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
mhammoud@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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:


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 1.9x

1https://github.com/okhat/LazyBM
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 \( c(d) \), combined as follows:

\[
S(d, Q) = s(d) + \sum_{t \in Q} w_t \cdot s(t, d) \tag{1}
\]

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 \( k \)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 BM25 [15], LBWM [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.


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, Dasoud 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, 30] 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 BM25-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).

²Since 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 VBWM is based on the code by Mallia et al. [26]. Our implementation of BMW, DBWM, and DBMM is based on the code by Dimopoulos et al. [14]. We implement LBMW, LBM, and IBMM similarly.

For the postings-oriented Block-Max strategies (i.e., all but the DocID-oriented DBWM, DBMW, 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 VBWM, whose variable block sizes are controlled indirectly by a parameter λ, we apply binary search over λ to obtain average block size of 64 ± 1 over the terms of our query set.

For the DocID-oriented strategies, we apply quantization and on-the-fly-generation under the “variable bound” setting recommended by Dimopoulos et al. [14], starting with blocks that span 128 document IDs for terms with over 218 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, 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 stop words.

Our index is compressed using the popular Elias-Fano (EF) encoding [39], using Facebook’s optimized folly implementation, 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.

![Figure 1: Cost of exhaustive DAAT across models.](image)

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

We implement Porter2 [35], using Facebook’s optimized folly implementation, 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.

and the ds2i libraries. We run these auxiliary experiments on another VM that has 32GB of RAM but is otherwise identical in its specifications. 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 RQ1 stated as follows:

**RQ1:** 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 \( U_t := \max_{s \in Q} s(t, d) \) 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 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 \( U_{t,b} := \max_{s \in \text{postings}} s(t, d) \) 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

---

1[https://github.com/rossanoventurini/Variable-BMW]
2[https://github.com/dimopoulos/WSDM13]
3[https://trec.nist.gov/data/million.query.html]
4[https://github.com/facebook/folly]
5[https://github.com/lemire/FastPFor]
pruning. On top of WAND’s pivot selection, BMW skips any document 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, Dimopoulou 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.

![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.](image-url)
Table 1: Mean and tail top-10 query latency after URL-ordering the documents in the EF index.

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

Table 2: Mean and tail top-10 query latency after URL-ordering the documents in the default (crawl-based) document order.

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

5 EVALUATION OF MAXSCORE VARIANTS

In this section, we describe and evaluate five representative members of the MaxScore family, examining RQ$_2$ and RQ$_3$ stated below.

**RQ$_2$:** How do the MaxScore-based strategies compare against WAND, the fastest among the WAND family?

**RQ$_3$:** How do the MaxScore-based 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 $k$.\footnote{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).}

**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 Dimopoulos et al. [14] 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, DBMM relies on DocID-based blocks for pruning. Similar to BMM and LBMM, DBMM selects its pivot...
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 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-\( k \) 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 DBMM, the fastest WAND-based strategy from §4. In our results, BMM’s speedup over MaxScore is negligible on average (with maximum speedup of 1.1x), 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 BM25 and LBMM 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 2x) 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 DBMM, 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
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 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.

### Algorithm 1 Per-Block LazyBM Processing

1. **Input:** Query Terms $Q$ (ordered by decreasing $df$), Current DocID-based Block $b$, Block Bounds $U$, Heap $topK$

2. **Output:** Heap $topK$

3. $P \leftarrow$ computePrefixSum($U$)

4. $Topt \leftarrow \{ t : |P[t| \leq \theta \}; T_{ess} \leftarrow Q - T_{opt}$

5. while $d \in T_{ess}$, nextDocWithinBlock$(b)$:

6. $ub \leftarrow s(d)$

7. for $t \in T_{ess}$:

8. if $t.docID() == d$: $ub \leftarrow U[t]$

9. for $t \in T_{opt}$:

10. if $ub > \theta$ or $ub + P[t] \leq \theta$ break

11. if $t.docID() < d$: $t.skipTo(d)$

12. if $t.docID() == d$: $ub \leftarrow t.blockUB()$

13. $score \leftarrow s(d)$

14. for $t \in T_{ess}$:

15. if $t.docID() == d$: $score \leftarrow s(t, d)$

16. for $t \in T_{opt}$:

17. if $ub + P[t] \leq \theta$ break

18. if $t.docID() < d$: $t.skipTo(d)$

19. if $t.docID() == d$: $score \leftarrow s(t, d)$

20. if $score > \theta$: $topK$.insert($d$, $score$)

### 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.

<table>
<thead>
<tr>
<th></th>
<th>MaxScore</th>
<th>LBMM</th>
<th>IBMM</th>
<th>DBMM</th>
<th>DBMW</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>31 (89)</td>
<td>29 (112)</td>
<td>83 (359)</td>
<td>14 (45)</td>
<td>13 (44)</td>
</tr>
<tr>
<td>F2EXP</td>
<td>53 (145)</td>
<td>58 (191)</td>
<td>98 (403)</td>
<td>35 (151)</td>
<td>31 (126)</td>
</tr>
<tr>
<td>PL2</td>
<td>187 (723)</td>
<td>139 (566)</td>
<td>101 (437)</td>
<td>39 (167)</td>
<td>35 (141)</td>
</tr>
<tr>
<td>SPL</td>
<td>506 (2156)</td>
<td>333 (1611)</td>
<td>131 (566)</td>
<td>120 (539)</td>
<td>58 (245)</td>
</tr>
<tr>
<td>LMDir</td>
<td>679 (3453)</td>
<td>656 (3482)</td>
<td>146 (627)</td>
<td>371 (2200)</td>
<td>159 (746)</td>
</tr>
</tbody>
</table>

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

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

### Lessons Learned

- **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 BM and LBMW considerably outperform WAND in §4, the corresponding improvements of BM and LBMW (and even IBMM) over canonical MaxScore fade in comparison. While DBMM demonstrates good latency results, it is not always competitive with DBMW.

- **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
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:

**RQ4:** 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, the skew 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 $1.6\times, 1.3\times, 3.5\times, 4.4\times$, and $4.7\times$ speedup under BM25, F2EXP, PL2, SPL, and LMDir, respectively. On average, LazyBM reduces mean and tail latency by $1.9\times$ and $2.2\times$, respectively.

Taking a closer look, we find that, while both strategies evaluate essentially the same number of documents and apply a comparable 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 $1.3\times$ 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-A’s balanced pruning, particularly its WAND-based filter, considerably reduces their query latency. As a result, LazyBM-AB results in $2.2\times$ and $2.5\times$ 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

\[\text{To aggregate the (mean or tail) query latencies across models, we compute the geometric mean of the individual latencies (similar to the TPC-D benchmark \cite{3}). Intuitively, this assigns equal weight to all settings and avoids skewing the averages towards the more "expensive" settings (e.g., LMDir with } k = 1000\text{), wherein LazyBM’s advantage is highest.} \]
bounds, which allows it to better leverage the MaxScore heuristic, and accordingly reduce the number of pivot selections substantially.

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.

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 Youssuf 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


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


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

