Boosting Search Performance Using Query Variations

RODGER BENHAM, RMIT University
JOEL MACKENZIE, RMIT University
ALISTAIR MOFFAT, The University of Melbourne
J. SHANE CULPEPPER, RMIT University

Rank fusion is a powerful technique that allows multiple sources of information to be combined into a single result set. Query variations covering the same information need represent one way in which different sources of information might arise. However, when implemented in the obvious manner, fusion over query variations is not cost-effective, at odds with the usual web-search requirement for strict per-query efficiency guarantees. In this work we propose a novel solution to query fusion, by splitting the computation into two parts – one phase that is carried out offline, to generate pre-computed centroid answers for queries addressing broadly similar information needs; and then a second online phase that uses the corresponding topic centroid to compute a result page for each query. To achieve this, we make use of score-based fusion algorithms whose costs can be amortized via the pre-processing step, and which can then be efficiently combined during subsequent per-query re-ranking operations. Experimental results using the ClueWeb12B collection and the UQV100 query variations demonstrate that centroid-based approaches allow improved retrieval effectiveness at little or no loss in query throughput or latency, and within reasonable pre-processing requirements. We additionally show that queries that do not match any of the pre-computed clusters can be accurately identified and efficiently processed in our proposed ranking pipeline.

CCS Concepts: • **Information systems → Retrieval efficiency**; **Search engine architectures and scalability**; **Information retrieval query processing**; **Combination, fusion and federated search**;

Additional Key Words and Phrases: Rank fusion, Dynamic Pruning, Efficiency, Effectiveness, Experimentation

1 INTRODUCTION

Rank fusion is used to combine knowledge from different result sets into a single, highly-effective, answer page. The fusion can be score-based, in which the retrieval scores of documents are aggregated; or rank-based, in which documents are assigned a weighting based solely on their positions in the separate lists. In both cases, the new top-k result set is derived by re-sorting the documents after aggregate weightings are computed. Vogt and Cottrell [85] describe several effects that allow fusion to produce an effective response: taking advantage of diversity in document representation (*skimming*); building consensus among ranked lists (*chorus*); and catering for differences in quality of rankers (*dark horse*). Vogt [83] makes a case for fusion in web search based on parallelizing "fast but inaccurate IR systems", and then combining the lists to obtain results commensurate with a single high-performance system. Fusing the output of a one-shot query issued to many IR systems has also received attention – for example, Vogt [84] empirically shows that there is an implicit upper-bound of systems that should be fused before diminishing returns on effectiveness are experienced.

Query fusion is a source of information independent of systems, and has known to be effective for several decades. For example, in one early study, Belkin et al. [8] show that unsupervised fusion of related queries on the same system yields greater effectiveness than fusing one query issued to many better-performing systems. More recently, Moffat et al. [69] found that varying the queries of an information need retrieves at least the same diversity of relevant documents as does varying the systems carrying out the retrieval. Here we develop those ideas, and investigate the CPU time and

This work was accepted for publication in the ACM Transactions on Information Systems (TOIS), 2019.

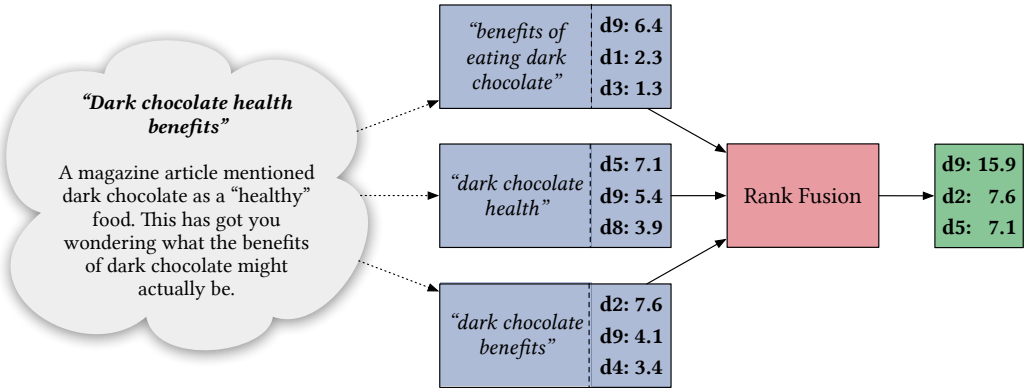

Fig. 1. The *query cluster* for the information need "dark chocolate health benefits", relating to the query `UQV100.034` [5]. Starting at the left, the cloud represents the user's information need, with possible query variants shown by dashed lines. A ranked list is then generated for each of those queries; those lists are combined (solid lines) to form a fused ranking (green box) for that set of queries, with that ranking referred to here as the *cluster centroid*. In this figure the CombSUM approach to fusion is assumed.

wall-clock time of query-based fusion in a web search environment. Processing query variations typically involves maintaining a heap data structure, with previous fusion techniques requiring one heap for each query variation. To efficiently fuse query variations, in Section 4 we introduce a fusion technique that only requires one heap, assuming that certain pre-conditions can be met.

In Section 5 we further reduce the cost of query-based fusion, and introduce a second new approach, this one based on re-ranking. The key idea is to amortize the cost of fusion, and use a pre-processing phase that computes *query centroids*, as illustrated in Figure 1, where a group of related queries are found using manual or automatic methods [11], and then their runs fused together into a single list [66, 77, 87]. Those centroid rankings are then held in a searchable cache. When a new query arrives, an association process identifies a matching cluster; at the same time, the query is processed to produce a top-k ranking. The centroid ranking and the query ranking are then fused, and a combined top-k ranking is computed. The goal is for the fused ranking be a consensus derived from all of the queries in the pre-processed cluster, together with the documents specifically identified by the query being handled.

As part of Section 5, we present several methods for combining the centroid ranking and the original query ranking. These include balanced interleaving between the query ranking and the centroid ranking, as occurs in online retrieval experiments [74]; carrying out a weighted CombSUM [35] between the two rankings; and employing a re-ranking approach in which the common-to-both documents are placed at the head of the result page, ordered by their position in the centroid.

Contributions. In particular, we:

- Investigate the wall-clock time of single-pass rank fusion and the parallel method described by Vogt [83];
- Propose a novel score-safe cost-effective single-pass rank fusion technique;
- Describe a novel query fusion architecture that efficiently re-ranks queries in an online setting; and
- Validate our results using the ClueWeb12B corpus and the publicly available query variations provided by the UQV100 test collection [5].

In what follows, Section 2 introduces related work; Section 3 describes the experimental setup; Section 4 explores a number of techniques for fusing multiple query variations *online*; Section 5 shows how the fusion can be computed *offline* and proposes various rankers that can utilize such pre-computed data; and Section 6 outlines some of the shortcomings of our investigation and describes possible future work.

2 BACKGROUND

2.1 Rank Fusion

It is well-known that combining the ranked retrieval outputs of query variations representing the same information need can improve retrieval effectiveness. For example, Belkin et al. [8] showed a representative sample of topic descriptions to ten experienced searchers, generating a pool of five Boolean queries for each topic. The outputs of the query variations were combined using an unweighted sum of retrieval scores. Those scores were computed using the INQUERY framework [81], which factors term weightings and the resulting list is not strictly Boolean. This aggregation of ranked-retrieval scores was later named CombSUM by Fox and Shaw [35], who investigated a range of rank fusion techniques. Bailey et al. [6] studied these phenomena further in their exploration of fusion and query variation consistency. Follow-on studies of query variation and rank fusion have shown that combining multiple individual rankings consistently boosts effectiveness [10–13].

Kozorovitsky and Kurland [48] showed that inter-document similarities can be used in a fusion framework to reward documents similar to those at the head of the result list. Other work has explored the role of fusion in diversification [53]; automatic generation of query perturbations [91]; the relationship between fusion and clustering [48]; and boosting tail query performance using supervised rank fusion [42]. No previous studies have explored the resource implications of these techniques in a large-scale search environment.

The use of supervised rank fusion with query variations has also been examined. Sheldon et al. [78] describe a supervised data fusion method named LambdaMerge, which optimizes a retrieval effectiveness metric based on fusion over user query reformulations and a wide range of document features. Lee et al. [52] extend the LambdaMerge framework from data fusion to collection fusion, where query-list features in a collection are averaged and used as query-vertical features. These approaches, as well as those of Huo et al. [42], are closely aligned with the work we present shortly. Other fusion techniques are also possible [3, 49].

The relationship between supervised fusion and learning-to-rank is an important issue in its own right, but one which is orthogonal to this work. Our approach here does not require supervised learning to achieve competitive results against strong baselines that were selected based on their well-established ability to efficiently and effectively rank documents in a web search environment. We will explore supervised approaches in future work.

2.2 Pseudo-Relevance Feedback

Documents. It has been known for a half century that appending terms from external sources to the user query can improve search effectiveness. Rocchio [76] suggests that users could submit queries using the vector space model, receive an initial list of documents in response to their query; then in a second step, examine those documents and run another query extended by the use of terms drawn from any documents identified as relevant. Additionally, the documents found by the user to not be relevant would have high-importance terms extracted and appended to the original query with a negative weighting, to lower the chances of non-relevant documents appearing in the ranked list.

To avoid the relevance assessment step, the top-R documents in the original query might be assumed to be relevant, with the top-E terms extracted from them appended to the original user query. Buckley et al. [18] note that this pseudo-relevance feedback approach was widely used in class projects at Cornell University in the 1980s, but that the origins of the approach were unknown. Pseudo-relevance feedback was also explored as an element of the third TREC conference [40]. In those experiments, large-scale relevance feedback, in which the top-30 documents of the initial retrieval had 500 terms extracted and appended to the user query, was shown to lead to an average 20% effectiveness improvement.

Language models [51, 73] can also be extended by pseudo-relevance feedback – referred to as relevance modeling. Working with TREC collections, Lavrenko and Allan [50] consider efficiency-related issues, finding that relevance modeling can be accomplished within acceptable real-time constraints if a pre-computed document similarity matrix is utilized. However, even on small collections, the required pre-computation is costly, and the technique is not suitable for web-scale operation. A range of other approaches to improve the efficiency of the longer second iteration of pseudo-relevance feedback have also been considered [11, 22, 89].

Queries. Relevance feedback approaches use documents as a source to add terms to the original query. As an alternative, in experiments using the TREC WT10g collection, Billerbeck et al. [14] show that forming query associations using the method described by Scholer and Williams [77] is more effective than using documents. Scholer and Williams [77] had found that a group of related queries can be formed by finding the set of top scoring queries for a document. This type of association has parallels in the initial stage of pseudo-relevance, and assumes that the high-scoring queries for a document are related, and hence can act as a surrogate for forming query associations when click-logs are not available. Terms are then selected from the associated queries to be added to the original query, using a selection formula due to Robertson and Walker [75]. Billerbeck and Zobel [15] explore the efficiency-effectiveness trade-offs compared to standard query expansion, and found that their approach is three-times faster than document-based query expansion on TREC 8 and TREC 10. There is a duality between this pseudo-relevance based approach and the single pass CombSUM approach that is described shortly in Section 4.1.

Even with these gains, the overhead of appending additional terms to a query can be high, and finding the right balance between efficiency and effectiveness remains an important practical challenge [50], especially on web-scale tasks. We revisit the cost of query expansion in on-line environments in Section 4, and then Section 5 shows how a pre-computed result list can improve effectiveness with negligible online processing overheads.

2.3 Efficient Index Traversal

The most commonly used structure for top-k document retrieval is the *inverted index*. Each unique term t is represented by a *postings list*, a sequence of document identifier/term-frequency $(d_{t,i}, f_{t,i})$ pairs, one for every document in which term t appears, where $d_{t,i}$ is the *docid* (document identifier) of the i th document containing t, and $f_{t,i}$ is the corresponding within-document term frequency. Inverted indexes provide efficient and scalable access to the necessary statistics for document ranking [93]. When a query is received, the postings lists associated with the query terms are fetched, and combined to rank and return the top-k documents.

The *index traversal strategy* – the way in which the postings lists are iterated – has a large impact on efficiency, with different postings layouts amenable to different traversal strategies. Here we employ a standard *document-ordered* index layout, and the corresponding *document-at-a-time* (DAAT) query processing strategies, plus dynamic pruning heuristics such as WAND [17] and more recent block-based variants (BMW) [23, 31, 32, 63]. These approaches tend to be more efficient

Table 1. The ClueWeb12B and UQV100 resources used. Note that the queries were stopped and Krovetz stemmed, reducing the number of distinct queries compared to that reported by Bailey et al. [5].

Documents	52,343,021
Topics	100
Total queries	10,835
Unique queries	4,175
Mean unique queries per topic	41.75
Hold-out queries	500

than the DaaT MaxScore and *term-at-a-time* (TaaT) approaches [80, 82], particularly for short queries, the most common scenario in web search. However, for long queries or large candidate sets, the case is less clear-cut [26, 34, 61]; moreover, fusion over query variations often leads to very long queries. That is, both WAND and MaxScore may have disadvantages as well as advantages, depending on the length of the query, the number of postings to process, the term selectivity, and a range of other factors.

2.4 Caching for Large-Scale Search

In order to meet *service level agreements* (SLAs), search engines must minimize unnecessary computation, with caching a common approach that increases query throughput at the cost of additional space consumption. Caches can be deployed at many levels of storage, including in-memory or on-disk [4, 86]. In IR there are two major approaches to caching. *List caching* involves storing commonly accessed postings lists in fast-access memory [21, 86]. For example, if the postings lists making up the index are stored on a SSD, a list caching strategy may opt to keep some subset of frequently-accessed postings in main-memory. Alternatively, *result caching* involves storing a query along with the relevant results (or some proxy thereof) that were returned for the query [33, 38]. In practice, both list and result caching are useful, and are deployed in tandem [4, 86]. Most caches utilize historical data such as static query logs or sliding windows of recent queries to build models of *what* to cache, and *when* to cache it, and can also be personalized on a per-user basis [57].

3 METHODOLOGY

Before providing details of the new techniques in Section 4 and 5, we first describe the experimental framework that is employed.

3.1 Hardware and Software

Our experiments are conducted on an idle Red Hat Enterprise Linux Server with 512 GiB of RAM and two Intel Xeon E5-2690 v4 CPUs, each with 14 physical cores. All algorithms were implemented with C++11 and compiled with GCC 7.3.1 using the highest optimization settings. Where multi-threading was used, up to 56 threads were spawned using the C++ STL threading libraries. All algorithms were implemented as components within the state-of-the-art VBMW code-base described by Mallia et al. [63][1], with our extensions also made publicly available (see Section 7).

3.2 Collections and Indexes

We conduct our experiments across the 52 million document ClueWeb12B corpus using the UQV100 query collection [5] and its 100 single-faceted topics derived from the multi-faceted TREC 2013 and 2014 Web Track topic descriptions. The UQV100 collection contains 10,835 query variations,

[1] https://github.com/rossanoventurini/Variable-BMW

sourced from crowd-workers who were presented with a narrative "backstory" for each topic, and asked to formulate a query in response. Bailey et al. [5] give details of the collection process and the queries that were collected. Moffat [67] explores further properties of this collection, and Moffat et al. [68] discuss the wider implications of query variations. Other ways in which query variations can be identified are discussed in Section 6.

To support the required experiments, we split the 10,835 UQV100 queries into two sets: a *training* set, used to build the query variation clusters, and a *testing*, or *hold-out* set, used to measure the final performance of the proposed approaches. The hold-out set was created by selecting five unique query variants per topic (that is, queries appearing only a single time in the UQV100 set), yielding a set of 500 queries across the 100 topics. Each hold-out query was drawn randomly from the corresponding topic's single-instance variations, without replacement. This hold-out approach differs from the simple hold-out method described by Fuhr [36], as all baselines and new techniques are evaluated against the total universe of topics, with at least five query impressions per-topic. The arrangement also avoids the limitations observed in other train-test splits in a single-query-per-topic evaluation scenario, where results are biased by the topic-effect of the generated split. The training set here represents the most commonly seen query variations for each topic, and hence can be regarded as being representative of what could be mined from logs in a production system. Table 1 summarizes the situation. Note that the number of distinct queries is less than in the underlying UQV collection because of our use of a Krovetz stemmer and a stop list.

We used Indri 5.11[2] to index the collection, and then converted the inverted index into the format expected by the VBMW code-base. Before building the VBMW index, we reordered the document identifier space using an open-source implementation [62][3] of the recursive graph bisection approach of Dhulipala et al. [30], as it has been shown to substantially improve index compression. The average block size of our VBMW index is approximately 40 integers per block, a result of binary searching for the parameter λ as discussed by Mallia et al. [63]. The final index was compressed using the Partitioned Elias-Fano mechanism [72].

3.3 Evaluation Metrics

Two different metrics are used to measure retrieval effectiveness: the recall-based NDCG approach [43] at a fixed cutoff depth of 10; and the utility-based RBP method [70] applied to full rankings (of length 1,000 documents), using a persistence of $\phi = 0.8$ and hence an expected viewing depth of five. All of the NDCG values were computed using `gdeval`[4]. These metrics and cutoffs were selected based on the judgment depth of the UQV100 collection [55], and result in relatively low RBP residuals arising.

Significance is computed using the Bonferroni-corrected paired t-test, and is denoted by † for $p < 0.05$, and by ‡ for $p < 0.001$. Significance was always tested with respect to the strongest available baseline.

4 REAL-TIME FUSION OF QUERY VARIANTS

This section considers whether the cost of computing query fusion immediately after queries are issued to an IR system is acceptable in a web-search scenario. Figure 2 introduces the components making up the query processing pipeline if no query clusters have been cached and the centroid cluster must be formed in real-time.

[2]https://www.lemurproject.org/indri.php
[3]https://github.com/pisa-engine/pisa
[4]http://trec.nist.gov/data/web/10/gdeval.pl

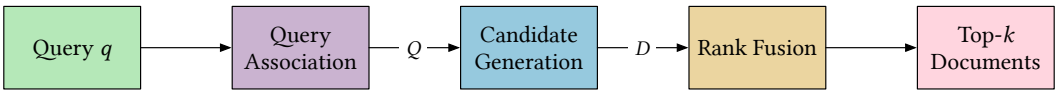

Fig. 2. Topology of an IR system implementing on-line query fusion. A query q is submitted by a user, and is associated with its related queries, the set Q. A set D of ranked document lists is formed from those queries, and passed to the rank fusion stage. Finally, a top-k document list representing the topic centroid is prepared and presented to the user.

4.1 Efficiently Processing Variations

Given a set of queries, including the one just entered by the user, the goal is to process them all, fuse their results, and return a single SERP (search engine result page), all the while noting that web-search systems typically impose strict per-query resource budgets [29, 44, 92].

Parallel Fusion (PF). The simplest approach is to spawn a process for each unique query variation in the cluster, and execute all of the queries in parallel. Once all threads have returned their top-k results, a rank-fusion algorithm is applied, to assemble the final SERP. This approach is viable provided there are sufficient CPU cores available, and requires that each thread operate to the same response-time requirement as the original query. If latency bounds are imposed by an SLA, queries running longer than a suitable time threshold can be terminated prematurely, and whatever partial results are available can be included during the fusion stage [44]. We measured three variants of the parallel fusion approach, denoted "PF-a", where a is a query processing strategy, one of VBMW, WAND, and MaxScore.

Single Pass DaaT. A drawback of the parallel approach is that many similar queries are processed concurrently, and hence that some of the corresponding postings lists are processed many times, without any commonality being exploited. An alternative is to perform all scoring operations in a single DaaT pass across the inverted index, concurrently building a top-k heap for each unique query variant. That is, an empty top-k heap is constructed for each unique query variant, and the postings lists for all terms are iterated in parallel, selecting as the pivot the minimum document ID across all of the cursors. At each processing step, all postings lists are advanced to align with the pivot, with variables tracking the current set of scores of the pivot document with respect to the terms appearing in each query. Once all aligned postings have been processed, the document scores are checked against the corresponding heaps, each of which is updated if necessary. Finally, when all postings cursors are exhausted, the set of heaps containing the top-k results for the query variations are be fused to create the required single SERP. We refer to this approach as "SP-Exhaustive".

Single Pass CombSUM. We now describe an efficient single-pass approach for computing the CombSUM [35] rank fusion score for a set of query variations, allowing improved efficiency through *dynamic pruning*. Given a set of ranked lists of documents, and a positive numeric score for each document in each list, the CombSUM score for a document d is the sum of the scores of d, computed over its appearances in the ranked lists. If there are ℓ lists, L_1 to L_ℓ, and the score of some document d in the i th of the lists is given by $s_{i,d}$ (with $s_{i,d} \equiv 0$ if $d \notin L_i$), then

$$\text{CombSUM}(d) = \sum_{i=1}^{\ell} s_{i,d} \,.$$

Now consider each of the component scores $s_{i,d}$, and the query q_i that led to it. If the scoring computation is an *additive* one, then

$$s_{i,d} = \sum_{t \in q_i} F(t,d)\,,$$

where $F(t,d)$ is the term-document score contribution associated with the term t in the document d according to the chosen retrieval model. Taking these together gives

$$\text{CombSUM}(d) = \sum_{i=1}^{\ell} \left(\sum_{t \in q_i} F(t,d) \right). \tag{1}$$

Now define $n_t \equiv |\{q_i \mid 1 \le i \le \ell \wedge t \in q_i\}|$, the number of input queries containing term t; and $Q \equiv \cup_{1 \le i \le \ell} q_i$, the union of the queries. Equation 1 can then be rewritten as

$$\text{CombSUM}(d) = \sum_{t \in Q} n_t \cdot F(t,d)\,, \tag{2}$$

making it clear that for additive similarity scoring mechanisms, the CombSUM score for a set of query variations can be computed by forming the union Q of the queries, counting term frequencies n_t across the variations, and then evaluating a single *super query* against the index of the collection by taking a linear sum of the individual contributions of $F(t,d)$. This requires that all of the scores are positive. Similarity models that are not additive, or that yield negative scores, may not be used in this way. Note that scores can be added without requiring any normalization step [35], since the component rankings were formed using a consistent weighting scheme. Indeed, normalization would require evaluating each query variation independently, defeating the benefit achieved via Equations 1 and 2. That is, one pass fusion would not be possible as shown here as the entire result lists for each query to a depth k would be required *before* normalization can be applied. Hence, in the following experiments, we use the BM25 similarity model, and do not normalize the individual ranking scores. In practice, normalization generally benefits fusion of different ranking algorithms more than the case explored here, where BM25 scores are term-wise independently distributed. Any other additive similarity models can be applied in a similar manner, such as PL2 divergence from randomness (DFR) [2]. We focus on BM25 as one well-known and widely-used exemplar.

Computing CombSUM in a single-pass, rather than by fusing runs of length k generated independently, brings a rather unexpected benefit – it allows the generation of a more precise score-safe ranking. This occurs because when a traditional fusion scenario is applied to separate lists, each truncated at k documents, there is no contribution made from the documents at positions $k+1$ and deeper – they are ignored (and inferred to have zero scores) when computing CombSUM. But those documents might still appear in the top-k of the fused list, in which case the ignored contributions might perturb the final ranking. In contrast, the one-pass method computes a correct top-k fused list as if the component lists had all been scored to an arbitrary depth.

In order to employ Equation 2, and apply dynamic pruning to the super query, the index traversal process must be slightly modified, with the upper-bound scores, U_t, also multiplied by n_t. For the block-based VBMW approach, the n_t multiplier to each block-max score, $U_{b,t}$, must also be applied as each block is encountered. Query processing does not differ in any other way. This approach is generalizable to all safe-to-k dynamic pruning traversal strategies; and hence we again test three variations, denoted "SP-CS-a", with a one of VBMW, WAND, and MaxScore. Computing CombSUM over query variations draws an interesting parallel with the Assoc-Assoc approach of Billerbeck et al. [14]. Instead of weighting terms by the Robertson and Walker [75] term-selection value formula, the term weightings are linearly scaled by the frequency of their occurrences in the related query set.

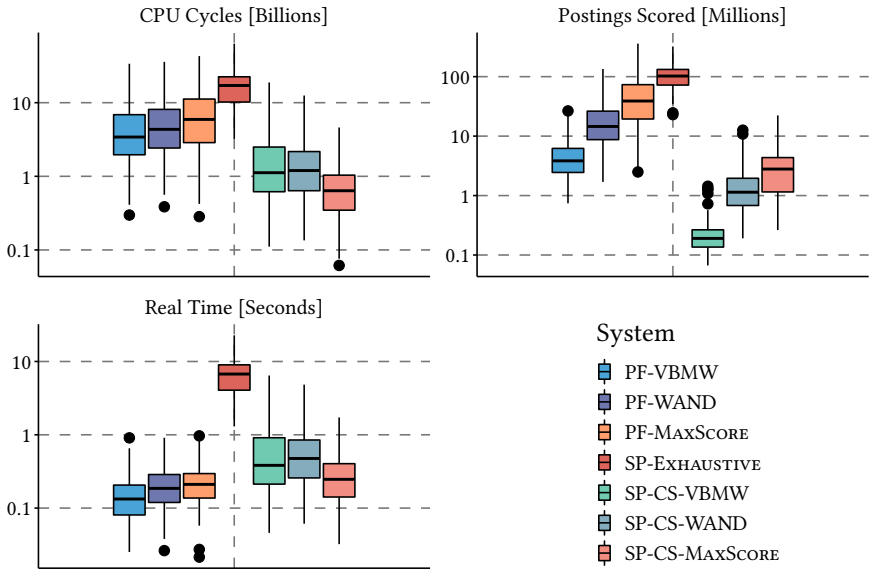

Fig. 3. Efficiency of rank-fusion algorithms, assuming that the starting point is a set of query variations. The panes show total cost in CPU cycles; total cost in terms of postings processed; and query latency.

4.2 Experiment: Real-Time Fusion

To test these approaches, we take all query variations for each UQV100 topic, and measure the cost of computing a fused top-100 result list, covering an average of 42 query variations per topic, each computed to depth $k = 1{,}000$, and then (except in the case of the SP-CS approaches) fusing the results. Three indicators are reported: the number of CPU cycles consumed; the number of postings scored; and response latency. In the case of the parallel approaches, the first two are summed over all threads. Note that the CPU measurements ignore the slight processing overhead generated by the forking and locking activities inherent in parallel execution.

Figure 3 shows the results. The first (top left) pane shows the total CPU time required. The three SP-CS methods are the most efficient in terms of processing cost, with the SP-CS-MaxScore approach slightly better than the other two. In the second pane (top right), the SP-CS-MaxScore implementation processes more postings on average than either the SP-CS-WAND and SP-CS-VBMW approaches – the latter two reduce the number of postings, at the cost of more non-posting processing. Finally, the third pane (bottom left) shows elapsed wall-clock time. The three PF approaches are the fastest, due to their extensive use of parallelism. Even though each query's latency is dictated by the slowest-running variation, execution on average is fast; and the use of suitable aggregation policies [92] can improve the *tail-latency* [60] of such approaches. Note, however, that this speed comes at a resource cost, as shown in the first pane.

All of the methods could have their latency (but not their aggregate workload) reduced by splitting the collection across a greater number of processors, and if the goal is to produce fused query result lists in real-time, then highly parallel implementations of any of the approaches are necessary. On the other hand, if the goal is to reduce CPU cycles, the SP-CS methods are clearly superior, with SP-CS-MaxScore having the least cost. Mallia et al. [63] concluded that the VBMW

Table 2. Fusion effectiveness as the number of query variants is increased. Variants are selected from the query clusters at random, with replacement; measured as averages over a set of ten such sequences. The values in parentheses are RBP residuals, recording the maximum extent of the RBP score uncertainty. Values with a shaded background represent the most effective configuration over that metric.

Num. variants	NDCG@10	RBP $\phi = 0.8$
1	0.182	0.426 (+0.067)
2	0.210	0.469 (+0.081)
5	0.236	0.514 (+0.043)
10	0.254	0.537 (+0.028)
20	0.256	0.540 (+0.021)
50	0.261	0.542 (+0.018)
100	0.262	0.550 (+0.017)

method is the best choice for disjunctive query processing; however the results in Figure 3 confirm that the MaxScore mechanism outperforms both WAND and VBMW when the result size is large, and when queries are long [64].

4.3 How Many Queries Should Be Fused?

Another way of reducing processing costs is to fuse fewer queries. To determine the impact that the number of variants has on processing costs, the set of all variations for a topic, including duplicates, were sampled with replacement to generate a stream of queries of the required length, thereby creating a plausible query stream for all topics, including those in which fewer than 100 crowd-workers generated a query. Those samples were then executed and fused, recording the cost in CPU cycles and the effectiveness of the final SERP. The selection process was carried out incrementally, with one variation drawn and measured, then a second added to it and measured, and so on; with that entire sequence repeated ten times, and the values recorded being the averages over those ten runs. Table 2 and Figure 4 show the results.

Table 2 makes it clear that adding variants increases effectiveness, but that the gains diminish as more variants are added. This outcome is at least partially a consequence of the methodology used, since "with replacement" means that there is an increasing probability of a previously-selected query being drawn again. Bailey et al. [6] also observe that, in general, adding more distinct variants improve the effectiveness of rank fusion, irrespective of metric or fusion mechanism.

Figure 4 shows that SP-CS-MaxScore retains its computational advantage across the range of variation set sizes, and that all of the SP-CS approaches are cheaper than the SP-Exhaustive and PF methods. In particular, the cost of the SP-CS approaches is largely determined by the number of unique *terms* that are required for processing, whereas the other methods are more closely tied to the number of unique *queries*. Since query variations often contain similar terms, the SP-CS approaches scale better.

4.4 Discussion

Three approaches for efficiently fusing a set of query variations have been described and measured. The SP-Exhaustive and PF approaches can be used with any rank-fusion algorithm, but are more costly than the SP-CS approaches, which are based on CombSUM and additive functions such as BM25. The PF approaches use nearly an order of magnitude more resources than the SP-CS methods, a notable disadvantage that more than offsets the ease with which they can be implemented across a cluster of processors. Also worth noting is that all of the methods might gain benefit from a

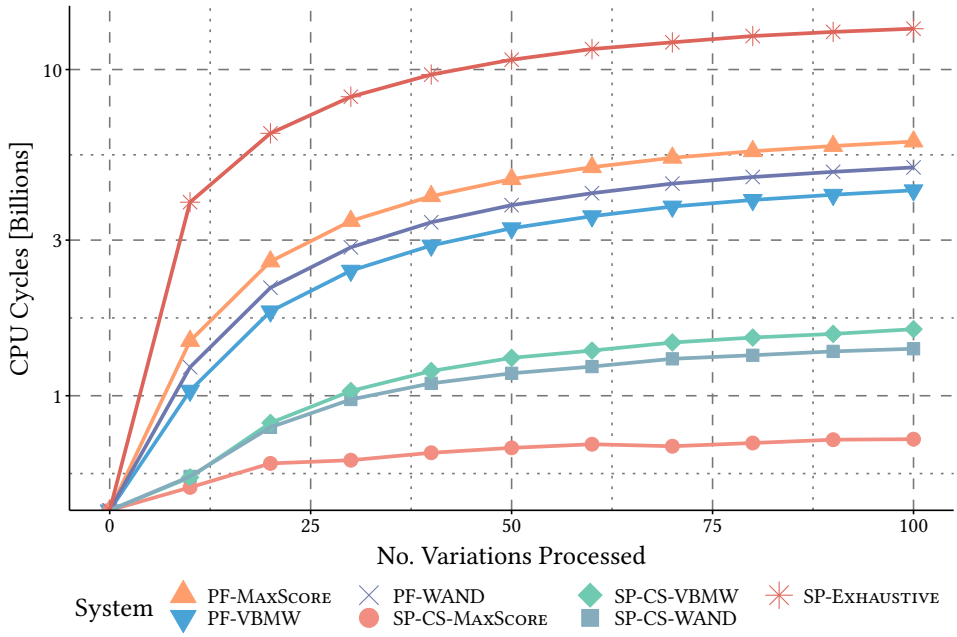

Fig. 4. Average per-topic cost in CPU cycles of approaches for generating fused rankings, as a function of the number of query variations being fused. Sampling is random from each cluster, with replacement, averaged over ten independent sequences. All of these methods obtain the same effectiveness, as listed in Table 2. Note the logarithmic vertical scale.

selective search framework [39, 45, 46], an option outside the scope of our investigation here. That is, rank fusion across query variations can be computed such that latency is small enough for online use, but only if substantial total computational resources are available. Better techniques are required if the overall resource cost must also be managed carefully.

This section has focused on fusion using the CombSUM method. There are other fusion techniques that can also be applied, some of which are easier than others to transform into single-pass implementations. Benham and Culpepper [10] compare the effectiveness of CombSUM with six other unsupervised fusion techniques on the ClueWeb12B corpus using the UQV100 query set, and demonstrate that CombSUM is competitive with the CombMNZ [35], RBC [6], and RRF [25] approaches.

5 USING PRE-COMPUTED CENTROIDS

The previous section supposed that the query fusion was to be carried out on-the-fly, and introduced a reduced-cost query fusion technique that in essence formed a super query that was evaluated. The risk of that approach is that if service level agreements governing response time and throughput must be complied with, on-the-fly query fusion may not be a viable approach. The key issue then becomes: is it possible to compute fused lists in an offline manner, and use those cached intermediate results to boost online query performance? To address this, we now consider the pros and cons of pre-computing centroid rankings offline, and then, when a query requires processing, efficiently folding the pre-computed results list and the query's result list into a single ranking. An important consideration that then arises is the quality of the query-cluster matching process, and our results include measuring effectiveness assuming both perfect and imperfect cluster identification.

5.1 Query Centroids

Fused centroid lists can be pre-computed using the methods described in Section 4, based on pseudo-documents formed from the union of the terms in each cluster. Since this is a pre-computation, *total resource requirement* is the appropriate cost measure, and not latency; with the cost equation further moderated by the expectation that the pre-processing time can be amortized over multiple subsequent queries that refer to that cluster. Hence, any desired fusion technique can be used, with no restriction to BM25 and/or CombSUM.

If query centroid data in the form of indexed pseudo-documents and consensus fused rankings are stored in main memory or on SSD, it can be searched and retrieved quickly as queries are processed. In the experiments described shortly, the fused query consensus rankings (the cluster ranking *centroids*) are stored in main memory, using an array that preserves the ranked order of the fused set; with document identifiers also maintained in an $O(1)$ average access time hash table. With these structures, around 15.6 kiB per topic was required on average to store a ranked list containing $k = 1{,}000$ documents (centroid lists were always computed and stored to a length of 1,000 documents). That space requirement can be reduced to 7.8 kiB per topic if the subsequent fusion is restricted to methods that are rank-based, where knowledge of document identifiers and ranks alone is sufficient without document scores.

For all experiments described in this section, query centroids were constructed by fusing the "held-in" training queries for that topic, again using CombSUM. This measurement of retrieval effectiveness changes on the held-out queries for each topic. Again, note that any desired fusion technique can be used in this stage of the computation.

To establish an upper-bound of the effectiveness of centroid rankings, we suppose first that the *cluster association* stage in the retrieval pipeline provides perfect matching between the held-out queries and the corresponding clusters. Assuming that perfect matching is possible, the simplest way to use pre-computed centroids to provide effective search results is to return the centroid without even running the query that triggered the association. We call this oracle approach RCC, for "return cluster centroid".

For effectiveness scores to be meaningfully compared, they must be considered relative to a sample of systems known to work well for the application domain – here being real-time web search. The baselines used in this section include two proximity-based algorithms and one learning-to-rank model. The first is the L2P approach proposed by Lu et al. [56], a bigram ranking model that linearly combines the BM25 score of a document with sequential bigrams. Unlike other term dependency models, L2P does not require global statistics for term dependencies, greatly improving overall efficiency. We also use a field-weighted variation of the sequential dependency model (SDM) [65] that operates across the document body, document title, and inlink text. The ClueWeb09B collection was employed in conjunction with the TREC 2009–2012 Web Track topics to tune parameters in both of these approaches. As a learning-to-rank baseline, denoted "LTR", the LightGBM[5] framework was used to build a LambdaRank model [19]. Instead of training on just the hold-out set, we train our model on both the topics in the hold-out set as well as the most common five variants from each topic. This additional exposure leads to a more robust model. We used ten-fold cross-validation to train and test the LtR model across a large set of features. The features were generated using the publicly available system described by Chen et al. [24], Gallagher et al. [37][6] and a description of the features can be found within that codebase. We refer the interested reader to Liu [54] and to Macdonald et al. [58, 59] for further information on building a competitive LtR system.

[5]https://github.com/Microsoft/LightGBM
[6]https://github.com/rmit-ir/tesserae

Table 3. Effectiveness of online fusion measured using two effectiveness metrics, and compared to three baselines. Significance is measured with respect to LTR, the strongest of those baselines. The scores listed in the "Mean" column are averages across the 500 held-out query variations, five for each of the 100 topics; the "W/T/L" numbers are the respective counts of those 500 queries for which the method exceeds the BM25 baseline by more than 10%, is within 10% of the baseline, and is less than 10% of the baseline. Values with a shaded background represent the most effective configuration over the specified measure in the column header, for each metric.

Method	NDCG@10	W/T/L	RBP $\phi = 0.8$	W/T/L
User Query (BM25)	$0.170^{\ddagger}$	—	$0.401^{\dagger}$ (+0.088)	—
L2P	$0.169^{\ddagger}$	160/236/104	$0.400^{\dagger}$ (+0.089)	136/264/100
SDM+FIELDS	$0.180^{\dagger}$	247/76/177	0.417 (+0.153)	215/117/168
LTR	0.200	281/53/166	0.435 (+0.205)	258/71/171
RCC	$0.263^{\ddagger}$	354/54/92	$0.553^{\ddagger}$ (+0.011)	311/101/88

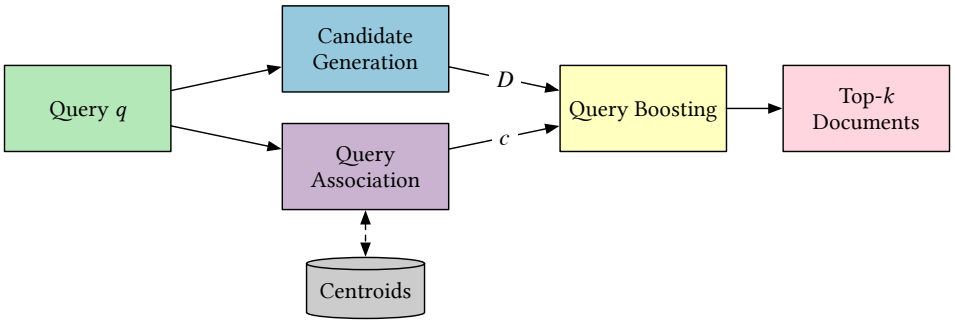

Fig. 5. Modifying the processing pipeline described in Figure 2, to leverage pre-computed query centroids. The user enters a query q, which is both evaluated against the document collection to create a top-k list D, and also used to search the set of pre-computed query centroids. If a match is found in the cache, centroid c and the top-k document list D are combined by the *query boosting* method, yielding a fused top-k result page.

Table 3 lists the effectiveness of the baseline systems across the held-out queries. The effectiveness of the Return Cluster Centroid (RCC) method is substantially better than any of the baselines, but it is based on unrealistic assumptions that are considered in detail below. The RCC approach also performs better than BM25 on the greatest number of the 500 test queries, and loses to BM25 the least. Of the non-centroid baselines considered, LTR is the most effective.

Table 3 provides the motivation for the question considered in this section: given that correctly-identified cluster centroids perform well even in connection with queries not included at the time the cluster centroid was formed, can they be employed in a practical search system in which cluster association may not be 100%? In particular, if an incorrect centroid ranking is returned, the likely result will be significant user dissatisfaction. To mitigate that risk, the user query and centroid ranking should instead be combined in some way, to ensure that documents responsive to the actual user query appear in the ranking.

5.2 Query Boosting Using Centroids

Figure 5, which can be compared with Figure 2, shows the proposed processing pipeline. Each incoming query is both executed against the collection (the step "candidate generation") and

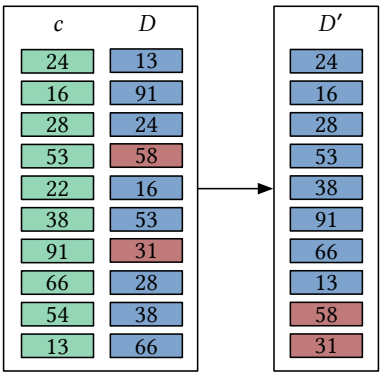

Fig. 6. Reference list re-ranking, where c is the k-document pre-computed centroid ranking, D is the k-document SERP generated for the user query, and D' is their k-document fused result, in this example with $k = 10$ for both. Documents in green are from the centroid list, documents in blue are common to it and the query list, and the documents in red are the ones from D that were not in c.

also checked against the set of pre-computed centroids. If an association is identified, the query boosting step combines the centroid ranking with the query's SERP to create a fused top-k list. If no association is found, the query's SERP is returned without further modification.

Three approaches to joining the centroid ranking and a query ranking were considered. While not an exhaustive list of possibilities, these methods demonstrate that improvements in retrieval effectiveness are achievable using relatively simple and inexpensive techniques.

Plain Interleaving. The first boosting approach is inspired by online retrieval experiments in which results from two different systems are presented as a single list. In a balanced interleave, elements are chosen in alternating fashion from two results, with the first chosen randomly. Here we start from the query SERP D, to ensure that the highest ranked document for the query is displayed first. At each stage thereafter the next highest ranked document not already included in the output list is taken in an alternating manner from one of the two lists, and added to the fused run. We denote this approach as "Interleave". A possible benefit of the interleaving model is its connection to A/B testing, where click logs may be helpful in deciding whether to preference the user query or the fused result centroid.

Linear Combination. A second option is to adopt the approach proposed by Vogt and Cottrell [85], and compute a per-document weighted sum of the min-max re-scaled centroid set and the re-scaled user query answer set, then use those scores as a descending-order sort key to form the SERP. To implement this approach we applied a weight of δ to the query centroid and $(1 - \delta)$ to the user query, and set δ to 0.5 as a starting point. When $\delta > 0.5$, the method approaches the bias exhibited in the Interleave method. Unlike the Interleave method, a linear combination does not guarantee an equal contribution from either the cluster centroid or the documents retrieved in response to the actual query. This is because inclusion in the final results list is sensitive to both the union of the two sets of documents, and also to the scores of those documents. We denote this approach as "LC".

Reference List Re-Ranking. The third approach intersects the query's ranking and the centroid ranking, adopting the ordering supplied by the centroid run c. Any remaining documents from the query's run D are placed after the intersection set, in the same order as they appear in D. Both the

Table 4. Mean query boosting time, total query execution time (milliseconds per query), and percentage overhead attributable to boosting when retrieving the top k documents for user queries computed with BM25. The query centroid rankings were always 1,000 documents long. Values with a shaded background represent the fastest boosting method for that value of k.

Length	Method	Boosting	Total	Overhead
$k = 10$	Interleave	0.072	15.061	+0.5%
	LC	0.158	15.146	+1.0%
	Ref-Reorder	0.027	15.015	+0.2%
$k = 100$	Interleave	0.072	23.729	+0.3%
	LC	0.164	23.821	+0.7%
	Ref-Reorder	0.041	23.698	+0.2%
$k = 1{,}000$	Interleave	0.129	44.358	+0.3%
	LC	0.258	44.487	+0.6%
	Ref-Reorder	0.179	44.408	+0.4%

centroid ranking order and the query's selection of documents are respected, with the documents in the intersection listed first. We denote this approach as "Ref-Reorder". Figure 6 provides an example in which two runs of length $k = 10$ are joined, favoring the documents that appear in both, and accepting the ordering of the consensus ranking. (In the experiments described shortly the centroid list was always 1,000 documents long, and only the length of c was varied as part of the experiment.)

Boosting Time. Table 4 provides boosting times for BM25 queries using the proposed methods, as well as the end-to-end time and the percentage overhead relative the original BM25 query. All three approaches have sub-millisecond performance. For the two smaller values of k the Ref-Reorder approach is the fastest; for $k = 1{,}000$ the Interleave approach is better. But in all cases the end-to-end latency from query submission to final top-k is dominated by the query's BM25 stage, and none of the boosting approaches add more than 1% overhead to the cost of generating the final SERP.

5.3 Query Boosting Effectiveness With Perfect Cluster Association

Having demonstrated that the computational cost of query boosting is very small, we now examine effectiveness, first considering the (almost certainly unattainable) situation in which cluster association is assumed to be infallible.

Table 5 lists effectiveness scores for the three centroid-based fusion methods, and compares their performance to the baseline approaches listed in Table 3, averaging across the 500 single-instance held-out queries. The number of wins, ties, and losses compared to the BM25 evaluation of the same 500 queries is also reported. The LC approach listed in Table 5 employed a parameter δ of 0.5, placing equal weight on the two rankings being combined. All boosting approaches significantly outperform the LtR baseline for both metrics; of the three, the Ref-Reorder mechanism is the most effective. The Interleave approach results in only 87 score degradations compared to NDCG, and 63 relative to RBP. The LC approach also demonstrates a good effectiveness profile, with 84 and 80 queries performing worse than the user query for NDCG and RBP, respectively. Note also that the RBP residuals are relatively small, indicating that the judgments are a good fit for the runs being scored.

Table 5. Effectiveness of centroid boosting measured using two effectiveness metrics. Significance is measured with respect to LtR, the strongest of the non-centroid baselines listed in Table 3. The scores listed in the "Mean" column are averages across the 500 held-out query variations, five for each of the 100 topics; the "W/T/L" numbers are the respective counts of those 500 queries for which the method exceeds the BM25 baseline by more than 10%, is within 10% of the baseline, and is less than 10% of the baseline. Values with a shaded background represent the most effective configuration over the specified measure in the column header, for each metric.

Method	NDCG@10	W/T/L	RBP $\phi = 0.8$	W/T/L
Interleave	$0.223^{\dagger}$	305/108/87	$0.486^{\ddagger}$ (+0.037)	287/150/63
LC	$0.231^{\ddagger}$	322/94/84	$0.504^{\ddagger}$ (+0.033)	290/130/80
Ref-Reorder	$0.243^{\ddagger}$	345/62/93	$0.527^{\ddagger}$ (+0.053)	304/95/101

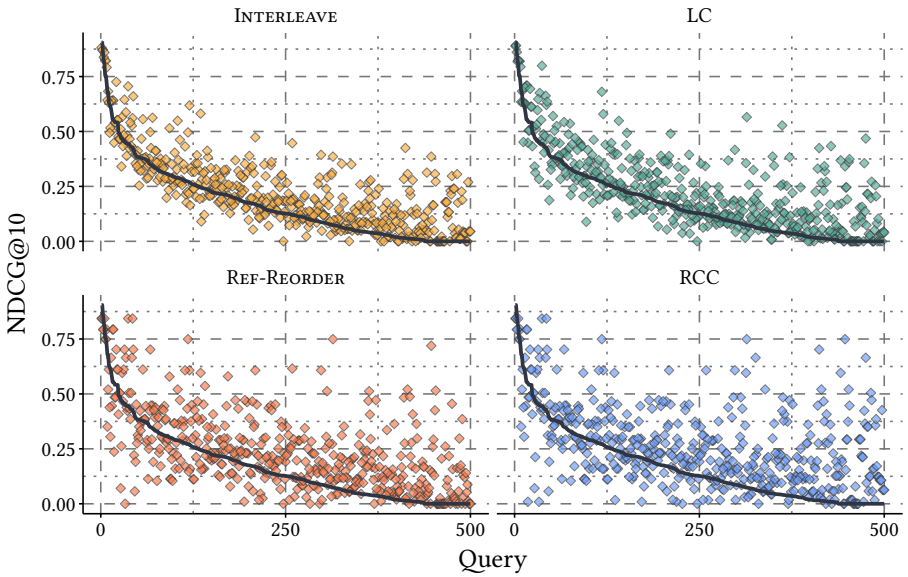

Fig. 7. Difference between NDCG@10 effectiveness score for the baseline BM25 query and the four query boosting approaches. The 500 held-out queries are ordered by their BM25 scores (the black line in each graph) and the corresponding scores from each of the different evaluation approaches are shown, one set in each pane.

To provide another perspective, Figure 7 shows the extent to which the query-by-query NDCG@10 scores shift up and down compared to the baseline BM25 ranking. The four panels represent effectiveness profiles for the three boosting techniques, plus the RCC method. The same decreasing-score ordering of queries is used in each of the panes, with the BM25 scores shown as a black line, and the paired scores for the corresponding boosting method shown as colored dots. The Interleave and LC methods have smaller dispersions than the RCC and Ref-Reorder methods; they sometimes improve substantially on the baseline score, but are also vulnerable to notable score degradations. A similar pattern of performance was observed when the same plots were generated for RBP.

If the cluster association process is infallible, the three query boosting methods are less effective than simply returning the centroid list. They are, however, still significantly more effective than LtR. We now turn to the more plausible case in which query cluster association is imperfect, and ask if the same relativities continue to hold.

5.4 Query Boosting Effectiveness With Fallible Cluster Association

Pre-computed centroid methods work well if queries can be correctly categorized against a previously-identified information need. But in a real system a reasonable percentage of incoming queries can be expected to differ from all known clusters, or worse, to be matched against the wrong cluster. How should queries that do not match any cluster be handled? And how can the impact of failed query matchings be minimized?

Centroid Identification. We first ask whether incoming queries can be reliably mapped to an existing query cluster. Wen et al. [88] show that combining query keywords and cross-reference similarity using document hierarchies from click-through data can give a precision and recall of approximately 95% in an experiment involving 20,000 query clusters. A 5% error rate is encouraging, given that precision can be traded against recall, to guard against the potential impact of incorrect cluster matches.

To replicate that evaluation, we built an index of *pseudo-documents*, each containing the union of the terms in the queries of one of the 100 UQV100 topics. Another 10,000 synthetic query clusters were formed from the anchor text of inlinks to Wikipedia articles in the ClueWeb09 corpus, adopting the approach of Dang and Croft [28], who showed that anchor text can be used as a reliable substitute for user queries. Each pseudo-document contained the union of the query terms (rather than allowing duplicates) so as to not bias cluster selection. Upon receiving a query, an index of the pseudo-documents was searched using BM25 to find the top-scoring pseudo-document. A range of policies for forming synthetic centroids from inlink data were employed to aid with pre-processing, sanitization, and diversity; code for these steps and a manually curated stop-list are available in our code-base (see Section 7). Across the 10,000 Wikipedia clusters, 221,989 queries were used (41,624 unique); with the largest and smallest clusters containing 442 and 11 queries respectively. Finally, 0, 100, 1,000, 5,000 and 10,000 "distractor" clusters were added to the 100 UQV100 clusters, to measure query association success rates using the 500 held-out UQV queries, where success consisted of selecting the correct UQV100 cluster in the presence of the distractors. These success rates were 97%, 97%, 94%, 91%, and 89% respectively across the range of distractor sizes, broadly in line with the results of Wen et al. [88].

Incorrect Centroid Association. The results in Table 5 assumed perfect cluster identification. To quantify the effect of incorrect matchings, we ran experiments in which each query was assigned to the wrong cluster with varying probabilities, so that the failure profiles of the proposed approaches can be compared in the presence of more realistic centroid matching. We denote the error rate as ϵ, and explore values of $\epsilon = 0.05$ (that is, with 5% of queries assigned to the wrong cluster, the rate attained by Wen et al. [88]) and a rather pessimistic $\epsilon = 0.2$. The latter is worse than the rate observed in the experiments with 10,000 distractor centroids reported above.

Table 6 shows the NDCG and RBP effectiveness scores attained in this evaluation scenario, as well as their respective wins and losses relative to the BM25 baseline. When $\epsilon = 0.0$, the perfect matching scenario, RCC (repeated from Table 3) provides the best effectiveness. But that advantage is eroded when errors are introduced, showing that RCC is fragile when faced with erroneous cluster matching. All of the approaches degrade as ϵ increases, but RCC decreases the most. Indeed, if a search engine receives an exact match well-known query, it can simply return a cached result,

Table 6. Effectiveness of online fusion approaches, where ϵ represents the rate in which incorrect clusters are selected. Significance is measured with respect to LtR, the strongest of the four baselines that were employed. Return Cluster Centroid (RCC) is tabulated with a perfect cluster matching rate of $\epsilon = 0.0$ as an initial reference point. For each effectiveness value listed for each ϵ, ten trials of randomly assigning an erroneous cluster at the respective error rate were computed, where the mean score was found for each query. The scores listed in the "Mean" column are averages across the 500 held-out query variations, five for each of the 100 topics; the "W/T/L" numbers are the respective counts of those 500 queries for which the method exceeds the BM25 baseline by more than 10%, is within 10% of the baseline, and is less than 10% of the baseline. Values with a shaded background represent the most effective boosting technique over the specified measure in that column, for each error rate considered.

Error Rate	Method	NDCG@10	W/T/L	RBP $\phi = 0.8$	W/T/L
$\epsilon = 0.0$	RCC	0.263	354/54/92	0.553 (+0.011)	311/101/88
$\epsilon = 0.05$	RCC	$0.249^{\ddagger}$	334/63/103	$0.523^{\ddagger}$ (+0.063)	289/101/110
	Interleave	$0.217^{\dagger}$	298/108/94	$0.472^{\ddagger}$ (+0.061)	269/163/68
	LC	$0.225^{\dagger}$	309/103/88	$0.489^{\ddagger}$ (+0.059)	271/141/88
	Ref-Reorder	$0.240^{\ddagger}$	343/66/91	$0.519^{\ddagger}$ (+0.056)	301/100/99
$\epsilon = 0.2$	RCC	0.209	278/63/159	0.441 (+0.215)	235/78/187
	Interleave	0.199	277/94/129	0.435 (+0.128)	222/170/108
	LC	0.203	275/95/130	0.444 (+0.137)	218/144/138
	Ref-Reorder	0.212	339/75/86	$0.498^{\ddagger}$ (+0.061)	290/119/91

which is what is commonly done in practice [4, 21, 33, 38, 57, 86]; what we are exploring here is the consequence of also allowing approximate matches to be exploited.

Overall, Ref-Reorder is the most robust technique, and remains significantly more effective than LtR, even when 5% of incoming queries are assigned to the wrong cluster. In the two lower sections of the table, with the error rate increased to $\epsilon = 0.2$, Ref-Reorder remains statistically significantly better than the LtR baseline. In contrast to the results of Table 5, this method now has the most wins *and* the least number of losses of any of the online fusion approaches considered on both evaluation metrics. Observe also that the higher error rate leads to increased RBP residuals, but with the Ref-Reorder average residual increasing by least.

Figure 8 extends Table 6 and shows the shape of the effectiveness decay as a function of error rate. Even with an unrealistically large error rate of $\epsilon = 0.5$ the NDCG score for Ref-Reorder is 0.205, and RBP is 0.462. That is, even with a very high error rate in terms of cluster identification, retrieval effectiveness remains close to the original BM25 run prior to the application of Ref-Reorder boosting (Table 3), making this an appealing approach for robust retrieval. Figure 8 also shows that our goal of safely boosting query performance has been met using all approaches outlined in Section 5.2, since the slope of the decay curve is steeper for RCC than any other approach. Returning the centroid list is at face value an attractive option, but is not resilient in the presence of errors.

6 DISCUSSION

We have presented three novel query effectiveness boosting techniques, built around the idea (Figure 5) of pre-computed query cluster centroids. We now consider some of the issues associated with this proposal, and a range of avenues for possible extension.

Fig. 8. The effectiveness of each query boosting technique listed in Section 5.2 as a function of clustering error. Like Table 6, the values reported are an average of ten trials for the respective error rate. All query boosting methods are more resilient to query association error than RCC which returns the centroid list. The thick-black horizontal line indicates the LtR score, and the gray line represents the User Query (BM25) score; both baselines correspond to Table 3.

Improving Real-Time Rank Fusion. Although the real-time rank fusion approaches outlined in Section 4 can be implemented in a low-latency manner, they nevertheless require substantial amounts of computation. For this reason, we opted to build query centroids offline using a cost-effective rank-fusion approach based on CombSUM (Section 5), and then deploy online boosting approaches that make use of those centroids. It would be interesting to further develop these approaches to make them more scalable and less resource intensive. One possibility would be to employ large-scale distributed architectures, with parallel fusion conducted across multiple index server nodes and multiple (perhaps even all) clusters at a time. As a motivation, recent work has shown that cost-effective supervised rank fusion techniques can outperform state-of-the-art learning-to-rank models [71].

Caching and Centroids. There has been a large volume of previous work on improving efficiency through caching in large-scale search engines [4, 21, 33, 86]. While the approach proposed here is essentially an alternative form of caching, it would be interesting to further examine the relationship between traditional caching techniques such as result caching, and fusion-based centroids. For example, deciding when to build new centroids, drop stale centroids, or update the results within a centroid, might all be interesting questions. Centroid-based approaches could also be applied in situations where the search engine is under a high load [16]. Given that they are computationally cheap, they would likely be suitable as *fallback* query processing approaches, especially when more expensive methods would violate latency SLAs.

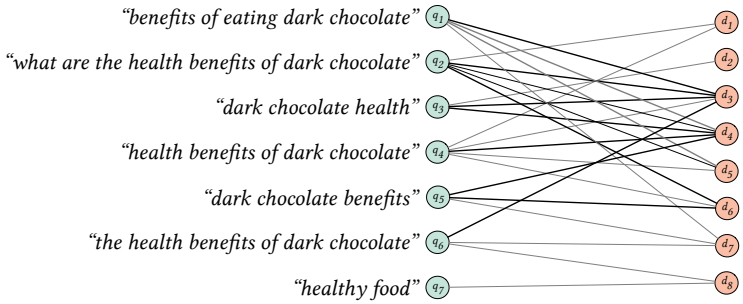

Fig. 9. A bipartite click-graph, showing the associations of document clicks from queries. The thickness of each line represents the frequency of clicks for that query and document pair.

Rank-Based Fusion. While score-based rank fusion techniques are effective, rank based fusion approaches such as RBC [6] and RRF [25] might also be of interest. Knowledge of the final rank order of documents seems like a key requirement for these methods; nevertheless, there may be efficient techniques possible for computing a final fused ranking in a single pass through the index. Nor is it obvious how rank-based fusion might be used to dynamically prune queries processed in parallel.

Simulating Query Intent and Data Privacy. To demonstrate our results in a laboratory setting, we used the UQV100 test collection, in which the query clustering is a direct consequence of the data collection process. In order to measure the impact of incorrect cluster identification, we then also carried out a failure analysis (Section 5.4). Future experimentation might verify the reliability and robustness of the approaches we propose, including whether it is helpful to form centroids with respect to the current step in a user information foraging activity; and how effective the new techniques are outside of test-collection settings.

To validate our observation against a full-scale search engine, query logs and click-graphs would be required to form these clusters using automatic methods. That data is not currently publicly available, and perhaps never will be after the concerns raised in connection with Cambridge Analytica [20] and the earlier AOL log release [7]. Without such data, academics are constrained to exploring performance improvements using data that is publicly available, as we have done here. The upside of using public resources such as the UQV100 queries – no matter how limited they may be in scope – is that the experiments are reproducible, and do not rely on access to private information that may have been gathered from users without adequate consent being given. Fortunately, previous research from industry research labs has shown that query clustering by intent is not only possible, but that it is being used in several different contexts, which we discuss now.

Forming Query Centroids Automatically. A range of authors have carried out experiments in connection with grouping queries by intent (that is, by information need). Relevant applications include query rewriting [9, 11, 41], and forming query clusters using query logs and click-graphs [27, 47, 79]. For example, web document click-graphs can be used to generate "virtual" queries by associating documents that are semantically close on the click-graph [90]. This technique was used to create gating features for query variations in LambdaMerge [78]. Craswell and Szummer [27] further demonstrated that random walks on a click graph can form effective query clusters in the domain of image search, noting that this is perhaps not unsurprising, given the prior work of Xue et al. [90].

Similarly, Kong et al. [47] note that:

> *Weighted bi-graph clustering capitalizes on organic search results to construct a bipartite graph with a set of queries and a set of URLs as nodes. Edge weights of the graph are computed with the impression and click data of (query, URL) pairs from a Bayesian perspective and are used to induce query (URL) pairwise similarities. Due to information embedded in Google search results, this method is superb in grouping semantically close queries together.*

Figure 9 shows an illustrative click-graph which can be combined with random walks to induce query variations directly from large query logs; using such a structure, an at-scale exploration of the ideas we have introduced here would be a useful further step in terms of validating the new approach.

Beyond Single-Faceted Information Needs. The UQV100 test collection is, by design, composed of topics for 100 single-facet information needs. Multi-faceted search would invariably provide additional challenges. If a suitable threshold in cluster association scores cannot be met to confidently associate a query with a single centroid, a fusion of many faceted query centroids may be required to resolve the user's information need. An alternative approach could involve building diversified query centroids for information needs that are typically diverse, allowing diversification to be implicitly included via the mechanisms we have introduced here. Search result diversification [1] is an important problem in its own right, and fusion techniques have already been shown to be highly effective for this problem [53].

7 CONCLUSIONS

We first showed that on-the-fly rank fusion over query variations is viable, and can be reasonably efficient using current state-of-the-art dynamic pruning techniques; but that if SLAs on query performance are being enforced, carrying out fusion at run-time remains costly.

We then demonstrated that query level fusion can instead be used to combine similar queries offline, making it a practical alternative. To validate that approach, we explored how query clusters can be used to improve the effectiveness of incoming queries using three different approaches, namely re-ranking, interleaving, and a linear combination of the cluster and the user query. In general, centroid-based re-ranking techniques are an attractive option, requiring on average only 200 microseconds to reorder 1,000 documents, while significantly improving effectiveness. Experiments using the ClueWeb12B UQV100 collection showed that the new approaches provide competitive efficiency, and, at the same time, effectiveness improvements over strong baselines in a performance-focused query processing framework.

Software. In the interests of reproducibility, our codebase is available at https://github.com/rmit-ir/centroid-boost.

Acknowledgements. This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP170102231), a Google Faculty Research Award, and an Amazon Research Award. We thank Xiaolu Lu and Luke Gallagher for their assistance with the L2P and LTR baselines.

REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. 2009. Diversifying search results. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 5–14.

[2] G. Amati and C. J. van Rijsbergen. 2002. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. on Information Systems* 20, 4 (2002), 357–389.

[3] Y. Anava, A. Shtok, O. Kurland, and E. Rabinovich. 2016. A probabilistic fusion framework. In *Proc. Int. Conf. on Theory of Information Retrieval (ICTIR)*. 1463–1472.

[4] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. 2008. Design trade-offs for search engine caching. *ACM Trans. on the Web* 2, 4 (2008), 1–28.

[5] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. 2016. UQV100: A test collection with query variability. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 725–728.

[6] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. 2017. Retrieval consistency in the presence of query variations. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 395–404.

[7] M. Barbaro and T. Zeller. 2006. A face is exposed for AOL searcher No. 4417749. https://nytimes.com/2006/08/09/technology/09aol.html. (Aug. 2006). Accessed: 2018-11-08.

[8] N. J. Belkin, C. Cool, W. B. Croft, and J. P. Callan. 1993. The effect of multiple query variations on information retrieval system performance. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 339–346.

[9] M. Bendersky, D. Metzler, and W. B. Croft. 2012. Effective query formulation with multiple information sources. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 443–452.

[10] R. Benham and J. S. Culpepper. 2017. Risk-reward trade-offs in rank fusion. In *Proc. Australasian Document Computing Symp. (ADCS)*. 1:1–1:8.

[11] R. Benham, J. S. Culpepper, L. Gallagher, X. Lu, and J. Mackenzie. 2018. Towards efficient and effective query variant generation. In *Proc. Conf. on Design of Experimental Search & Information Retrieval Systems (DESIRES)*. 62–67.

[12] R. Benham, L. Gallagher, J. Mackenzie, T. T. Damessie, R.-C. Chen, F. Scholer, A. Moffat, and J. S. Culpepper. 2017. RMIT at the 2017 TREC CORE track. In *Proc. Text Retrieval Conf. (TREC)*.

[13] R. Benham, L. Gallagher, J. Mackenzie, B. Liu, X. Lu, F. Scholer, A. Moffat, and J. S. Culpepper. 2018. RMIT at the 2018 TREC CORE track. In *Proc. Text Retrieval Conf. (TREC)*.

[14] B. Billerbeck, F. Scholer, H. E. Williams, and J. Zobel. 2003. Query expansion using associated queries. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 2–9.

[15] B. Billerbeck and J. Zobel. 2004. Techniques for efficient query expansion. In *Proc. Symp. on String Processing and Information Retrieval (SPIRE)*. 30–42.

[16] D. Broccolo, C. Macdonald, S. Orlando, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. 2013. Load-sensitive selective pruning for distributed search. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 379–388.

[17] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 426–434.

[18] C. Buckley, G. Salton, J. Allan, and A. Singhal. 1995. Automatic query expansion using SMART: TREC 3. In *Proc. Text Retrieval Conf. (TREC)*.

[19] C. Burges, R. Ragno, and Q. V. Le. 2006. Learning to rank with nonsmooth cost functions. In *Proc. Conf. on Neural Information Processing Systems (NIPS)*. 193–200.

[20] C. Cadwalladr and E. Graham-Harrison. 2018. Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach. https://theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election. (March 2018). Accessed: 2018-11-08.

[21] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge. 2010. A refreshing perspective of search engine caching. In *Proc. Int. Conf. on the World Wide Web (WWW)*. 181–190.

[22] M.-A. Cartright, J. Allan, V. Lavrenko, and A. McGregor. 2010. Fast query expansion using approximations of relevance models. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 1573–1576.

[23] K. Chakrabarti, S. Chaudhuri, and V. Ganti. 2011. Interval-based pruning for top-k processing over compressed lists. In *Proc. Int. Conf. on Data Engineering (ICDE)*. 709–720.

[24] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. 2017. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 445–454.

[25] G. V. Cormack, C. L. A. Clarke, and S. Büttcher. 2009. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 758–759.

[26] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. 2017. A comparison of document-at-a-time and score-at-a-time query evaluation. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 201–210.

[27] N. Craswell and M. Szummer. 2007. Random walks on the click graph. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 239–246.

[28] V. Dang and W. B. Croft. 2010. Query reformulation using anchor text. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 41–50.

[29] J. Dean and L. A. Barroso. 2013. The tail at scale. *Comm. ACM* 56, 2 (2013), 74–80.

[30] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. 2016. Compressing graphs and indexes with recursive graph bisection. In *Proc. Conf. on Knowledge Discovery and Data Mining (KDD)*. 1535–1544.

[31] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. 2013. Optimizing top-k document retrieval strategies for block-max indexes. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 113–122.

[32] S. Ding and T. Suel. 2011. Faster top-k document retrieval using Block-Max indexes. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 993–1002.

[33] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. 2006. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. on Information Systems* 24, 1 (2006), 51–78.

[34] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. 2011. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proc. Conf. on Very Large Databases (VLDB)* 4, 12 (2011), 1213–1224.

[35] E. A. Fox and J. A. Shaw. 1993. Combination of multiple searches. In *Proc. Text Retrieval Conf. (TREC)*. 243–252.

[36] N. Fuhr. 2018. Some common mistakes in IR evaluation, and how they can be avoided. *SIGIR Forum* 51, 3 (2018), 32–41.

[37] L. Gallagher, R.-C. Chen, R. Blanco, and J. S. Culpepper. 2019. Joint optimization of cascade ranking models. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 15–23.

[38] Q. Gan and T. Suel. 2009. Improved techniques for result caching in web search engines. In *Proc. Int. Conf. on the World Wide Web (WWW)*. 431–440.

[39] F. Hafizoglu, E. C. Kucukoglu, and I. S. Altingovde. 2017. On the efficiency of selective search. In *Proc. European Conf. on Information Retrieval (ECIR)*. 705–712.

[40] D. K. Harman. 1995. Overview of the third text retrieval conference (TREC-3). In *Proc. Text Retrieval Conf. (TREC)*.

[41] Y. He, J. Tang, H. Ouyang, C. Kang, D. Yin, and Y. Chang. 2016. Learning to rewrite queries. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 1443–1452.

[42] S. Huo, M. Zhang, Y. Liu, and S. Ma. 2014. Improving tail query performance by fusion model. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 559–658.

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

[44] S. Kim, Y. He, S.-W. Hwang, S. Elnikety, and S. Choi. 2015. Delayed-dynamic-selective (DDS) prediction for reducing extreme tail latency in web search. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 7–16.

[45] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2016. Does selective search benefit from WAND optimization?. In *Proc. European Conf. on Information Retrieval (ECIR)*. 145–158.

[46] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2017. Efficient distributed selective search. *Information Retrieval* 20, 3 (2017), 221–252.

[47] J. Kong, A. Scott, and G. M. Goerg. 2016. Improving semantic topic clustering for search queries with word co-occurrence and bigraph co-clustering. *Google Inc* (2016). https://ai.google/research/pubs/pub45569.pdf

[48] A. K. Kozorovitsky and O. Kurland. 2011. Cluster-based fusion of retrieved lists. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 893–902.

[49] O. Kurland and J. S. Culpepper. 2018. Tutorial: Fusion in information retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 1383–1386.

[50] V. Lavrenko and J. Allan. 2006. Real-time query expansion in relevance models. *IR 473, University of Massachusetts Amherst* (2006).

[51] V. Lavrenko and W. B. Croft. 2001. Relevance-based language models. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 120–127.

[52] C.-J. Lee, Q. Ai, W. B. Croft, and D. Sheldon. 2015. An optimization framework for merging multiple result lists. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 303–312.

[53] S. Liang, Z. Ren, and M. de Rijke. 2014. Fusion helps diversification. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 303–312.

[54] T.-Y. Liu. 2009. Learning to rank for information retrieval. *Foundations & Trends in Information Retrieval* 3, 3 (2009), 225–331.

[55] X. Lu, A. Moffat, and J. S. Culpepper. 2016. The effect of pooling and evaluation depth on IR metrics. *Information Retrieval* 19, 4 (2016), 416–445.

[56] X. Lu, A. Moffat, and J. S. Culpepper. 2016. Efficient and effective higher order proximity modeling. In *Proc. Int. Conf. on Theory of Information Retrieval (ICTIR)*. 21–30.

[57] H. Ma and B. Wang. 2012. User-aware caching and prefetching query results in web search engines. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 1163–1164.

[58] C. Macdonald, R. L. T. Santos, and I. Ounis. 2013. The whens and hows of learning to rank for web search. *Information Retrieval* 16, 5 (2013), 584–628.

[59] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. 2013. About learning models with multiple query-dependent features. *ACM Trans. on Information Systems* 31, 3 (2013), 11:1–11:39.

[60] J. Mackenzie. 2017. Managing tail latencies in large scale IR systems. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 1369.

[61] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, and J. Lin. 2018. Query driven algorithm selection in early stage retrieval. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 396–404.

[62] J. Mackenzie, A. Mallia, M. Petri, J. S. Culpepper, and T. Suel. 2019. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Proc. European Conf. on Information Retrieval (ECIR)*. 339–352.

[63] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. 2017. Faster blockmax WAND with variable-sized blocks. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 625–634.

[64] A. Mallia, M. Siedlaczek, and T. Suel. 2019. An experimental study of index compression and DAAT query processing methods. In *Proc. European Conf. on Information Retrieval (ECIR)*. 353–368.

[65] D. Metzler and W. B. Croft. 2005. A Markov random field model for term dependencies. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 472–479.

[66] D. Metzler, S. Dumais, and C. Meek. 2007. Similarity measures for short segments of text. In *Proc. European Conf. on Information Retrieval (ECIR)*. 16–27.

[67] A. Moffat. 2016. Judgment pool effects caused by query variations. In *Proc. Australasian Document Computing Symp. (ADCS)*. 65–68.

[68] A. Moffat, P. Bailey, F. Scholer, and P. Thomas. 2017. Incorporating user expectations and behavior into the measurement of search effectiveness. *ACM Trans. on Information Systems* 35, 3 (2017), 24:1–24:38.

[69] A. Moffat, F. Scholer, P. Thomas, and P. Bailey. 2015. Pooled evaluation over query variations: Users are as diverse as systems. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 1759–1762.

[70] A. Moffat and J. Zobel. 2008. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. on Information Systems* 27, 1 (2008), 2.1–2.27.

[71] A. Mourão and J. Magalhães. 2018. Low-complexity supervised rank fusion models. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 1691–1694.

[72] G. Ottaviano and R. Venturini. 2014. Partitioned Elias-Fano indexes. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 273–282.

[73] J. Ponte and W. B. Croft. 1998. A language modeling approach to information retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 275–281.

[74] F. Radlinski, M. Kurup, and T. Joachims. 2008. How does clickthrough data reflect retrieval quality?. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 43–52.

[75] S. E. Robertson and S. Walker. 2000. Microsoft Cambridge at TREC-9: Filtering track. In *Proc. Text Retrieval Conf. (TREC)*.

[76] J. J. Rocchio. 1971. Relevance feedback in information retrieval. *The SMART retrieval system: Experiments in automatic document processing* (1971), 313–323.

[77] F. Scholer and H. E. Williams. 2002. Query association for effective retrieval. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 324–331.

[78] D. Sheldon, M. Shokouhi, M. Szummer, and N. Craswell. 2011. LambdaMerge: Merging the results of query reformulations. In *Proc. ACM Int. Conf. on Web Search and Data Mining (WSDM)*. 795–804.

[79] J. Shen, M. Karimzadehgan, M. Bendersky, Z. Qin, and D. Metzler. 2018. Multi-task learning for email search ranking with auxiliary query clustering. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 2127–2135.

[80] T. Strohman, H. Turtle, and W. B. Croft. 2005. Optimization strategies for complex queries. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 219–225.

[81] H. Turtle and W. B. Croft. 1991. Evaluation of an inference network-based retrieval model. 9, 3 (1991), 187–222.

[82] H. R. Turtle and J. Flood. 1995. Query evaluation: Strategies and optimizations. *Information Processing & Management* 31, 6 (1995), 831–850.

[83] C. C. Vogt. 1999. *Adaptive combination of evidence for information retrieval.* Ph.D. Dissertation. http://cseweb.ucsd.edu/groups/guru/docs/theses/vogt-thesis.pdf

[84] C. C. Vogt. 2000. How much more is better? Characterizing the effects of adding more IR systems to a combination. In *Proc. Recherche d'Information etses Applications (RIAO)*. 457–475.

[85] C. C. Vogt and G. W. Cottrell. 1999. Fusion via a linear combination of scores. *Information Retrieval* 1, 3 (1999), 151–173.

[86] J. Wang, E. Lo, M. L. Yiu, J. Tong, G. Wang, and X. Liu. 2014. Cache design of SSD-based search engine architectures: An experimental study. *ACM Trans. on Information Systems* 32, 4 (2014), 1–26.

[87] J. R. Wen, J. Y. Nie, and H. J. Zhang. 2001. Clustering user queries of a search engine. In *Proc. Int. Conf. on the World Wide Web (WWW)*. 162–168.

[88] J. R. Wen, J. Y. Nie, and H. J. Zhang. 2002. Query clustering using user logs. *ACM Trans. on Information Systems* 20, 1 (2002), 59–81.

[89] H. Wu and H. Fang. 2013. An incremental approach to efficient pseudo-relevance feedback. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 553–562.

[90] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, and W. Fan. 2004. Optimizing web search using web click-through data. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 118–126.

[91] X. Xue and W. B. Croft. 2013. Modeling reformulation using query distributions. *ACM Trans. on Information Systems* 31, 2 (2013), 6:1–6:34.

[92] J.-M. Yun, Y. He, S. Elnikety, and S. Ren. 2015. Optimal aggregation policy for reducing tail latency of web search. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 63–72.

[93] J. Zobel and A. Moffat. 2006. Inverted files for text search engines. *Comput. Surveys* 38, 2 (2006), 6.1–6.56.

Received November 2018; revised May 2019; accepted July 2019