# Finding the Best of Both Worlds: Faster and More Robust Top-$k$ Document Retrieval

Omar Khattab*
Stanford University
okhattab@stanford.edu

Mohammad Hammoud
Carnegie Mellon University in Qatar
mhhamoud@cmu.edu

Tamer Elsayed
Qatar University
telsayed@qu.edu.qa

## ABSTRACT

Many top-$k$ document retrieval strategies have been proposed based on the WAND and MaxScore heuristics and yet, from recent work, it is surprisingly difficult to identify the "fastest" strategy. This becomes even more challenging when considering various retrieval criteria, like different ranking models and values of $k$. In this paper, we conduct the first *extensive* comparison between ten effective strategies, many of which were never compared before to our knowledge, examining their efficiency under five representative ranking models. Based on a careful analysis of the comparison, we propose LazyBM, a remarkably simple retrieval strategy that bridges the gap between the best performing WAND-based and MaxScore-based approaches. Empirically, LazyBM considerably outperforms all of the considered strategies across ranking models, values of $k$, and index configurations under both mean and tail query latency.

## KEYWORDS

Query Evaluation; Dynamic Pruning; Efficiency; Web Search

**ACM Reference Format:**
Omar Khattab, Mohammad Hammoud, and Tamer Elsayed. 2020. Finding the Best of Both Worlds: Faster and More Robust Top-$k$ Document Retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401076

## 1 INTRODUCTION

Search engines typically employ a multi-stage ranking architecture to answer user queries [25, 40, 42]. In the first stage, the search engine retrieves the top-$k$ candidate results based on a standard *ranking model* (e.g., BM25 [32]). Subsequently, the top-$k$ candidates are *re-ranked* more holistically, using more expensive yet more precise machine-learned models (e.g., based on neural networks [18, 29]), before the final list is presented to the searcher.

To respect stringent latency constraints, the first-stage ranking must be completed in just a matter of tens or hundreds of milliseconds [19]. To this end, numerous retrieval strategies have been devised to efficiently obtain the top-$k$ results from the inverted index without exhaustively scoring all possible candidates [5, 6, 14, 15, 26, 35, 38]. Among those, *safe* strategies are ones that retrieve the exact top-$k$ results as exhaustive search. Most of these follow a *document-at-a-time* (DAAT) paradigm and stem from the classic WAND [5] or the canonical MaxScore strategies [37, 38].

Despite the renewed attention this area has received over the last few years, it is surprisingly difficult to identify a "best" retrieval strategy or, perhaps, even to determine if significant progress has occurred since the introduction of MaxScore in the 1990s. Ding and Seul [15] found Block-Max WAND (BMW) to be decisively better than basic WAND, and Chakrabarti *et al.* [6] found Block-Max MaxScore (BMM) to greatly outpace vanilla MaxScore. However, recent work [9] suggests that BMW often provides little or no gain over WAND and another [14] finds that vanilla BMM is barely competitive with MaxScore. Recently, Mallia *et al.* [28] found MaxScore to outperform state-of-the-art VBMW [26] under many of their settings. Interestingly, all of the mentioned studies evaluate efficiency using the BM25 ranking model. Considering that related work illuminates the dependence of pruning efficiency on the ranking model deployed [30], it remains unclear how the majority of these strategies fare under various models.

We begin by tackling precisely these issues. We conduct an *extensive* comparison between ten representative query evaluation strategies, controlling for indexing-related variables and exploring the effects of the size of retrieved list $k$ and the ranking model. We find that the seemingly small differences between existing strategies manifest themselves clearly in terms of performance, especially as we consider various retrieval models.

Based on our analysis, we propose LazyBM, a remarkably simple and efficient top-$k$ retrieval strategy. In contrast to WAND's *pessimistic* pruning and MaxScore's *eager* evaluation, LazyBM adopts a balanced pruning heuristic that judiciously *layers* both together to swiftly and yet aggressively utilize local bounds for pruning. As we show empirically, this enables LazyBM to considerably and consistently outperform existing strategies across ranking models, values of $k$, and index configurations.

Our contributions in this work are three-fold:

(1) We conduct an extensive comparison that brings together ten effective dynamic pruning strategies and evaluates them across five representative retrieval models in a first comprehensive kind of study (§3, 4, and 5).
(2) We propose a simple retrieval strategy (§6) that layers the heuristics of WAND and MaxScore with little overhead. We release our reference implementations as open source.[1]
(3) We thoroughly evaluate the proposed strategy (§7). Over the fastest strategy, it speeds up *mean* latency by about 1.9×

*Work done while the first author was at Carnegie Mellon University in Qatar.

[1] https://github.com/okhat/LazyBM


*SIGIR '20, July 25–30, 2020, Virtual Event, China*

ACM ISBN 978-1-4503-8016-4/20/07.
https://doi.org/10.1145/3397271.3401076

on average (up to 4.0×) and *tail* latency by about 2.2× on average (up to 4.7×).

## 2 BACKGROUND & RELATED WORK

Search engines typically employ an *inverted index* to represent a document collection. The inverted index enumerates for every unique term $t$ its *postings list*, i.e., the list of *IDs* of documents containing $t$. To answer a query $Q$, the search engine consults the index to identify the best $k$ matches, as scored by a *ranking model* that assigns each document $d$ a *score* $S(d, Q)$ estimating its relevance to $Q$. In a multi-stage search pipeline, the *first-stage* ranking model is generally expressed as a linear combination of *per-term* score contributions $s(t, d)$ [9, 26], and may also involve a *term-independent* component $s(d)$, combined as follows:

$$S(d, Q) \;=\; s(d) + \sum_{t \in Q \cap d} w_t \cdot s(t, d) \qquad (1)$$

where $w_t$ is the query-dependent weight of $t$. These top-$k$ matches can be subsequently *re-ranked* before the top few are finally presented to the searcher. Such re-ranking often considers the results set more holistically, relying on tens or even hundreds of features and deploying more sophisticated ranking algorithms (e.g., boosted trees [22] or neural networks [18, 29]). Owing to the high cost of feature extraction and score computation *per document* of such re-rankers, they are only applied on a limited number of promising candidate results.

**Top-$k$ Retrieval Strategies.** Given a ranking model, search engines deploy *top-k retrieval strategies* to process queries on top of an inverted index. The strategies leverage *dynamic pruning* heuristics to identify top-$k$ matches for each query *without* exhaustively scoring every candidate. The majority of recent safe query evaluation strategies adopt a *Document-At-A-Time* (DAAT) processing scheme [37]. Specifically, they process (or skip) every candidate document before proceeding to consider the next. To identify the top-$k$ documents, DAAT-based strategies maintain the $k^{\text{th}}$ largest document score seen so far as a *threshold* $\theta$. Documents whose scores exceed $\theta$ are inserted into a *top-k heap*, which stores the best $k$ candidates seen as the query is being processed.

Basic DAAT evaluation exhaustively scores every document that contains at least one query term. To do so, it repeatedly identifies the *smallest unprocessed* document ID, say $d$, across all query terms, and accumulates the partial scores of $d$ as indicated by Equation 1. If $d$'s score exceeds $\theta$, it is inserted into the heap. To avoid exhaustive evaluation, more advanced DAAT-based strategies, which generally fall into the MaxScore and the WAND families [37], employ various heuristics for pruning unpromising results at query time. In §4, we describe WAND and four of its variants BMW [15], LBMW [35], DBMW [14], and VBMW [26], and conduct the first extensive comparison between the members of this family. In §5, we pursue the same for the MaxScore family, including the variants BMM [14, 35], LBMM [35], IBMM [6], and DBMM [14]. While all of these strategies are optimized and evaluated with BM25 (combined with PageRank in [35]), our work sets itself apart in that we consider a large variety of effective ranking models and explicitly seek heuristics that *robustly* improve retrieval efficiency.

**Evaluating Strategies Across Ranking Models.** The literature includes little work that evaluates pruning heuristics across ranking models. Petri *et al.* [30] compare WAND and BMW under BM25 and LM [43]. Macdonald *et al.* [23, 24] consider the problem of computing approximate upper bounds for BM25, LM, and DLH13 [1] models. They study the effect of their approximations on the efficiency of WAND, MaxScore, and BMW.

**Accelerating DAAT Pruning Strategies.** The literature also includes work on augmenting DAAT-based strategies with auxiliary mechanisms for quickly eliminating unpromising candidate documents. For example, Daoud *et al.* [10, 11] study index *tiering* strategies, Dimopoulos *et al.* [13] study techniques for candidate *pre-filtering* relying on SIMD vectorization and exploiting query logs, and Petri *et al.* [31] consider initial threshold prediction. Yet other mechanisms include treap-based conditional skips [4] on top of a provided pruning strategy and efficiently parallelizing pruning within shared memory [34] or distributed memory [33]. While we study top-$k$ retrieval strategies in this work, future work will need to consider how the competitive algorithms in our study fare under such higher-level optimizations.

**Other Query Evaluation Paradigms.** The majority of work in the last decade focuses on DAAT mechanisms, which appears to be in use by Google [12] and Bing [19]. However, there is also rich literature on Term-At-A-Time (TAAT) [17, 38] and Score-At-A-Time (SAAT) [20, 21]. TAAT approaches were popular in classic retrieval systems at small scale, but the introduction of superior DAAT pruning strategies overshadowed them, especially since TAAT is slower on larger indexes [17]. SAAT approaches are popular primarily for their *anytime* controllable relaxation of effectiveness to meet deadlines [20]. We focus on DAAT, which is adopted by the most efficient safe strategies [9].

## 3 EXPERIMENTAL SETUP

In §4, 5, and 7, we thoroughly evaluate five WAND-based top-$k$ evaluation strategies, five MaxScore-based strategies, and our proposed strategy, respectively. This section outlines the experimental setup that guides our subsequent comparisons and analyses.

**Ranking Models.** We model the computation of document scores as formulated in Equation 1. To study the *robustness* of various query evaluation strategies, we consider five diverse and competitive ranking models, namely, BM25 [32], Language Modeling with Dirichlet Smoothing (LMDir) [43], PL2 [2], SPL [7], and F2EXP [16]. While BM25 is a probabilistic retrieval model, LMDir, PL2, SPL, and F2EXP are representatives of the language modeling, divergence from randomness, information-based, and axiomatic approaches to retrieval, respectively. Our choice is directly based on recent work by Yang *et al.* [41], whose work focused on the *effectiveness* of these models in the context of reproducibility.[2]

More broadly than these models, our goal is to identify pruning heuristics that can effectively speed up query processing under a wide variety of realistic settings. These choices allow us to scrutinize the robustness of all strategies while exploring a range of computation costs and score distributions. For instance, *exhaustive* DAAT evaluation with SPL ranking model is over 8× more expensive than the cheapest models BM25 and F2EXP (see Figure 1)

[2]Since not all of these ranking models have recommended default parameters, the parameters of each model were tuned on the TREC Web Track 2013 queries [8] in terms of NDCG@20 in the same manner of Yang *et al.* [41].

due to SPL's more expensive term–document weight computations. Moreover, like Petri *et al.* [30] observed for BM25 and LMDir, we see that the scores (in particular, the *block bounds*) for BM25 and to a lesser extent F2EXP skew heavily towards their maximum value per term—much more so than PL2, SPL, and LMDir. In effect, this renders *fine-grained* pruning more crucial under PL2, SPL, and LMDir. As we show in §7, these differences in cost and distributions can be nearly hidden by effective top-$k$ pruning heuristics that swiftly bypass all but few of the score computations.

**Implementation of Query Evaluation Strategies.** Our code is written in C++ and compiled with GCC (g++) v8 using the highest optimization settings. It brings together two open-source repositories plus our own implementations in a common evaluation framework. Our implementation of WAND, MaxScore, BMW, and VBMW is based on the code by Mallia *et al.* [26].[3] Our implementation of BMM, DBMW, and DBMM is based on the code by Dimopoulos *et al.* [14].[4] We implement LBMW, LBMM, and IBMM similarly.

For the *postings-oriented* Block-Max strategies (i.e., all but the *DocID-oriented* DBMW, DBMM, and our proposed LazyBM), we use blocks of size 64 postings, as in [14, 35]. With this, postings-oriented strategies require 3.1GiBs to store the term and block bounds (except for the LMDir model, where they use 4.4GiBs due to the static document scores). For VBMW, whose *variable* block sizes are controlled indirectly by a parameter $\lambda$, we apply binary search over $\lambda$ to obtain *average* block size of $64 \pm 1$ over the terms of our query set.

For the DocID-oriented strategies, we apply quantization and on-the-fly-generation under the "variable" setting recommended by Dimopoulos *et al.* [14], starting with blocks that span 128 document IDs for terms with over $2^{18}$ postings. Under this configuration, DocID-oriented strategies require 3.9GiBs across ranking models to store the upper bounds. This is largely possible due to on-the-fly-generation, as a result of which they do not need to store bounds for the vast majority of terms. In §7, we revisit the impact of the block sizes on LazyBM and the fastest competitor.

**Datasets & Evaluation Settings.** We conduct our evaluation using the ClueWeb12-B13 (Category B) Web collection, which consists of 52M English Web pages. We use 1,000 queries sampled randomly from the TREC Million Query Track (MQT) 2007–2009,[5] a query set comprising 60K queries in total. Our sample reflects the length distribution of the original query set: about 15%, 27%, 24%, 14%, and 7% contain one to five terms, respectively, and the remaining 13% have six or more terms. We index the collection and preprocess the queries using the recently-released PISA toolkit [27]. We use the default Porter2 stemmer and remove no stopwords.

Our index is compressed using the popular Elias-Fano (EF) encoding [39], using Facebook's optimized folly implementation,[6] and retaining the default (crawl-based) order of documents. We run our experiments on a VM with 8 CPUs and 64GB of RAM hosted on a private research cloud. In addition, we test the impact of re-ordering the documents using their URLs and of using a VarintG8IU [36]-encoded index, employing the implementation from the FastPFor[7]

[3]https://github.com/rossanoventurini/Variable-BMW
[4]https://github.com/dimopoulosk/WSDM13
[5]https://trec.nist.gov/data/million.query.html
[6]https://github.com/facebook/folly
[7]https://github.com/lemire/FastPFor

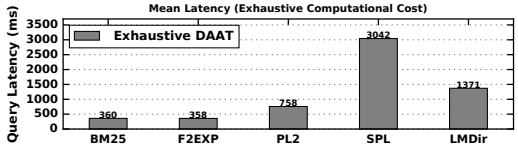


**Figure 1: Cost of *exhaustive* DAAT across models.**

and the ds2i[8] libraries. We run these auxiliary experiments on another VM that has 32GB of RAM but is otherwise identical in its specifications.[9] Similar to related work [14, 15, 26, 35], our index—in particular, the postings lists corresponding to terms in our query set—is entirely memory-resident.

**Evaluation Metrics.** We consider mean and *tail* query latency (in particular, latency at the 95th percentile). We report results for top-$k$ retrieval with $k = 10$ and $k = 1000$.

## 4 EVALUATION OF WAND VARIANTS

In this section, we describe the WAND family of pruning strategies and conduct an extensive comparison between five representative WAND-based strategies from the literature. Specifically, this section aims to answer $\mathbf{RQ}_1$ stated as follows:

$\mathbf{RQ}_1$: How do the WAND-based strategies compare under various ranking models? In particular, which of their pruning heuristics consistently enhance efficiency?

### 4.1 Strategies

To aid our analysis, we briefly describe the strategies below. For the complete descriptions, refer to the original respective papers.

**Weak AND (WAND).** In 2003, Broder *et al.* [5] proposed the Weak-AND (WAND) strategy. For every term $t$, WAND maintains an *upper bound* $U_t := \max_{t \in d} s(t, d)$ over $t$'s contributions to document scores. As its name indicates, WAND attempts to narrow the efficiency gap between disjunctive (OR) and conjunctive (AND) query processing. It employs a *pivot selection* step to identify the next document to be evaluated, potentially bypassing many unpromising documents. During pivot selection, WAND *sorts* the query terms in increasing order of each term's *smallest unprocessed* document ID. In this order, WAND computes a prefix sum of the terms' upper bounds, identifying the *pivot term*, that is, the first term $t$ for which the sum exceeds the top-$k$ threshold $\theta$ (§2). WAND takes $t$'s smallest unprocessed document ID as the next candidate $d$, safely skipping any unprocessed documents with smaller IDs. Subsequently, WAND employs a *score computation* step that computes $S(d, Q)$ and inserts $d$ into the heap if its score exceeds $\theta$.

**Block-Max WAND (BMW).** In 2011, Ding & Suel [15] introduced the *Block-Max* index, which splits the postings of every term $t$ into equal-sized *blocks* (e.g., groups of 128 consecutive postings). For each block $b$, it maintains a block upper bound $U_{t,b} := \max_{t \in d \wedge (t,d) \in b} s(t, d)$ over the contributions corresponding to postings within $b$. Intuitively, a block bound $U_{t,b}$ is often much *tighter* than the term bound $U_t$ used by WAND, enabling more aggressive

[8]https://github.com/ot/ds2i
[9]The results for the VarintG8IU index are very similar to the other configurations. In the interest of space, we only include them for the best-performing strategies.

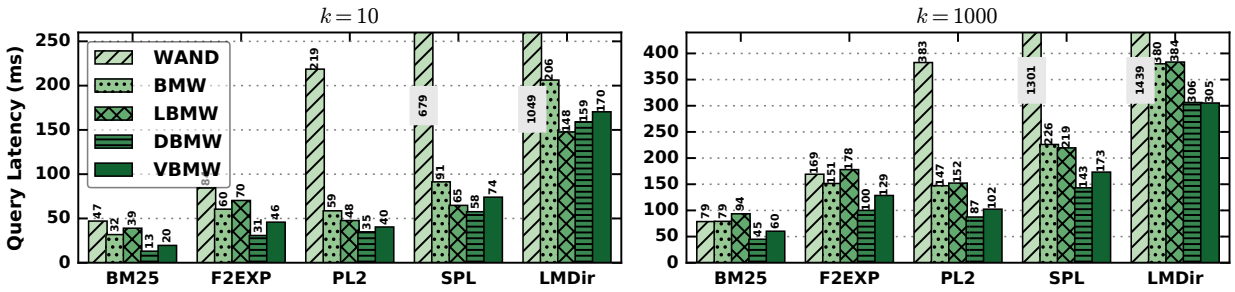


**Figure 2: Mean query latency (in milliseconds) of five WAND-based strategies across five ranking models. Latency is reported for top-$k$ queries for $k = 10$ and $k = 1000$.**

pruning. On top of WAND's pivot selection, BMW skips any document $d$ whose block bound falls short of $\theta$. When this occurs, it bypasses all the documents that share the same block bound as $d$.

**Local BMW (LBMW).** In 2012, Shan *et al.* [35] proposed LBMW, which modifies the *pivot selection* stage of BMW. While BMW's pivot selection relies on the (global) term upper bounds (like WAND), LBMW exploits the (local) block bounds for this purpose. In selecting the pivot term, it computes a *local* upper bound for every term $t$ by considering the maximum of all block upper bounds for $t$ up to the last term's smallest unprocessed document ID. This enables LBMW to select fewer candidate documents for further evaluation, at the cost of more sophisticated pivot selection mechanism. In Shan *et al.*'s experiments, this gives LBMW substantial advantage over BMW when the ranking model incorporates a static document score (e.g., PageRank), that is, when $s(d)$ in Equation 1 is non-zero. In this section, this applies to the LMDir model.

**DocID-oriented BMW (DBMW).** In 2013, Dimopoulos *et al.* [14] introduced a *DocID-oriented* Block-Max structure. In BMW and LBMW, every block consists of a fixed number of consecutive *postings* (e.g., 64). In DocID-oriented Block-Max, a postings list is divided into equal-sized *intervals* of document IDs, and blocks correspond to postings in one interval (e.g., document IDs 1024 to 2047 constitute a block, irrespective of how many postings therein). DBMW is a straightforward adaptation of BMW to DocID-oriented block boundaries. Notably, it enjoys simplified (and hence faster) skip computations, since skipping over a block can be calculated with bit-shifts instead of complex data-dependent computations.

**Variable BMW (VBMW).** More recently, in 2017, Mallia *et al.* [26] suggested another mechanism for selecting block boundaries. In contrast to *constant-sized* postings-based blocks, they proposed a *dynamic programming* algorithm for automatically selecting the postings-based size of each individual block so as to minimize the bound *error* (i.e., average difference between each block bound and the corresponding individual scores) while maintaining the *average* block sizes unchanged for more effective skipping.

## 4.2 Results

For each strategy, we run top-10 and top-1000 queries with the five representative ranking models and report mean query latency in Figure 2. Mean and tail (in particular, 95th percentile) latency across the main and auxiliary index settings are also reported in Tables 1 & 2. Starting with the figure, we notice considerable variation in performance in the WAND family, as we discuss below.

As the figure shows, BMW invariably outperforms WAND. Besides only modest gains under BM25 (only with $k = 10$), BMW demonstrates increasingly larger margins as we move to more expensive models. Refer to Figure 1 for the computational costs of DAAT across the five models. To explain, the improved ability of BMW in skipping score computations translates to larger gains when the cost of such computations is increased. Under the hood, we also see that BMW's advantage in skipping (i.e., as measured by the number of *skipped* documents and score computations) also *increases* as we move from left to right, further widening the latency gap between WAND and BMW (statistics not shown due to space).

Overall, these results reinforce the notion that BMW has a substantial edge over WAND, especially when considering robustness to various ranking models. Nevertheless, under BM25, we confirm Crane *et al.* [9]'s recent observation that BMW's advantage is largely confined to short queries (i.e., up to 3–4 terms). This is clear with $k = 1000$, where BMW delivers no speedup over WAND. However, this trend is not evident for the rest of the models.

Shifting our attention to LBMW, we observe its great advantage over BMW under LMDir with $k = 10$, which exhibits a non-zero term-independent component $s(d)$ responsible for penalizing long documents. By design, the modified pivot selection mechanism characterizing LBMW optimizes pruning for such models. Besides LMDir with $k = 10$, the results show no clear winner among LBMW and BMW. While LBMW reduces the number of pivot selections, doing so imposes overheads that offset much of the potential latency gains. Overall, the modified pivot selection of LBMW appears worthwhile only when the ranking model computations are sufficiently expensive to warrant the added overheads *and* with relatively small $k$ where a larger threshold $\theta$ facilitates longer skips.

Next, we examine DBMW and VBMW, both of which essentially employ BMW's pruning on top of variable-sized blocks. VBMW explicitly optimizes its choices of block boundaries to tighten their corresponding upper bounds. DBMW adopts a simpler rule that delimits blocks by intervals of document IDs. To our knowledge, the literature contains no direct comparison between these approaches. More often than not (i.e., in 8 out of 10 settings), we see that DBMW and VBMW are the two fastest strategies. Interestingly, DBMW is faster in almost all of these cases, being the single fastest strategy under all but two setting (i.e., LMDir with $k = 10$, where LBMW is superior and LMDir with $k = 1000$, where DBMW and VBMW show comparable latencies).

| | WAND | BMW | LBMW | DBMW | VBMW |
|---|---|---|---|---|---|
| **BM25** | 47 (184) | 32 (137) | 39 (171) | **13** (**44**) | 20 (85) |
| **F2EXP** | 84 (341) | 60 (255) | 70 (302) | **31** (**126**) | 46 (207) |
| **PL2** | 219 (920) | 59 (252) | 48 (207) | **35** (**141**) | 40 (178) |
| **SPL** | 679 (3341) | 91 (430) | 65 (293) | **58** (**245**) | 74 (333) |
| **LMDir** | 1049 (5821) | 206 (1018) | **148** (**721**) | 159 (746) | 170 (836) |

**Table 1: Mean and tail top-10 query latency for results of Figure 2. As in the figure, this experiment uses an EF index with the default (crawl-based) document order.**

| | WAND | BMW | LBMW | DBMW | VBMW |
|---|---|---|---|---|---|
| **BM25** | 32 (134) | 18 (77) | 22 (94) | **9** (**31**) | 12 (45) |
| **F2EXP** | 62 (249) | 35 (155) | 38 (164) | **19** (**79**) | 27 (120) |
| **PL2** | 183 (810) | 38 (166) | 25 (108) | **24** (**96**) | 27 (121) |
| **SPL** | 661 (3174) | 61 (301) | **34** (**147**) | 40 (178) | 51 (235) |
| **LMDir** | 1003 (5530) | 141 (687) | **83** (**412**) | 106 (491) | 116 (534) |

**Table 2: Mean and tail top-10 query latency after URL-ordering the documents in the EF index.**

Upon digging deeper, we find that the heuristics suggested by DBMW and VBMW are generally competitive with each other in terms of pruning ability (i.e., document evaluations and score computations), considerably improving upon BMW. However, DBMW uniquely reaps the benefits of DocID-oriented blocks, particularly faster access to the upper bounds and cheaper computation of the skip limits (i.e., the position delimiting the current block). It is this low overhead that allows DBMW to invariably surpass VBMW in terms of query latency.

Tables 1 and 2 contrast mean and tail latency across crawl- and URL-ordering, respectively. As the tables show, the fastest methods in mean latency tends to be the fastest with tail latency as well, with DBMW (and occasionally LBMW) outperforming all other WAND-based strategies. We observed similar results with the VarintG8IU index, omitted in the interest of space.

**Lessons Learned.** In summary, the results above allow us to glean the following insights:

(1) **Block-Max is crucial**: Employing a Block-Max structure contributes substantially to the efficiency and robustness of WAND-based pruning.
(2) ***Low-overhead* skipping is challenging**: The manner in which the Block-Max metadata is constructed and utilized matters a great deal. Heuristics that enable *low-overhead* skipping are particularly effective for consistently reducing query latency (e.g., considering LBMW vs. the simpler BMW and VBMW vs. the simpler DBMW).
(3) ***Local* pivot selection is promising**: Maximally utilizing the Block-Max information for pivot selection, and not only candidate filtering, is a promising feature that appears insufficiently tapped by existing WAND variants.

We next consider the MaxScore family. In contrast to WAND's rather *pessimistic* sort-based pivot selection, MaxScore centers on a computationally cheap heuristic for filtering unpromising documents, a quality made particularly attractive by lesson (2) above.

# 5 EVALUATION OF MAXSCORE VARIANTS

In this section, we describe and evaluate five representative members of the MaxScore family, examining $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ stated below.
$\mathbf{RQ}_2$: How do the MaxScore-based strategies compare under various ranking models? In particular, which of their pruning heuristics consistently enhance efficiency?
$\mathbf{RQ}_3$: How do these strategies compare against DBMW, the fastest among the WAND family?

## 5.1 Strategies

Below, we briefly describe the five MaxScore variants that we investigate in this section.

**MaxScore.** Described by Turtle and Flood [38] in 1995, the classic MaxScore preceded WAND in utilizing term upper bounds for pruning. During query evaluation, MaxScore uses $\theta$ to designate as many query terms as possible as *non-essential*, i.e., terms whose *sum of upper bounds* falls short of $\theta$. Evidently, no document containing *only* non-essential terms can join the top $k$. Accordingly, MaxScore restricts its processing to documents appearing within the postings lists of the remaining, *essential*, terms. For each such document $d$, MaxScore terminates its scoring as soon as the partial computed score and the upper bounds of the unprocessed terms indicate that $d$ cannot join the top $k$.

**Block-Max MaxScore (BMM).** Described by Shan *et al.* [35] in 2012, BMM is an enhancement of MaxScore that straightforwardly integrates the Block-Max metadata for more fine-grained pruning. The implementation, as described in detail in [14] and [37], evaluates the same documents as MaxScore, but terminates the scoring of the *non-essential* terms for a candidate document $d$ as soon as the *block-based* upper bound on the unprocessed terms indicates that $d$ cannot join the top $k$.[10]

**Local BMM (LBMM).** Shan *et al.* [35] also introduced the variant LBMM that more aggressively utilizes the Block-Max metadata. On top of BMM, before evaluating a document $d$, LBMM first estimates a *local* upper bound on the score of $d$ by summing the terms' block bounds. A document $d$ is only evaluated if its local upper bound exceeds the threshold $\theta$—if this bound falls short of $\theta$, it is safe to skip all the documents delimited by the same bound.

**Interval-based BMM (IBMM).** The first Block-Max variant of MaxScore was proposed by Chakrabarti *et al.* [6] in 2011. In contrast to the other strategies, IBMM splits the processing into two major stages. In the *interval generation* stage, this postings-oriented partitions the domain of document IDs into consecutive *intervals* using the boundaries of its blocks. As a result, it delimits the documents in each interval with a tight block-based upper bound. In the subsequent *interval pruning* stage, it iterates over the intervals whose upper bounds exceed $\theta$, and processes each interval in a manner similar to MaxScore but replacing the term bounds with block bounds.

**DocID-based BMM (DBMM).** The most recent Block-Max variant of MaxScore was proposed by Dimopoulos *et al.* [14] in 2013. Similar to DBMW from §4, DBMW relies on DocID-based blocks for pruning. Similar to BMM and LBMM, DBMM selects its pivot

[10] To be consistent with the literature, we reserve the term BMM for this basic variant of MaxScore. For disambiguation, we refer to Chakrabarti *et al.* [6]'s *earlier* variant as Interval-based Block-Max MaxScore (IBMM).

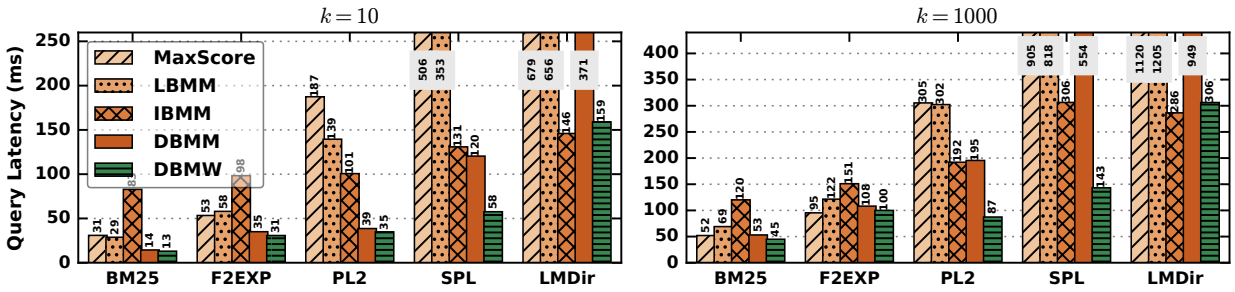


**Figure 3: Mean query latency (in milliseconds) of four MaxScore-based strategies (in addition to DBMW from §4) across five ranking models. Latency is reported for top-$k$ queries for $k = 10$ and $k = 1000$. BMM never outperforms MaxScore by more than 1.1×, hence, its results are not shown to maintain clarity of the figure.**

documents based on the (global) term partitioning of MaxScore. Before evaluating a pivot document $d$, DBMM applies a series of checks that filter out $d$ if its block-based upper bound is below $\theta$. If a document $d$ is filtered out because the sum of the block bounds falls short of $\theta$, DBMM applies an optimization, Next-Live-Block (NLB), that bypasses over the current *dead* DocID-oriented block and any subsequent blocks whose DocID-oriented upper bound is below $\theta$. While similar to IBMM's skipping of unpromising intervals, the NLB optimization of DBMM does not require a preprocessing step for interval generation, owing to the natural alignment of the DocID-oriented blocks.

## 5.2 Results

As we did for the WAND-based strategies, we report mean query latency for running top-10 and top-1000 queries under the five ranking models for each strategy. The results are shown in Figure 3; further, mean and tail latency across two settings are in Tables 3 & 4. For reference, the figure and tables include DBMW, the fastest WAND-based strategy from §4. In our results, BMM's speedup over MaxScore is negligible on average (with maximum speedup of 1.1×), thus we omit BMM from Figure 3 for clarity. Our findings for BMM generalize the BM25-based results of Mallia *et al.* [28] and Dimopoulos *et al.* [14]. To elucidate, BMM employs MaxScore's partitioning of terms unmodified, selecting as many pivot documents as MaxScore does. Although BMM can terminate the scoring of an individual document earlier (i.e., by relying on the block bounds), most queries contain just a few terms. This leaves little room for gains due to early-termination while still burdening BMM with additional overheads for its filtering logic.

In fact, if we restrict our attention to the BM25 model, we see that LBMM and IBMM show little or no gains over MaxScore (and, in fact, IBMM is significantly slower). For BM25, the simple logic of MaxScore lends itself naturally to low query latency, achieving results comparable with BMW and LBMW from §4. In case of IBMM, the strategy's interval generation stage adds significant overhead, while having little utility for BM25, whose computations are fairly cheap. This bottleneck was observed under BM25 by Dimopoulos *et al.* [14]. Similarly, LBMM contributes no gain under BM25, since BM25 scores are term-bounded and the block bounds are not much tighter than the term bounds in practice [30]. Notably, DBMM delivers a sizable speedup (i.e., over 2×) under BM25 with $k = 10$, due to its effective DocID-oriented NLB optimization. This gain disappears with $k = 1000$, however, similarly to the other MaxScore variants.

Besides BM25 (and F2EXP, which displays similar results), Figure 3 demonstrates large gains that LBMM, IBMM, and DBMM consistently deliver over the MaxScore baseline. Interestingly, IBMM and DBMM, which both leverage interval-based pruning, deliver noticeable gains across various ranking models. For $k = 10$, the gains of DBMM are clearly superior under PL2 and SPL, but not LMDir. For $k = 1000$, the gap greatly narrows between IBMM and DBMM. In fact, the pattern even reverses under SPL and LMDir. Digging deeper to explain this, we see that IBMM and DBMM differ in a crucial way. When DBMM scores a specific interval, it uses MaxScore-based (global) term partitioning, which is based on the term upper bounds. In contrast, IBMM pays the added cost of per-interval partitioning, which pays off when larger savings are possible (i.e., with $k = 1000$) or when per-interval static scores can affect the essential terms (i.e., under LMDir).

We now shift our attention to DBMW, the fastest strategy from §4. Despite the reliance of DBMM and DBMW on the same Block-Max metadata, DBMW is clearly superior in efficiency and robustness. To explain, DBMM's disadvantage can be thought of as a product of insufficiently aggressive pruning *and* relatively high overhead per pivot document. More precisely, we identify the following three deficiencies in DBMM that prevent it from realizing the potential of low-overhead MaxScore-based pruning hypothesized in §4. First, besides *dead* blocks bypassed by its NLB optimization, DBMM uses MaxScore's term partitioning for pivot selection. Second, while DBMM uses block-based bounds to rule out unpromising pivots, it does so at the granularity of a document, bearing the cost of a series of filters for every pivot document selected. That threshold $\theta$ is updated relatively infrequently renders these repeated per-document filters rather redundant across documents sharing the same block bounds. Lastly, while avoiding WAND's pessimistic pruning, DBMM suffers from MaxScore's overly *eager* document evaluation. In particular, DBMM's pre-evaluation filters are *postings-agnostic*. That is, if a document $d$ is marked for evaluation, DBMM computes and accumulates individual term contributions $s(t, d)$ concurrently with its movement of the pointers in the postings of query terms. As a result, within any DocID-oriented block where the sum of block bounds exceeds $\theta$, DBMM must compute at least one score per document containing an essential term before it can possibly rule out that candidate. Empirically, the first

| | MaxScore | LBMM | IBMM | DBMM | DBMW |
|---|---|---|---|---|---|
| **BM25** | 31 (89) | 29 (112) | 83 (359) | 14 (45) | **13** (**44**) |
| **F2EXP** | 53 (145) | 58 (191) | 98 (403) | 35 (151) | **31** (**126**) |
| **PL2** | 187 (723) | 139 (566) | 101 (437) | 39 (167) | **35** (**141**) |
| **SPL** | 506 (2156) | 353 (1611) | 131 (566) | 120 (539) | **58** (**245**) |
| **LMDir** | 679 (3453) | 656 (3482) | **146** (**627**) | 371 (2200) | 159 (746) |

**Table 3: Mean and tail top-10 query latency for results of Figure 3. As in the figure, this experiment uses an EF index with the default document order.**

| | MaxScore | LBMM | IBMM | DBMM | DBMW |
|---|---|---|---|---|---|
| **BM25** | 27 (90) | 20 (82) | 70 (328) | 11 (**30**) | **9** (31) |
| **F2EXP** | 46 (143) | 49 (195) | 84 (363) | 22 (81) | **19** (**79**) |
| **PL2** | 191 (750) | 118 (523) | 81 (360) | **20** (**68**) | 24 (96) |
| **SPL** | 514 (2236) | 303 (1310) | 98 (419) | 52 (192) | **40** (**178**) |
| **LMDir** | 678 (3325) | 608 (3347) | 114 (508) | 203 (1070) | **106** (**491**) |

**Table 4: Mean and tail top-10 query latency after URL-ordering the documents in the EF index.**

deficiency manifests itself in terms of increased number of pivot selections relative to DBMW, the second in terms of increased cost per pivot document, and the third in terms of increased number of score computations.

In addition to the main results in Figure 3, the Tables 3 and 4 report mean and tail latency for the MaxScore family and DBMW under crawl and URL ordering, respectively. Similar to the WAND family, the results show that the fastest method in terms of mean latency is often the fastest (or very close to the fastest) at the tail. Beyond this, the tables strongly confirm DBMW's rather consistent advantage over the best methods in the MaxScore family.

**Lessons Learned.** The results in this section confirm the importance of *block-based* pruning and further highlight the difficulty of their effective use. Additionally, we observe the following:

(1) ***Aggressive* skipping is challenging**: Owing to the *eager* nature of MaxScore and its more coarse-grained pruning paradigm, it appears more challenging to effectively use Block-Max within MaxScore-based strategies. For instance, while BMW and LBMW considerably outperform WAND in §4, the corresponding improvements of BMM and LBMM (and even IBMM) over canonical MaxScore fade in comparison. While DBMM demonstrates good latency results, it is not always competitive with DBMW.

(2) **Local *DocID-based* pivot selection is promising**: The results emphasize a tradeoff between DBMM's low-overhead pruning and IBMM's more aggressive interval processing. While the former excels under "cheap" models, the latter fares better in more expensive scenarios. It is thus natural to ask if we could capitalize on DBMM's mutually aligned blocks to bridge this gap.

Overall, our comparison of ten query evaluation strategies shows that DBMW is the current state-of-the-art strategy, with remarkable efficiency and robustness. Still, the findings of this section leave something to desire. Despite the importance of low-overhead pruning observed in §4, none of the five MaxScore variants considered in this section delivers on the expectation that computationally-simpler MaxScore equipped with Block-Max metadata should outperform the "pessimistic" WAND-based pruning. Next, we describe LazyBM, a top-$k$ retrieval strategy that precisely fulfills this promise.

## 6 BEST OF BOTH WORLDS: LAZYBM

We now describe LazyBM, a top-$k$ retrieval strategy that leverages the lessons learned in §4 and 5. In contrast to MaxScore's *eager* evaluation and WAND's *pessimistic* pruning, our proposed scheme is *lazy*: it does as little work as possible to find promising pivot

**Algorithm 1 Per-Block LazyBM Processing**

**Input:** Query Terms $Q$ (ordered by decreasing $df$), Current DocID-based Block $b$, Block Bounds $U$, Heap $topK$
**Output:** Heap $topK$
$P \leftarrow$ computePrefixSum($U$)
$T_{opt} \leftarrow \{t : P[t] \leq \theta\}$; $T_{ess} \leftarrow Q - T_{opt}$
**while** $d \leftarrow T_{ess}$.nextDocWithinBlock($b$) :
  $ub \leftarrow s(d)$
  **for** $t \in T_{ess}$: **if** $t$.docID() == $d$: $ub$ += $U[t]$
  **for** $t \in T_{opt}$ :
    **if** $ub > \theta$ or $ub + P[t] \leq \theta$: break
    **if** $t$.docID() < $d$: $t$.skipTo($d$)
    **if** $t$.docID() == $d$: $ub$ += $t$.blockUB()
  **if** $ub > \theta$ :
    $score \leftarrow s(d)$
    **for** $t \in T_{ess}$: **if** $t$.docID() == $d$: $score$ += $s(t, d)$
    **for** $t \in T_{opt}$ :
      **if** $ub + P[t] \leq \theta$: break
      **if** $t$.docID() < $d$: $t$.skipTo($d$)
      **if** $t$.docID() == $d$: $score$ += $s(t, d)$
    **if** $score > \theta$: $topK$.insert($d$, $score$)

document (i.e., like MaxScore), yet once it identifies a pivot it tries to avoid evaluating its score if at all possible (i.e., like WAND).

Among the strategies described so far, LazyBM is most similar to DBMM. In particular, like DBMM, LazyBM uses DocID-oriented Block-Max and adopts a MaxScore-based control flow. For this reason, it is useful to describe LazyBM's pruning as a series of three transformations that address the shortcomings of DBMM highlighted in §5. These transformations aim to reduce the overhead of pruning, specifically via lowering the cost of processing a selected pivot and an evaluated document, while simultaneously increasing the degree of skipping, specifically by exploiting the alignment of the DocID-oriented bounds.

**Amortized Pivot Filtering**: To reduce the computational cost per *selected pivot*, LazyBM amortizes the computation of the sum of block-based upper bounds at the granularity of a DocID-oriented block. Upon crossing a block boundary, LazyBM computes a *prefix sum* of the term's block bounds (with the terms sorted in decreasing order of their document frequencies ($df$s)). Naturally, a block is bypassed if the total bound falls short of $\theta$. Otherwise, LazyBM uses this prefix sum for lookup-based pruning when evaluating individual documents. §7 refers to this simple optimization over DBMM as LazyBM-A.

**Balanced Pruning**: To minimize the cost of score computations per *evaluated document*, LazyBM adopts an optimization that enables it to swiftly yet aggressively utilize the block bounds for

pruning. In particular, LazyBM *layers* WAND's relatively expensive pruning heuristic *on top of* MaxScore's cheaper but less strict heuristic. That is, once LazyBM selects a pivot document $d$ based from the union of essential terms, it computes a WAND-inspired postings-informed bound on $d$'s score prior to computing any term contributions $s(t, d)$ (also prior to decompressing any posting's term frequency). Document $d$ is evaluated only if this relatively tight upper bound exceeds $\theta$. We refer to LazyBM with both amortized pivot selection and balanced pruning as LazyBM-AB.

**Local Term Partitioning**: To cut down on the number of *pivot selections*, LazyBM *local*-partitions the terms into essential and non-essential with respect to each individual block. While IBMM incorporates a similar optimization by running MaxScore on top of its runtime-generated interval metadata, this requires an expensive preprocessing step that offsets most of the potential gains as observed in §5. In contrast, LazyBM capitalizes on the natural alignment of the DocID-oriented blocks within DBMM to cheaply incorporate local MaxScore-based pruning. §7 refers to the algorithm implementing all three mentioned optimizations as LazyBM.

Algorithm 1 summarizes LazyBM's processing of an individual DocID-oriented block. For each block $b$, LazyBM is provided $U$, the block bounds for the query terms $Q$. The algorithm first *locally* partitions the terms into essential and optional with respect to the current block (Lines 3 and 4). Through the essential terms, it repeatedly identifies the next pivot document $d$ within $b$ applying the MaxScore locally to $b$ (Line 5). For each pivot document $d$, LazyBM computes a *postings-informed* upper bound (Lines 6–11), the tightest bound that can be obtained on $d$ short of computing term contributions. This filter is terminated once the outcome of comparing the bound against $\theta$ is known (Line 9). If $d$ passes both the MaxScore-based (Line 5) and WAND-based filters (Line 12), it proceeds to the main computation stage (Lines 13–18). This computes $d$'s score precisely as in BMM, and if $d$'s score exceeds $\theta$, it is inserted to the top-$k$ heap (Line 19).

## 7 EVALUATION OF LAZYBM

In this section, we experimentally evaluate LazyBM in detail against DBMW, the best performing strategy from §4 and 5.[11] In particular, we address the following research question:

**RQ$_4$**: How does LazyBM compare against state-of-the-art DBMW across settings? If there are gains, what is their source?

Figures 4 compares LazyBM against DBMW in terms of *mean* and *tail* query latency. At the tail, we show latency at the 95th percentile of queries; we observe a similar pattern at each of the 90th and 99th percentiles. The figure demonstrates LazyBM's superiority to DBMW's already-remarkable performance across both mean and tail query latency with $k = 10$ and $k = 1000$. LazyBM's advantage can be seen across ranking models, with the gap widening under the more expensive ones. Relative to DBMW, LazyBM delivers up to 1.6×, 1.3×, 3.5×, 4.4×, and 4.7× speedup under BM25, F2EXP, PL2, SPL, and LMDir, respectively. On average, LazyBM reduces mean and tail latency by 1.9× and 2.2×, respectively.

Taking a closer look, we find that, while both strategies evaluate essentially the same number of documents and apply a comparable

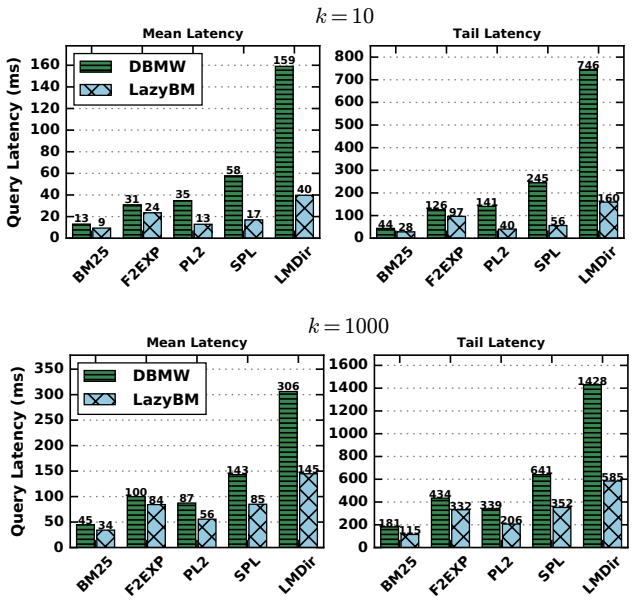


**Figure 4: Mean and tail query latency for $k = 10$ (top) and $k = 1000$ (bottom), comparing LazyBM against DBMW across five ranking models.**

number of score computations on average, LazyBM consistently excels in reducing the number of pivot documents selected and the number of pointer movements. That is, while LazyBM and DBMW both reap the benefit of a WAND-based reduction in the number of documents evaluated, LazyBM sets itself apart by additionally exploiting MaxScore-based pruning to quickly identify those documents that are sufficiently promising to apply the (rather more expensive) WAND-based filter.

To better understand how the speedup is achieved, Figure 6 demonstrates the breakdown of gains in latency relative to DBMM as we apply the three main optimizations of LazyBM. For reference, the figure also shows the latency of DBMW. The latencies reported are averaged across all five ranking models and both values of $k$.[12]

To begin with, we can see that LazyBM-A outperforms DBMM by about 1.3× in terms of mean and tail latency. While LazyBM-A cost reduction per selected pivot boosts efficiency across all ranking models (detailed results not shown), we see that it contributes most of the gains over DBMW under the cheap BM25 and F2EXP models and only limited gains under PL2, SPL, and LMDir. Since PL2, SPL, and LMDir exhibit expensive $s(t, d)$ computations, LazyBM-AB's balanced pruning, particularly its WAND-based filter, considerably reduces their query latency. As a result, LazyBM-AB results in 2.2× and 2.5× average speedup against DBMM in terms of mean and tail query latency, respectively, and begins to outperform the state-of-the-art strategy DBMW. When incorporating all three optimizations together, LazyBM gains further advantage over DBMM and DBMW due to its *low-overhead* local utilization of the block

[11] In our experiments, we find DBMW to retain its advantage as the fastest strategy across mean latency, tail latency, and latency across query lengths.

[12] To aggregate the (mean or tail) query latencies across models, we compute the *geometric mean* of the individual latencies (similar to the TPC-D benchmark [3]). Intuitively, this assigns equal weight to all settings and avoids skewing the averages towards the more "expensive" settings (e.g., LMDir with $k = 1000$), wherein LazyBM's advantage is highest.

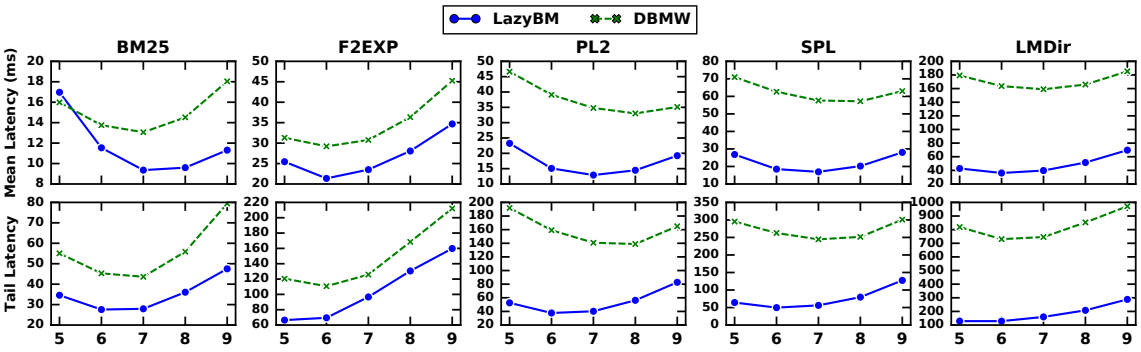


**Figure 5: Mean (top) and tail (bottom) query latency of LazyBM and DBMW as a function of the DocID-oriented block size with $k = 10$. Results with $k = 1000$ demonstrate a similar pattern.**

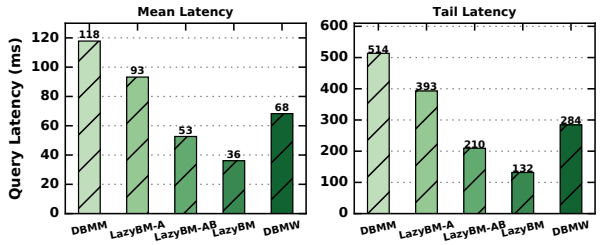


**Figure 6: Mean (left) and tail (right) query latency of LazyBM broken down by optimization against DBMM and DBMW.**

bounds, which allows it to better leverage the MaxScore heuristic, and accordingly reduce the number of pivot selections substantially.

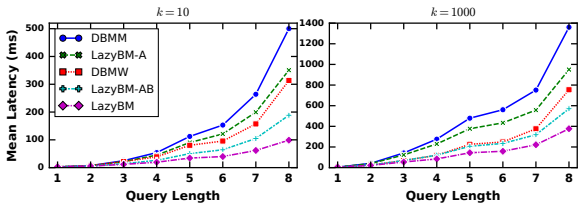


**Figure 7: Mean query latency as a function of query length for DBMM, LazyBM and its variants LazyBM-A and LazyBM-AB, and DBMW.**

Mean query latency is also shown broken down by query length in Figure 7, geometrically-averaged across models. While LazyBM and DBMW are competitive with each other for short queries (i.e., those of length at most three terms), LazyBM increasingly picks up for longer queries.

Subsequently, to verify that LazyBM's advantage is robust across various memory budgets, Figure 5 compares LazyBM against DBMW across a wide spectrum of block sizes. At one end, with block size $2^5$, both strategies require nearly 16GiBs for the Block-Max metadata. At the other end with $2^9$, both require only 1GiB. As the figure shows, LazyBM demonstrates consistent gains across the entire range in terms of both mean and tail latency.

Lastly, Table 5 confirms LazyBM's robust advantage over DBMW across our auxiliary experiments' index settings.

| | VarintG8IU Index | | URL-ordered Index | |
|---|---|---|---|---|
| | **DBMW** | **LazyBM** | **DBMW** | **LazyBM** |
| **BM25** | 12 (38) | **9 (26)** | 9 (31) | **7 (20)** |
| **F2EXP** | 30 (129) | **22 (95)** | 19 (79) | **13 (47)** |
| **PL2** | 31 (124) | **12 (38)** | 24 (96) | **9 (24)** |
| **SPL** | 53 (230) | **17 (56)** | 40 (178) | **18 (61)** |
| **LMDir** | 147 (691) | **38 (156)** | 106 (491) | **27 (109)** |

**Table 5: Mean and tail top-10 latency upon switching to VarintG8IU index encoding and to URL re-ordering, respectively. The results are consistent with Figure 4.**

## 8 CONCLUSIONS AND FUTURE WORK

In this paper, we investigated strategies for efficient top-$k$ document retrieval. We focused on the *robustness* of said strategies, particularly how they perform under representative ranking models and values of $k$. We conducted an extensive empirical comparison between ten strategies, many of which were never compared before to our knowledge. Based on a careful analysis of the results, we proposed LazyBM, a remarkably simple query evaluation strategy that bridges the gap between the best performing WAND-based and MaxScore-based approaches. Experimentally, LazyBM greatly *and* consistently outperforms all of the considered strategies across ranking models and values of $k$ in terms of both mean and tail query latency. Moreover, its gains are robust to memory budget, query length, and index configurations.

Future work will examine how LazyBM can contribute to end-to-end effective and efficient retrieval, specifically by expanding the set of ranking models that can be employed for top-$k$ retrieval. While we used a representative Web collection to run our experiments (i.e., the most recent ClueWeb collection), future work will also consider how top-$k$ retrieval strategies behave on top of different collection types (e.g., microblogs or news).

## ACKNOWLEDGMENTS

We thank Yousuf Ahmad, Reem Suwaileh, and Mucahid Kutlu for valuable discussions and insights. This publication was made possible by NPRP grant# NPRP 7-1330-2-483 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

# REFERENCES


[1] Giambattista Amati. 2006. Frequentist and bayesian approach to information retrieval. In *European Conference on Information Retrieval*. Springer, 13–24.

[2] Gianni Amati and Cornelis Joost Van Rijsbergen. 2002. Probabilistic models of information retrieval based on measuring the divergence from randomness. *TOIS* (2002).

[3] Ramesh Bhashyam. 1996. TPC-D—the challenges, issues and results. *ACM SIGMOD Record* 25, 4 (1996), 89–93.

[4] Edward Bortnikov, David Carmel, and Guy Golan-Gueta. 2017. Top-k query processing with conditional skips. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 653–661.

[5] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *CIKM*.

[6] Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. 2011. Interval-based pruning for top-k processing over compressed lists. In *ICDE*.

[7] Stéphane Clinchant and Eric Gaussier. 2010. Information-based models for ad hoc IR. In *SIGIR*.

[8] Kevyn Collins-Thompson, Paul N. Bennett, Fernando Diaz, Charlie Clarke, and Ellen M. Voorhees. 2014. TREC 2013 Web Track Overview. In *TREC*.

[9] Matt Crane, J Shane Culpepper, Jimmy Lin, Joel Mackenzie, and Andrew Trotman. 2017. A comparison of Document-at-a-Time and Score-at-a-Time query evaluation. In *WSDM*.

[10] Caio Moura Daoud, Edleno Silva de Moura, Andre Carvalho, Altigran Soares da Silva, David Fernandes, and Cristian Rossi. 2016. Fast top-k preserving query processing using two-tier indexes. *Information Processing & Management* 52, 5 (2016), 855–872.

[11] Caio Moura Daoud, Edleno Silva de Moura, David Fernandes, Altigran Soares da Silva, Cristian Rossi, and Andre Carvalho. 2017. Waves: a fast multi-tier top-k query processing algorithm. *Information Retrieval Journal* 20, 3 (2017), 292–316.

[12] Jeffrey Dean. 2009. Challenges in building large-scale information retrieval systems: invited talk. In *WSDM*. 1–1.

[13] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. A candidate filtering mechanism for fast top-k query processing on modern cpus. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 723–732.

[14] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. Optimizing top-k document retrieval strategies for block-max indexes. In *WSDM*.

[15] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *SIGIR*.

[16] Hui Fang and ChengXiang Zhai. 2005. An exploration of axiomatic approaches to information retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 480–487.

[17] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Zien. 2011. Evaluation strategies for top-k queries over memory-resident inverted indexes. *VLDB* (2011).

[18] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 55–64.

[19] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L Cox, and Scott Rixner. 2014. Predictive parallelization: Taming tail latencies in web search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 253–262.

[20] Jimmy Lin and Andrew Trotman. 2015. Anytime ranking for impact-ordered indexes. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*. ACM, 301–304.

[21] Jimmy Lin and Andrew Trotman. 2017. The role of index compression in score-at-a-time query evaluation. *Information Retrieval Journal* 20, 3 (2017), 199–220.

[22] Claudio Lucchese, Franco Maria Nardini, Raffaele Perego, Salvatore Orlando, and Salvatore Trani. 2018. Selective Gradient Boosting for Effective Learning to Rank. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 155–164.

[23] Craig Macdonald, Iadh Ounis, and Nicola Tonellotto. 2011. Upper-bound approximations for dynamic pruning. *ACM Transactions on Information Systems (TOIS)* 29, 4 (2011), 17.

[24] Craig Macdonald and Nicola Tonellotto. 2017. Upper Bound Approximation for BlockMaxWand. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*. ACM, 273–276.

[25] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2017. Efficient & effective selective query rewriting with efficiency predictions. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 495–504.

[26] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster blockmax wand with variable-sized blocks. In *SIGIR*.

[27] Antonio Mallia, Michał Siedlaczek, Joel Mackenzie, and Torsten Suel. 2019. PISA: Performant Indexes and Search for Academia. (2019).

[28] Antonio Mallia, Michał Siedlaczek, and Torsten Suel. 2019. An experimental study of index compression and DAAT query processing methods. In *European Conference on Information Retrieval*. Springer, 353–368.

[29] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. 2016. Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 83–84.

[30] Matthias Petri, J Shane Culpepper, and Alistair Moffat. 2013. Exploring the magic of WAND. In *Proceedings of the 18th Australasian Document Computing Symposium*. ACM, 58–65.

[31] Matthias Petri, Alistair Moffat, Joel Mackenzie, J Shane Culpepper, and Daniel Beck. 2019. Accelerated Query Processing Via Similarity Score Prediction. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 485–494.

[32] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at TREC-3. *NIST Special Publication* (1995).

[33] Oscar Rojas, Veronica Gil-Costa, and Mauricio Marin. 2013. Distributing efficiently the Block-Max WAND algorithm. *Procedia Computer Science* (2013).

[34] Oscar Rojas, Veronica Gil-Costa, and Mauricio Marin. 2013. Efficient parallel block-max WAND algorithm. In *European Conference on Parallel Processing*. Springer, 394–405.

[35] Dongdong Shan, Shuai Ding, Jing He, Hongfei Yan, and Xiaoming Li. 2012. Optimized top-k processing with global page scores on block-max indexes. In *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, 423–432.

[36] Alexander A Stepanov, Anil R Gangolli, Daniel E Rose, Ryan J Ernst, and Paramjit S Oberoi. 2011. SIMD-based decoding of posting lists. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 317–326.

[37] Nicola Tonellotto, Craig Macdonald, Iadh Ounis, et al. 2018. Efficient Query Processing for Scalable Web Search. *Foundations and Trends® in Information Retrieval* (2018).

[38] Howard Turtle and James Flood. 1995. Query evaluation: strategies and optimizations. *IP & M* (1995).

[39] Sebastiano Vigna. 2013. Quasi-succinct Indices. In *WSDM*.

[40] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *SIGIR*.

[41] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible ranking baselines using Lucene. *JDIQ* (2018).

[42] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. 2016. Ranking relevance in yahoo search. In *SIGKDD*.

[43] Chengxiang Zhai and John Lafferty. 2004. A study of smoothing methods for language models applied to information retrieval. *TOIS* (2004).