 3 is given much less weight).

In summary, λ-Merge incorporates a variety of features, and finds interesting trade-offs between them.

5.1 Experiments on GOV2

For comparison purposes, we also experimented on the TREC GOV2 dataset,[3] using the `<title>` of TREC topics 751–800 for training and topics 801–850 for testing. For each training and test query, we use the top five random walk candidates from the search engine logs described earlier.[4] The inputs for merging are the results of the original query (ORG) and results of the random walk candidates.

Table 4 shows results on the test set. Here, we measure the NDCG results at cut-off 100, but also evaluate the results according to *precision at 5* (P@5) which emphasizes relevance at top ranks. Overall, both CombRW and λ-Merge substantially improve over the ORG baseline. λ-Merge also manages to outperform the RAPP(Ω) oracle on P@5. We found similar trends on the TREC WT10g dataset, although improvements were smaller.

6. CONCLUSIONS

In this paper, we explored merging methods to enhance retrieval effectiveness under query reformulation, in particular, when reformulation candidates are generated by random walks on the click graph. These methods issue multiple queries and merge their results: CombRW is a simple unsupervised method that weights results lists according to random walk probabilities, and λ-Merge is a supervised merging technique that extends LambdaRank. The λ-Merge method learns a merging function to directly optimize a retrieval metric (e.g. NDCG), and can utilize user-defined features that indicate relevance of the documents, as well as quality and drift of results lists as a whole.

In experiments on Bing data, the simple CombSUM merging method performed poorly: robustness analysis showed that it was heavily degraded by reformulations that were worse than the original query. By considering walk probabilities, CombRW eliminated much of this downside risk and outperformed the original query. Of the merging methods, λ-Merge achieved the best performance. Merging proved much more powerful than selecting a single formulation: for many individual queries, λ-Merge performed better than any single formulation, and overall was competitive with an oracle that always chose the best single query.

Our feature analysis shows that λ-Merge is indeed taking into account a wide variety of document and gating features. The gating network performed very well with simple features that distinguish the original query from the reformulation, but λ-Merge also made effective use of combinations of weak features, and achieved the best performance when presented with all of the features.

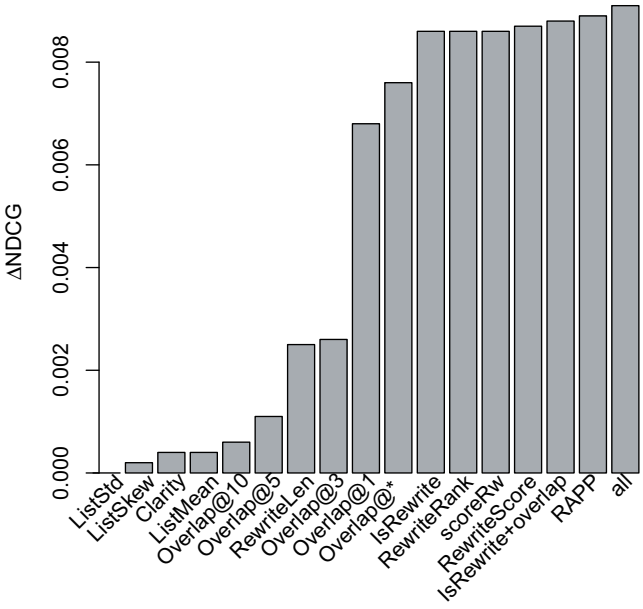

Figure 4: Retrieval accuracy for various combinations of gating features when one random walk candidate is used for merging. Similar trends were found for experiments with five random walk candidates.

Table 4: Merging the results of the original query with up to five reformulation candidates extracted from query logs on the TREC GOV2 dataset.

	P@5	NDCG@100
ORG	0.548	0.431
CombRW	0.584	0.438
λ-Merge	0.596	0.447
RAPP(Ω)	0.592	0.457

[3] `http://ir.dcs.gla.ac.uk/test_collections/gov2-summary.htm`

[4] In the test set there were no random-walk candidates for 22 queries. For those, the results for the original query remain unchanged.

One avenue for future work is to explore different architectures and training metrics within the λ-Merge architecture. We conjecture that nonlinear gating functions may make better use of a large number of features. Another open area is to explore reformulations from multiple sources, such as session rewrites, anchor-text, etc. λ-Merge may also be beneficial in other merging scenarios such as federated search.

Acknowledgment

The authors are thankful to Ed Snelson for his insightful suggestions on early stages of this work.

7. REFERENCES

[1] N. Balasubramanian and J. Allan. Learning to select rankers. In *Proceedings of the 33rd ACM SIGIR conference*, pages 855–856, Geneva, Switzerland, 2010.

[2] N. Balasubramanian, G. Kumaran, and V. Carvalho. Exploring reductions for long web queries. In *Proceedings of the 33rd ACM SIGIR conference*, pages 571–578, Geneva, Switzerland, 2010.

[3] N. Balasubramanian, G. Kumaran, and V. Carvalho. Predicting query performance on the web. In *Proceedings of the 33rd ACM SIGIR conference*, pages 785–786, 2010.

[4] C. Burges, R. Ragno, and Q. Le. Learning to rank with nonsmooth cost functions. In *Proceedings of the 20th NIPS conference*, pages 193–200, Vancouver, Canada, 2006.

[5] J. Callan. Distributed information retrieval. In B. Croft, editor, *Advances in information retrieval, Chapter 5*, volume 7 of *The Information Retrieval Series*, pages 127–150. Kluwer Academic Publishers, 2000.

[6] N. Craswell and M. Szummer. Random walks on the click graph. In *Proceedings of the 30th ACM SIGIR conference*, pages 239–246, Amsterdam, The Netherlands, 2007.

[7] S. Cronen-Townsend, Y. Zhou, and B. Croft. Precision prediction based on ranked list coherence. *Information Retrieval*, 9(6):723–755, 2006.

[8] S. Cronen-Townsend, Y. Zhouand, and B. Croft. Predicting query performance. In *Proceedings of the 25th ACM SIGIR conference*, pages 299–306, Tampere, Finland, 2002.

[9] V. Dang and B. Croft. Query reformulation using anchor text. In *Proceedings of the Third ACM WSDM conference*, pages 41–50, New York, NY, 2010.

[10] V. Dang, M. Bendersky, and B. Croft. Learning to rank query reformulations. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 807–808, Geneva, Switzerland, 2010.

[11] F. Diaz. Performance prediction using spatial autocorrelation. In *Proceedings of the 30th ACM SIGIR conference*, pages 583–590, Amsterdam, The Netherlands, 2007.

[12] P. Donmez, K. Svore, and C. Burges. On the local optimality of LambdaRank. In *Proceedings of the 32nd ACM SIGIR conference*, pages 460–467, Boston, MA, 2009.

[13] J. Guo, G. Xu, H. Li, and X. Cheng. A unified and discriminative model for query refinement. In *Proceedings of the 31st ACM SIGIR conference*, pages 379–386, Singapore, 2008.

[14] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, 1991.

[15] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[16] R. Jones and D. Fain. Query word deletion prediction. In *Proceedings of the 26th ACM SIGIR conference*, pages 435–436, Toronto, Canada, 2003.

[17] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th WWW conference*, pages 387–396, Edinburgh, UK, 2006.

[18] R. Kraft and J. Zien. Mining anchor text for query refinement. In *Proceedings of the 13th WWW conference*, pages 666–674, New York, NY, USA, 2004.

[19] G. Kumaran and V. Carvalho. Reducing long queries using query quality predictors. In *Proceeding of the 32nd ACM CIKM conference*, pages 564–571, Boston, MA, USA, 2009.

[20] V. Lavrenko and B. Croft. Relevance based language models. In *Proceedings of the 24th ACM SIGIR conference*, pages 120–127, New Orleans, LA, 2001.

[21] M. Montague and J. Aslam. Metasearch consistency. In *Proceedings of the 24th ACM SIGIR conference*, pages 386–387, New Orleans, LA, 2001.

[22] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM CIKM conference*, pages 42–49, Washington, DC, 2004.

[23] J. Shaw and E. Fox. Combination of multiple searches. In *The Second Text REtrieval Conference*, pages 243–252, 1994.

[24] L. Si and J. Callan. Modeling search engine effectiveness for federated search. In *Proceedings of the 28th ACM SIGIR conference*, pages 83–90, Salvador, Brazil, 2005.

[25] M. Szummer and C. M. Bishop. Discriminative writer adaptation. In *10th Intl. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 293–298, Oct. 2006.

[26] T. Tao and C. Zhai. Regularized estimation of mixture models for robust pseudo-relevance feedback. In *Proceedings of the 29th ACM SIGIR conference*, pages 162–169, Seattle, WA, 2006. ACM.

[27] E. Terra and C. Clarke. Scoring missing terms in information retrieval tasks. In *Proceedings of the 13th ACM CIKM conference*, pages 50–58, Washington, D.C., USA, 2004.

[28] V. Vinay, I. Cox, N. Milic-Frayling, and K. Wood. On ranking the effectiveness of searches. In *Proceedings of the 29th ACM SIGIR conference*, pages 398–404, Seattle, Washington, 2006.

[29] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *Proceedings of the 17th ACM CIKM conference*, pages 479–488, Napa Valley, CA, 2008.

[30] J. Xu and B. Croft. Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th ACM SIGIR conference*, pages 4–11, Zurich, Switzerland, 1996.

[31] X. Xue, S. Huston, and 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, Toronto, ON, 2010.

[32] Z. Yin, M. Shokouhi, and N. Craswell. Query expansion using external evidence. In *Proceedings of the 31st ECIR conference*, pages 362–374, Toulouse, France, 2009.

[33] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *Proceedings of the 28th ACM SIGIR conference*, pages 512–519, Salvador, Brazil, 2005.

[34] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow. Metasearch and federation using query difficulty prediction. In *Proceedings of the ACM SIGIR Workshop on Predicting Query Difficulty*, Salvador, Brazil, 2005.