Window Navigation with Adaptive Probing for Executing BlockMax WAND

Jinjin Shao, Yifan Qiao, Shiyu Ji, Tao Yang
Department of Computer Science, University of California
Santa Barbara, California, USA

ABSTRACT

BlockMax WAND (BMW) and its variants can effectively prune low-scoring documents for fast top-k disjunctive query processing. This paper studies a boosting approach that further accelerates document retrieval by executing BMW, or one of its variants, on a sequence of posting windows with an order prioritized to tighten the threshold bound earlier. This optimization could add benefits to safely eliminate more operations involved in posting block visitation and document score evaluation. This paper evaluates such index navigation for BMW and two of its variants.

KEYWORDS

Top-k Query Evaluation, Dynamic Pruning, Efficiency

ACM Reference Format:
Jinjin Shao, Yifan Qiao, Shiyu Ji, Tao Yang. 2021. Window Navigation with Adaptive Probing for Executing BlockMax WAND. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3404835.3463109

1 INTRODUCTION

Document retrieval for top-k search with disjunctive query semantics identifies all documents containing at least one query term , and it often uses a simple additive linear ranking to select top-k results for later-stage re-ranking. This initial stage of query processing is time-consuming in searching over a large document set. Dynamic pruning techniques such as MaxScore [33], WAND [2], and Block-Max WAND (BMW) [12] can greatly speed up the top-k document retrieval process by skipping the evaluation of documents that are unable to appear in the final top-k results. As posting lists are often compressed in blocks, BMW [12] stores the maximum score per block and leverages such scores to skip unnecessary decompression and inspections of posting blocks. There are various improvements following the work of BMW (e.g. [6, 10, 16, 17, 20–23, 25, 26, 29, 31]) and the next section discusses some of them.

Pruning of posting block visitations or document evaluations happens if the top-k threshold lower bound is higher than the estimated ranking score upper bounds for documents, and the effectiveness of pruning heavily depends on the tightness of the updated threshold lower bound. BMW and its variants visit the posting lists typically following an increasing order of document IDs. The question posed in this paper is whether posting visitation order can be changed to examine query-specific high-scoring documents first, which naturally tightens the threshold bound earlier and thus helps to skip more operations.

The main contribution of this paper is a complementary scheme to safely accelerate BMW or its variants by changing their way of visiting the index through *window navigation with adaptive probing*. Without changing the internal core structure of BMW or its variants, we consider document retrieval flows through document ID windows and prioritizes a subset of windows that may hold high-scoring documents for a given query. That results in earlier pruning of many windows and the retrieval only navigates through unpruned windows while following a dynamically-selected order. We have evaluated the application of our technique in BMW with uniform block sizes, variable-sized BMW [22], and DBMW with document-ID-oriented blocks [10, 11]. Our evaluation finds that the proposed technique can significantly reduce the query time for short queries with 2 and 3 terms, and a certain range of k values.

2 DEFINITIONS AND BACKGROUND

The top-k search problem uses an *inverted index* with a set of terms, and a *document posting list* of each term represents documents that contain such a term. *Ranking* for document retrieval uses a simple additive formula based on term features of each document which is $RankScore(d) = \sum_{t \in Q} w_t S(t, d)$, where Q is the set of all search terms, $S(t, d)$ is the feature score of term t for document d, and w_t is the weight for the corresponding term. An example of the additive formula is widely used BM25 [15]. A posting record of each term contains document ID and its term feature score.

BMW [12] assumes that a posting list, sorted in ascending order of document IDs, is compressed and stored in blocks. Some information of a posting block p can be accessible without decompression, and such information contains the maximum weighted term feature score among all documents and the maximum document ID in this block, denoted as $BlockMax(p)$ and $MaxDocID(p)$.

Our work is inspired by the candidate filtering approach [10, 24] which divides the scope of document IDs visited during document retrieval as a sequence of equal-sized windows. A query processing scheme to call a Live Block computation scheme that detects if a window can be pruned. Window pruning is related to interval-based pruning in [3]. These techniques prune a window or an interval when the estimated score upper bound is below the top-k threshold, and a similar technique is used in the local Block-Max method [29]. Following the posting block window setting of [10, 11], to further improve efficiency, LazyBM in [17] combines the WAND-based and MaxScore [33]-based pruning. The major difference of our effort compared to the above work is that we do not execute

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07.
https://doi.org/10.1145/3404835.3463109

the document retrieval strictly in a sorted order of document IDs, and we dynamically detect and prioritize windows that may hold high-scoring documents. The work in [3] proposes to use a sorted order of its intervals in terms of interval score upper bound, which has an expensive sorting and execution logic, and outweighs its benefits as verified in [17]. Our work uses a fast selection strategy in a linear complexity to identify a set of windows that hold high scoring documents while we do not explicitly sort them.

Our work is orthogonal to the previous work that improves BMW as we can use a variation of BMW in our algorithm instead of the original BMW. One class of improvement is to estimate top-k threshold in advance before running BMW based on sampling, feature-based prediction, and past queries [9, 16, 25, 26, 26, 34]. The second class is to use a tiered index [7, 8, 28]. The third class is to improve pruning condition [1, 17]. The fourth class is to support different block structures: VBMW [22] uses variable sized blocks to reduce the gap between the *MaxScore* value and the actual average document term weight in a block, and DBMW [10, 11] uses document-ID-oriented posting blocks and each of them covers a fixed number of consecutive document IDs. We will demonstrate the use of our work for VBMW and DBMW in addition to BMW.

3 INDEX NAVIGATION WITH PROBING

This section presents our proposed window-driven index navigation with adaptive probing to execute BMW, denoted as BMW^p. Besides the original BMW, a variant of BMW can also be applied in each phase of the algorithm described below. All returned documents satisfy the top-k scoring constraint with safe pruning.

Like [10], we divide the whole range of searchable document IDs into a disjoint set of document ID windows, and unlike [10], these windows are not equal-sized. A posting block p intersects with a window W, denoted by $p \in W$, if any document ID in this window is between $MinDocID(p)$ and $MaxDocID(p)$ (both inclusive). Given a set of posting blocks B, the maximum window score of a window W is defined as $WinMax(W) = \sum_{p \in B \wedge p \in W} BlockMax(p)$.

We focus on block-aligned windows. Our evaluation adopts the boundary IDs of each window to be identical to $MaxDocID(p)$ of a block p in this window. Fig. 1 shows 6 windows whose boundaries only aligned with the maximum IDs of posting blocks for two terms. The reason is that if we can skip the visitation of a window, the cost of decompressing a block can be avoided. Alternatively, we can opt to have windows aligned with the $MinDocID(p)$ of each block. Unlike [3], we do not use both the minimum and maximum IDs of posting blocks as window boundaries. We intend to let the number of block-aligned windows be as small as possible because such a number affects the probing time cost significantly as shown later. Compared to BMW, BMW^p adds field $MinDoc(p)$ as a small extra space overhead for each posting block.

We define $\Omega(Z)$ as the top-k threshold found by running Block-Max WAND through top Z windows with the highest $WinMax$ values, for a given set of windows. Define $\Gamma(W)$ as the highest rank score of documents that appear in window W.

If we arbitrarily select k windows, at least k distinct documents can be returned, and their minimum rank score can be used as a lower bound of the final top-k threshold. Our goal is to find most promising windows and probe these windows with BMW to yield documents whose rank scores can be close to those of final top-k documents. With N being the total number of windows, we set parameter value Z to limit the number of such promising windows with the largest $WinMax$ values to be probed.

The question is, how to find a good value for Z? A smaller and the earlier derivation of the top-k threshold bound will create more opportunities to prune the unnecessary visitations of posting blocks and their documents. Z needs to be sufficiently larger than k since the highest-scoring document of the top window with the largest $WinMax$ value may not be included in the final top-k documents. It is challenging to find an optimal solution and our idea is to find a reasonable cutoff value Z which satisfies

$$\min_{Z}\{Z \geq \alpha * k \text{ such that } \Omega(Z) \geq WinMax(W^Z)\}$$

where α is a constant and W^Z is the window which has the Z-th largest $WinMax$ value. Let NT^Z denote the set of the remaining $N - Z$ windows which are not in top Z. The above inequality means Z is sufficiently large to discover a tight bound for top-k threshold θ. Then all windows in NT^Z can be pruned because

$$\Omega(Z) \geq WinMax(W^Z) \geq \max_{p \in NT^Z} WinMax(p) \geq \max_{p \in NT^Z} \Gamma(p).$$

To find the minimum value for Z following the above inequality efficiently, we first use an exponential probing strategy by guessing a series of cutoff point candidates $Z_1, Z_2, ..., Z_t$ where $Z_i = Z_1 * b^{i-1}$ and b is a constant. In our evaluation, we use $b = 8$ and $\alpha = 12.8$.

Description of BMW^p is presented as follows.

- Derive the block-aligned document ID windows based on the posting block metadata of searched terms.
- We let $Z_1 = \alpha k$ as a starting value, $Z_2 = Z_1 * b, \cdots$, and $Z_t = Z_1 * b^{t-1}$, until $Z_{t+1} > N$.
- Find the top Z_i window positions in the derived N windows where $1 \leq i \leq t$. That is done recursively as follows:
  - Find top Z_t windows among all N windows using a quick selection algorithm [5].
  - Recursively find top Z_i windows among top Z_{i+1} windows. This procedure ends when we find top Z_1 windows.
- Starting from Z_1, run BMW on posting blocks in selected windows Z_i from $i = 1$ to t until we find the smallest i value such that $\Omega(Z_i) \geq WinMax(W^{Z_i})$. Let Z_j be the smallest Z_i found.
- In case that the j value derived above is $j = t$, and it does not satisfy $\Omega(Z_j) \geq WinMax(W_{Z_j})$, the final pass is to set $\theta = \Omega(Z_j)$, and prune any un-probed window W in NT^{Z_j} if it satisfies $\theta > WinMax(W)$. Then execute BMW through all the surviving windows, and continue to update the top-k threshold θ. During this process, prune any window W satisfying $\theta > WinMax(W)$.

Notice that posting blocks in windows that are inspected by an earlier round of probing will be annotated with a special mark on their metadata, and will not be re-inspected in a later round when running BMW. Thus during the final pass that has to run BMW through the surviving $N - Z_j$ windows, windows that have been examined earlier will not be re-examined again.

Example. Fig. 1 shows retrieval navigation for a two-term query with 6 windows $W_1, \cdots, W_6$ in ascending order of document ID ranges. The descending order of windows in their $WinMax$ values is W_5, W_2, W_1, W_4, W_3 and W_6, even though the algorithm does not explicitly derive this order. The left side of this figure shows that BMW^p plans to probe top Z_1 and Z_2 windows where $Z_1 = 2$, $Z_2 = 4$, $k = 1$, and $\alpha = 2$. The right side shows the actual windows visited during retrieval. The top window locating step identifies W_2 and W_4 are the 2nd and 4th top positions. Round 1 probes top 2 windows W_2 and W_5. Round 2 probes the top 4 windows W_1, W_2, W_4, and W_5 while skipping W_2 and W_5 marked with a strike-through (since they have been probed), and their corresponding BMW execution is depicted by dotted arrows. The details on how to do skipping these windows with a strike-through are discussed in the next paragraph. For the final pass, assuming the derived threshold $\Omega(4) \geq WinMax(W_4)$, the two un-probed windows W_3 and W_6 are pruned immediately and marked with slashes because $\Omega(4) \geq \Gamma(W_3)$ and $\Omega(4) \geq \Gamma(W_6)$. That is inferred by the fact that $\Omega(4) \geq WinMax(W_4)$, and we also know that $WinMax(W_4) \geq \max(WinMax(W_3), WinMax(W_6))$. In summary, index navigation with probing selects and visits a total of 4 windows in the order of W_2, W_5, W_1, and W_4 to safely complete the document retrieval.

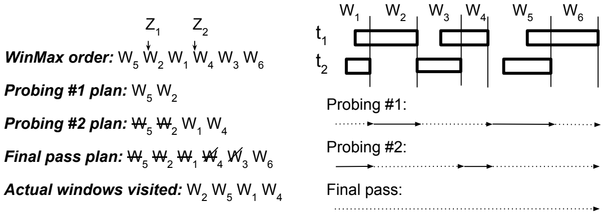
Z1
Z2
WinMax order: W5 W2 W1 W4 W3 W6
Probing #1 plan: W5 W2
Probing #2 plan: ~~W5~~ ~~W2~~ W1 W4
Final pass plan: ~~W5~~ ~~W2~~ ~~W1~~ ~~W4~~ W3 W6
Actual windows visited: W2 W5 W1 W4
W1 W2 W3 W4 W5 W6
t1
t2
Probing #1:
Probing #2:
Final pass:

Figure 1: Navigation of windows with 2-round probing

Modified BMW for inspecting only selected windows. We apply BMW as described in [12] with some changes in its two functions *Next(d)* and *NextShallow(d)*. Function *Next(d)* returns the first posting record whose document ID is no less than d. Function *NextShallow(d)*, described in Section 5.1 of [12], returns the first posting block whose maximum document ID is no less than d, without loading and decompressing the data of any block. To apply BMW on a specific set of windows, we first mark each posting block in all selected windows by examining the metadata of all posting blocks without any block data decompression. Next, we can apply BMW directly on posting blocks window by window from the beginning in ascending order of document IDs. The modified functions *Next(d)* and *NextShallow(d)* return the first posting record or posting block, skipping all marked posting blocks.

Time cost of BMW^p. For simplicity, the cost of index loading from a disk is not included. Notice that many search systems pre-load compressed index into memory [13]. The total time cost for multi-round probing plus the final pass using BMW is $O(N) + O(T_{BMW}^{Z_j}) + O(T_{BMW}^{N-Z_j})$. Here $T_{BMW}^{Z_j}$ is the time cost for running BMW through top Z_j windows with the largest $WinMax$ values and $T_{BMW}^{N-Z_j}$ is the time for running BMW through the remaining windows. The item $O(N)$ is the time cost for driving windows from the posting block metadata of searched terms, and for locating the top scored window positions $Z_1, Z_2, ..., Z_t$. Cost of window generation is linear to the number of windows, which is proportional to the number of posting blocks. Finding the first position Z_t is $O(N)$ with a $O(N)$ quick selection algorithm, and the next position Z_{t-1} costs $\frac{N}{b}$. With the recursion ended at Z_1, the total time to locate these t positions are $O(N + \frac{N}{b} + \frac{N}{b^2} + \cdots)$ which is less than $O(\frac{bN}{b-1})$.

Since the documents in windows are only visited during window-driven index navigation, the total cost of running BMW in BMW^p is bounded by the total time for running an original BMW through all windows in a document ID sorted order while BMW^p should skip more operations related to block visits and document evaluation.

4 EVALUATION

Datasets. The experiments are performed on the ClueWeb 2009 TREC category B with 33.6 million documents after filtering with Waterloo spam score [4] threshold 60. The header and body text for all documents is extracted using Indri [27]. All words are lowercased and stemmed using the Krovetz stemmer [18]. The inverted index is compressed using SIMD-BP128 [19]. We use TREC 2006 Efficiency Track topics where each query term has more than 100 postings. We randomly select 2500 queries containing two to six terms, where the length distributions are 16.8%, 28.2%, 27.7%, 17.3%, and 10.0%.

Testing environment. The algorithms evaluated are implemented in C++, and compiled with GCC 4.8.5 using the highest optimization settings. For the block partitioning in VBMW, we use the released code from [22]. The experiments are conducted with Intel Xeon E5-2680 v3 2.50GHz and 128GB memory. Query processing times are reported in milliseconds using one CPU core on average with multiple runs. Ranking follows the linear additive formula BM25 [15]. Before timing the queries, all compressed posting lists and metadata for tested queries are loaded into memory, and the previous work typically makes the same assumption [17, 22].

Baselines. We apply our scheme to BMW with uniform block sizes (64 postings); VBMW [22] with variable-sized blocks (the average size is 64 postings); and DBMW [10, 11] for document-ID-oriented blocks where each block covers 1024 consecutive document IDs. We assume that the block metadata for VBMW is preloaded with space optimization described in [10, 11]. LazyBM [17] belongs to the same sub-family as DBMW with the same posting structure. We have not used LazyBM [17] as its implementation is not in public domain yet, and our current implementation of LazyBM is still slower than DBMW. We will study this more in the future.

A comparison with the baselines. Table 1 reports the mean query times in milliseconds for different algorithms with query lengths varying from 2 to 6. The window-driven execution of VBMW and DBMW is denoted as VBMW^p and DBMW^p respectively. This table also lists the 95th percentile *tail latency* for each query length within parentheses, corresponding to the response time occurring in the 95th percentile, as defined in [20]. Table 2 further shows the reduction ratio of the mean time for different query length and k values. A negative ratio indicates an improvement while a positive ratio indicates a degradation, and the overhead of probing and window navigation outweighs benefits.

BMW^p significantly outperforms BMW for 2 terms with all tested k values and for 3 terms with $k \leq 100$. For the mean time and 2 terms, BMW^p is 3.5x and 1.2x faster than BMW when $k = 10$ and $k = 1000$, respectively. For the tail latency and 2 terms, it is 4.9x and 1.8x faster when $k = 10$ and $k = 1000$, respectively. For 4 or more terms, BMW^p loses its advantages when $k \geq 50$. For long queries, $WinMax$ value is less accurate as a proxy for possible document scores. As k becomes larger, window navigation with probing becomes less effective, especially for long queries. One reason is that the gap of average score between top-k documents and remaining documents becomes smaller when k is large, and thus earlier probing becomes less effective for window pruning.

Like BMW, applying the proposed technique makes VBMW and DBMW significantly faster for queries with 2 terms and $k \leq 1000$ and with 3 terms and $k \leq 100$. When $k = 10$, VBMW becomes 7.8x and 10x faster for the mean and tail latencies, respectively. The reduction ratio when applied to VBMW is in general higher than to BMW because the variable block size design in VBMW makes *BlockMax* value closer to the score average within a block and hence $WinMax$ value is more accurate as a proxy for possible document scores. DBMW^p also does very well for 2 and 3 terms, and because each equal sized block uses consecutive IDs, the cost of window generation is zero, which reduces optimization overhead.

Table 1 also lists WAND [2] and interval-pruning [3] integrated with BMW-based traversal (denoted as IBMW) in Rows 11 and 12 as a reference point. In general, they are slower than the other algorithms, and IBMW costs too much in interval generation.

Impact of adaptive probing. Row 5 of Table 1 lists the time of BMW^p without probing, denoted as BMW^w. It simply generates windows and runs the final BMW pass. Even though some windows are pruned, the overhead of generating windows and window pruning outweighs the benefits, and the overall time is even slower than BMW. Another alternative, denoted as BMW^{ws} in Row 6, is to sort all block-aligned windows first and visit them in a descending order of their $WinMax$ values. BMW^p outperforms BMW^{ws} significantly, which illustrates the benefits of adaptive probing. It is too expensive to sort and jump around to access the sorted windows.

Time cost breakdown. Table 3 shows the breakdown of query processing time for BMW^p, and the total number of windows handled. The percentage of window selection cost in the total cost is small, varying from 2.6% to 5.5%. If our fast $O(N)$ selection method is not used, sorting based on $WinMax$ would be $O(N \log N)$ cost, where N is the total number of windows, and this translates to about 7x to 10x more cost. Specifically, sorting all windows explicitly would cost 3.0ms, 11.8ms, 27.2ms, 44.1ms, and 65.1ms for query length from 2 to 6, respectively. That would add too much overhead.

The time cost distribution of VBMW^p is similar proportionally as that of BMW^p. For DBMW^p, there is no cost for window generation. As every posting block covers 1024 document IDs uniformly, there are total about fixed 33,000 windows for this ClueWeb dataset.

Table 1: Mean query latency and tail latency in ms

Q. len.	2	3	4	5	6
			$k = 10$		
BMW	14.5 (84)	50.2 (229)	119.5 (413)	220.0 (735)	348.4 (1023)
BMW^p	4.2 (17)	33.1 (119)	102.8 (331)	189.6 (761)	309.7 (1042)
BMW^w	16.1 (57)	88.5 (470)	171.0 (487)	302.3 (923)	473.7 (1453)
BMW^{ws}	15.7 (60)	86.9 (328)	174.2 (545)	274.2 (890)	485.1 (1528)
VBMW	11.1 (80)	38.2 (203)	81.1 (321)	156.7 (479)	282.6 (671)
VBMW^p	1.4 (8)	13.1 (70)	57.3 (240)	105.7 (445)	251.5 (625)
DBMW	12.3 (63)	44.7 (166)	93.6 (241)	158.0 (378)	253.5 (537)
DBMW^p	1.1 (7)	15.9 (89)	45.3 (183)	99.9 (373)	180.7 (526)
WAND	33.7 (186)	74.3 (324)	144.0 (507)	244.5 (806)	484.5 (1157)
IBMW	25.4 (104)	96.6 (520)	210.2 (693)	345.1 (1289)	557.1 (1615)
			$k = 100$		
BMW	19.1 (108)	64.1 (277)	146.8 (460)	271.4 (800)	476.3 (1248)
BMW^p	9.6 (36)	57.1 (199)	153.1 (459)	326.2 (1045)	485.5 (1395)
VBMW	14.7 (96)	56.1 (246)	112.5 (352)	196.2 (518)	315.7 (709)
VBMW^p	7.4 (30)	49.2 (178)	121.9 (348)	236.2 (607)	368.7 (831)
DBMW	16.1 (83)	54.5 (182)	118.8 (289)	198.5 (435)	302.2 (632)
DBMW^p	7.2 (37)	36.7 (163)	90.0 (284)	175.1 (510)	289.6 (726)
			$k = 1000$		
BMW	33.3 (140)	101.5 (300)	224.4 (460)	409.9 (922)	690.0 (1426)
BMW^p	27.7 (78)	127.0 (367)	282.2 (656)	495.8 (1304)	722.0 (1710)
VBMW	32.4 (142)	101.9 (291)	218.9 (468)	350.1 (669)	534.1 (961)
VBMW^p	23.6 (64)	104.8 (283)	255.1 (518)	432.3 (834)	665.2 (1182)
DBMW	33.6 (166)	99.3 (252)	214.4 (439)	368.8 (653)	503.2 (802)
DBMW^p	23.0 (113)	76.4 (220)	178.7 (390)	327.0 (662)	470.7 (850)

Table 2: Mean time reduction ratio (-) or increase ratio (+)

	Q. len.	k=10	k=30	k=50	k=100	k=1000
$\frac{T(\text{BMW}^p)}{T(\text{BMW})}-1$	2	-71.3%	-58.2%	-52.6%	-49.6%	-16.6%
	3	-34.1%	-23.1%	-16.6%	-11.0%	+25.2%
	4+	-13.0%	-9.1%	+2.2%	+9.0%	+16.9%
$\frac{T(\text{VBMW}^p)}{T(\text{VBMW})}-1$	2	-87.2%	-69.6%	-63.8%	-49.9%	-27.3%
	3	-65.8%	-43.7%	-27.0%	-12.3%	+2.9%
	4+	-23.8%	-6.5%	+0.6%	+15.3%	+21.4%
$\frac{T(\text{DBMW}^p)}{T(\text{DBMW})}-1$	2	-90.9%	-76.7%	-73.8%	-55.4%	-31.6%
	3	-64.3%	-59.1%	-57.5%	-32.7%	-23.1%
	4+	-39.1%	-37.9%	-32.4%	-13.6%	-11.9%

Table 3: Mean time breakdown in ms for BMW^p

Query length	2	3	4	5	6
Num. of windows	41.5K	145.8K	323.7K	509.2K	737.7K
Window generation	1.4	7.6	19.2	32.1	38.1
Window selection	0.2	1.1	3.0	5.2	8.1
Adaptive probing	1.8	16.1	53.0	105.2	186.0
Final pass	0.7	8.3	27.6	47.2	77.5

5 CONCLUDING REMARKS

This paper proposes and evaluates a performance boosting scheme for BMW or its variants by changing their top-k search flow with window navigation and adaptive probing. The evaluation shows this technique can offer a significant speed advantage for short queries with a certain range of k values. For the mean response time, the reduction ratio varies from 16.6% to 90.9% with 2 terms and $k \leq 1000$, and from 11.0% to 65.8% with 3 terms and $k \leq 100$. The reduction trend is similar for the 95th percentile tail latency.

Considering that the average number of words in queries of popular search engines is between 2 and 3 [14, 30], the proposed technique can be very effective for a search engine which deploys many disjoint index partitions and when k does not need to be large for each individual partition that contributes part of top results.

ACKNOWLEDGMENTS

We thank the anonymous referees for their valuable comments. This work is supported in part by NSF IIS-2040146 and a Google faculty research award. This work used the Extreme Science and Engineering Discovery Environment [32] supported by NSF ACI-1548562. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Edward Bortnikov, David Carmel, and Guy Golan-Gueta. 2017. Top-k Query Processing with Conditional Skips. In *Proc. of the 26th International Conference on World Wide Web Companion.* 653–661.

[2] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient Query Evaluation Using a Two-level Retrieval Process. In *Proc. of the 12th ACM International Conference on Information and Knowledge Management.* 426–434.

[3] Kaushik Chakrabarti, Surajit Chaudhuri, and Venkatesh Ganti. 2011. Interval-Based Pruning for Top-k Processing over Compressed Lists. In *Proc. of the 2011 IEEE 27th International Conference on Data Engineering.* 709–720.

[4] Gordon V. Cormack, Mark D. Smucker, and Charles L.A. Clarke. 2011. Efficient and Effective Spam Filtering and Re-ranking for Large Web Datasets. *Information Retrieval* 14, 5 (2011), 441–465.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition.* MIT Press, Cambridge, MA, USA.

[6] 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 *Proc. of the 10th ACM International Conference on Web Search and Data Mining.* 201–210.

[7] Caio Moura Daoud, Edleno Silva Moura, David Fernandes, Altigran Soares Silva, Cristian Rossi, and Andre Carvalho. 2017. Waves: a Fast Multi-tier Top-k Query Processing Algorithm. *Information Retrieval* 20, 3 (2017), 292–316.

[8] 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.

[9] Lídia Lizziane Serejo de Carvalho, Edleno Silva de Moura, Caio Moura Daoud, and Altigran Soares da Silva. 2015. Heuristics to Improve the BMW Method and Its Variants. *Journal of Information and Data Management* 6, 3 (2015), 178–191.

[10] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. A Candidate Filtering Mechanism for Fast Top-k Query Processing on Modern CPUs. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 723–732.

[11] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. 2013. Optimizing Top-k Document Retrieval Strategies for Block-Max Indexes. In *Proc. of the 6th ACM International Conference on Web Search and Data Mining.* 113–122.

[12] Shuai Ding and Torsten Suel. 2011. Faster Top-k Document Retrieval Using Block-Max Indexes. In *Proc. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 993–1002.

[13] 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. *Proc. VLDB Endow.* 4, 12 (2011), 1213–1224.

[14] Bernard J. Jansen and Amanda Spink. 2006. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. *Information Processing & Management* 42, 1 (2006), 248–263.

[15] Karen Spärck Jones, Steve Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments. *Information Processing & Management* 36, 6 (2000), 779–840.

[16] Andrew Kane and Frank Wm. Tompa. 2018. Split-Lists and Initial Thresholds for WAND-Based Search. In *Proc. of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval.* 877–880.

[17] Omar Khattab, Mohammad Hammoud, and Tamer Elsayed. 2020. Finding the Best of Both Worlds: Faster and More Robust Top-k Document Retrieval. In *Proc. of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 1031–1040.

[18] Robert Krovetz. 2000. Viewing morphology as an inference process. *Artificial Intelligence* 118, 1-2 (2000), 277–294.

[19] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Softw. Pract. Exp.* 45, 1 (2015), 1–29.

[20] Joel Mackenzie, J. Shane Culpepper, Roi Blanco, Matt Crane, Charles L. A. Clarke, and Jimmy Lin. 2018. Query Driven Algorithm Selection in Early Stage Retrieval. In *Proc. of the 11th ACM International Conference on Web Search and Data Mining.* 396–404.

[21] Joel Mackenzie and Alistair Moffat. 2020. Examining the Additivity of Top-k Query Processing Innovations. In *Proc. of the 29th ACM International Conference on Information and Knowledge Management.* 1085–1094.

[22] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with Variable-sized Blocks. In *Proc. of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 625–634.

[23] Antonio Mallia, Michal Siedlaczek, and Torsten Suel. 2019. An Experimental Study of Index Compression and DAAT Query Processing Methods. In *Proc. of 41st European Conference on IR Research, ECIR' 2019.* 353–368.

[24] Antonio Mallia, Michał Siedlaczek, and Torsten Suel. 2021. Fast Disjunctive Candidate Generation Using Live Block Filtering. In *Proc. of the 14th ACM International Conference on Web Search and Data Mining.* 671–679.

[25] Antonio Mallia, Michal Siedlaczek, Mengyang Sun, and Torsten Suel. 2020. A Comparison of Top-k Threshold Estimation Techniques for Disjunctive Query Processing. In *Proc. of the 29th ACM International Conference on Information and Knowledge Management.* 2141–2144.

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

[27] The Lemur Project. 2020. https://www.lemurproject.org/indri.php. (2020).

[28] Cristian Rossi, Edleno S. de Moura, Andre L. Carvalho, and Altigran S. da Silva. 2013. Fast Document-at-a-time Query Processing using Two-tier Indexes. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 183–192.

[29] 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 *Proc. of the 15th ACM International Conference on Web Search and Data Mining.* 423–432.

[30] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. 1999. Analysis of a very large web search engine query log. In *ACM SIGIR Forum*, Vol. 33. ACM, 6–12.

[31] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2018. Efficient Query Processing for Scalable Web Search. *Foundations and Trends in Information Retrieval* 12, 4-5 (2018), 319–500.

[32] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 05 (2014), 62–74.

[33] Howard Turtle and James Flood. 1995. Query Evaluation: Strategies and Optimizations. *Information Processing & Management* 31, 6 (1995), 831–850.

[34] Erman Yafay and Ismail Sengor Altingovde. 2019. Caching Scores for Faster Query Processing with Dynamic Pruning in Search Engines. In *Proc. of the 28th ACM International Conference on Information and Knowledge Management.* 2457–2460.