Information Retrieval Journal manuscript No.
(will be inserted by the editor)

Efficient Query Processing Techniques for Next-Page Retrieval

Joel Mackenzie · Matthias Petri ·
Alistair Moffat

Received: date / Accepted: date

Abstract In top-k ranked retrieval the goal is to efficiently compute an ordered list of the highest scoring k documents according to some stipulated similarity function such as the well-known BM25 approach. In most implementation techniques a min-heap of size k is used to track the top scoring candidates. In this work we consider the question of how best to retrieve the *second* page of search results, given that a first page has already been computed; that is, identification of the documents at ranks $k + 1$ to $2k$ for some query. Our goal is to understand what information is available as a by-product of the first-page scoring, and how it can be employed to accelerate the second-page computation, assuming that the second-page of results is required for only a fraction of the query load. We propose a range of simple, yet efficient, next-page retrieval techniques which are suitable for accelerating Document-at-a-Time mechanisms, and demonstrate their performance on three large text collections.

Keywords Top-k Retrieval · Min-Heaps · Experimentation

This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP200103136).

J. Mackenzie
The University of Melbourne, Melbourne, Australia
E-mail: joel.mackenzie@unimelb.edu.au

M. Petri
Amazon Alexa, Manhattan Beach, CA, USA
E-mail: mkp@amazon.com

A. Moffat
The University of Melbourne, Melbourne, Australia
E-mail: ammoffat@unimelb.edu.au

1 Introduction

Top-k similarity search is a well-known problem in information retrieval. Given a collection of documents, D, and a query of q terms $\mathcal{Q} = \{t_1, t_2, \ldots, t_q\}$, the goal is to find the k highest scoring documents in D according to a similarity function $\mathcal{S}(\mathcal{Q}, d)$ (Zobel and Moffat, 2006). Those k documents are then presented in a search result page (SERP), for the query issuer to peruse. In the majority of cases one page of results satisfies the user's information need, and no further involvement from the system is required. But for some fraction ρ of queries, $0 \leq \rho \leq 1$, a second SERP is requested by the user, so that they can continue their browsing; with that fraction depending in part upon the type of search being performed. For example, job search has different properties to web search, with users much more likely to examine documents deep in the ranking (Mansouri et al., 2018; Spina et al., 2017).

We consider techniques for efficiently computing that second page of results beyond the initial top-k SERP. One obvious approach to second page retrieval is to undertake a "top-$2k$" computation each time a second SERP is requested, and then discard the first k documents so as to present documents $k+1$ to $2k$. This might be palatable if $\rho \ll 1$, but becomes expensive when ρ is close to one, since two query executions are required. Another obvious approach is to always compute the top-$2k$ documents for every query, and cache the second page of results until (and if) required. But when ρ is very small, this approach might be equally wasteful.

In this work we demonstrate that there is useful information that can be retained from the first SERP's top-k computation to accelerate computation of a second SERP. In particular, we describe three alternatives that employ state details captured at the end of the initial top-k traversal, and allow a suite of trade-offs between latency and effectiveness that sit in the space between the two obvious baselines that were noted above.

Our presentation proceeds as follows. Section 2 motivates the notion of efficient next page retrieval and gives an overview of the query processing mechanism we assume as a starting point. Section 3 then describes a range of options for accelerating the computation of second result pages; and then Section 4 compares a range of second-SERP mechanisms on three large data sets, and shows that useful information can indeed be carried over from the first top-k evaluation to reduce the time needed to generate a subsequent second page of search results. We also consider the possibility of very rapidly generating a high-quality approximate second-page ranking, without requiring it to be exactly the same as the second page of a full top-$2k$ computation, and demonstrate further trade-off possibilities. Finally, Section 5 discusses some limitations of our proposal, and considers possible extensions of it.

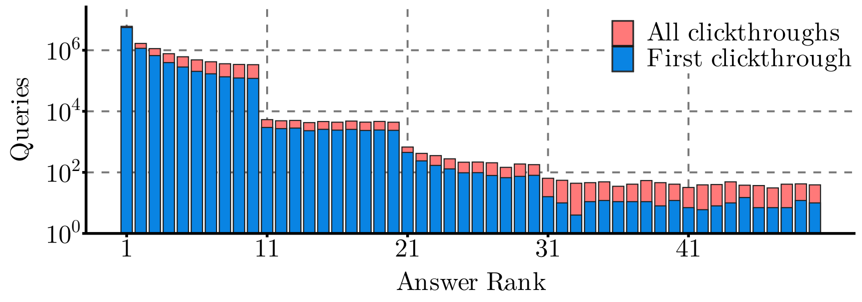

Fig. 1: Click positions, stratified by the *first* clickthrough for each query, and the total across *all* clickthroughs for each query, for the first 50 ranks in the SERPS associated with the 2006 MSN query log. (Redrawn from data presented as Figure 8 of Zhang and Moffat (2006).)

2 Background

First, we motivate the problem of efficient next-page retrieval by reviewing the literature on user behavior. Then, we provide an overview of the query processing mechanisms we assume as starting points for our analysis.

User Browsing Behavior. A range of studies have examined how users interact with ranked lists of results on search engine result pages (SERPs). Early work explored user interaction patterns through large-scale query logs derived from web search systems, demonstrating that users typically browse in a top-down fashion, with the majority of clicks occurring on the first page, and the majority of users only viewing a single SERP (Jansen and Spink, 2006; Jansen et al., 2000; Silverstein et al., 1999; Spink et al., 2001). These observations have been confirmed in more recent studies (Costa and Silva, 2010).

Figure 1, which is constructed from data generated by Zhang and Moffat (2006) and replicates Figure 8 of that paper, shows an example clickthrough distribution over approximately 12 million clicks into SERPS associated with the MSN search engine in May 2006. As expected, most of the user attention is focused on the early positions of the first page, with a large drop-off across page boundaries, and with only a very small percentage of users proceeding into the second results page.

But user behavior also depends on the search task and the user's goal. For example, the users of *job search* services are much more persistent than are web search users (Mansouri et al., 2018; Spina et al., 2017; Wicaksono and Moffat, 2018). Spina et al. (2017, Figure 1) provide a clickthrough graph similar to Figure 1 above, covering two types of search task – job search, carried out by individuals, and talent search, carried out by company recruiters, in both cases with results presented via SERPs of length twenty rather than the SERPs of length ten shown in Figure 1. For those two tasks the clickthrough rates at depth 21 are around 10% and 20% respectively of the clickthrough rate at rank one, suggesting that 10–20% of users access second pages of twenty results. User behavior also varies depending on the size of the SERP, and user models have been designed to capture the effects of the SERP size and screen size on user browsing behavior (Azzopardi and Zuccon, 2016; Kelly and Azzopardi, 2015). In our experiments we explore a range of values of ρ, the fraction of queries which have a second page request, to capture various situations and to understand the trade-offs which arise in different search contexts; some subset of that range might then be appropriate to each type of search task. For standard web search, for example, we might restrict our attention to $\rho \approx 1\%$, but we might take $\rho \approx 5\%$ when considering job and talent search (Spina et al., 2017).

Efficient Query Processing. A wide range of text similarity computations can be computed as a sum, over the terms t that appear in a query $\mathcal{Q}$, of term-document score contributions:

$$\mathcal{S}(\mathcal{Q}, d) = \sum_{t \in \mathcal{Q}} \mathcal{C}(t, d)\,.$$

For example, the well-known BM25 mechanism (Robertson and Zaragoza, 2009) can be fitted into this framework. To compute the top-k answer relative to a collection of documents, the postings lists for the query's terms are merged, with each of the documents that contain any of the terms having its score computed in turn. In this *document-at-a-time* (DAAT) processing mode, a set of top-k document-score pairs are maintained in a min-heap, with at any given time the kth largest value defining a *heap entry threshold*, denoted θ. As each document is scored, it either enters the heap, displacing the current kth smallest, and causing θ to increase; or it is immediately discarded as being outside the current top-k set, and hence guaranteed to also be outside the final top-k set. After all of the documents have been considered, θ reaches its final value, denoted here as Θ_k.

This underlying processing approach can be accelerated through the use of *dynamic pruning* techniques, which monitor θ and compare it to an easily-computed upper score bound estimate for each document, based on its postings. If the upper bound estimate indicates that the document could attain a similarity score $\mathcal{S}(\mathcal{Q}, d) > \theta$, then that document's score is fully computed, and properly compared against θ. But when it is clear from the upper bounds on the term-document contributions that $\mathcal{S}(\mathcal{Q}, d) \leq \theta$, then d's score does not need to be computed, and d can be *bypassed*. This observation led to the WAND mechanism, in which each term's postings list is augmented by a single maximum (over documents d in the collection D) $\mathcal{C}(t, d)$ value (Broder et al., 2003); then, with increased accuracy in the upper bound estimates, to the block-max WAND (BMW) approach (Ding and Suel, 2011), which stores an upper bound per term per block of postings; and finally to the variable block-max WAND (VBMW) implementation (Mallia et al., 2017), in which the lengths of the postings blocks are adjusted to capture local variations in $\mathcal{C}(t,d)$, thereby providing even tighter upper bounds. Mallia et al. (2019b) and Mackenzie and Moffat (2020) describe experiments that compare these three alternatives, and the baseline DaaT mechanism that they are all derived from.

At the end of DaaT top-k processing for a query, there will thus be four categories of documents that have been identified:

– Those that were scored, entered the heap, and remained in the heap until the end of the processing to form the top-k set (the *successful* documents);
– Those that were scored, entered the heap, but were later evicted from the heap as a result of a higher-scoring document being encountered (the *ejected* documents);
– Those that were scored, but when their score was calculated, did not make it into the heap (the *denied* documents); and
– Those that didn't get scored because of the dynamic pruning filter (the *bypassed* documents).

The top-k answer is exactly the first set, and none of them can appear in the second page of results. However the second SERP might contain documents from any of the other three categories; that is, the second results page could contain ejected, denied, or even bypassed documents. Figure 2 illustrates the way in which the increasing heap threshold θ partitions documents into these four classes, with the top-3 (top) and the top-6 (bottom) documents being identified. In the latter case fewer documents can be bypassed (dark gray points), more documents must be fully scored (green, blue, and pink points), more of them enter the heap (green and blue points), and more of them remain in the heap right through the query execution (green points).

Advance knowledge of a lower bound on the final kth largest similarity score can assist the dynamic pruning process (Fontoura et al., 2011). For example, de Carvalho et al. (2015) store the values of the kth highest scores in each postings list for certain values of k. Initializing θ to the largest (across the terms) of the kth largest (across documents) contributions $\mathcal{C}(t,d)$ is then safe, because for the given query there must be at least k similarity scores greater than or equal to that value. Kane and Tompa (2018), Yafay and Altingovde (2019), and Petri et al. (2019) have explored similar options, and Mallia et al. (2020) compare a range of such initializations.

A final point to note in this section is that mechanisms for top-k and second page retrieval can be categorized as being either safe or approximate. A *safe* technique guarantees delivery of rankings that are identical to those attained by an exhaustive full query evaluation over all documents and over all terms; on the other hand, an *approximate* (or *non-safe*) approach might construct rankings deviating from that ideal output, as a consequence of heuristics that reduce computation time. The fidelity of a non-safe approximation can be judged by computing a rank-weighted or unweighted overlap coefficient between the ideal (safe) and approximate rankings, with some suitable tie-breaking rule used to handle same-score documents consistently (Lin and Yang, 2019; Yang et al., 2016).

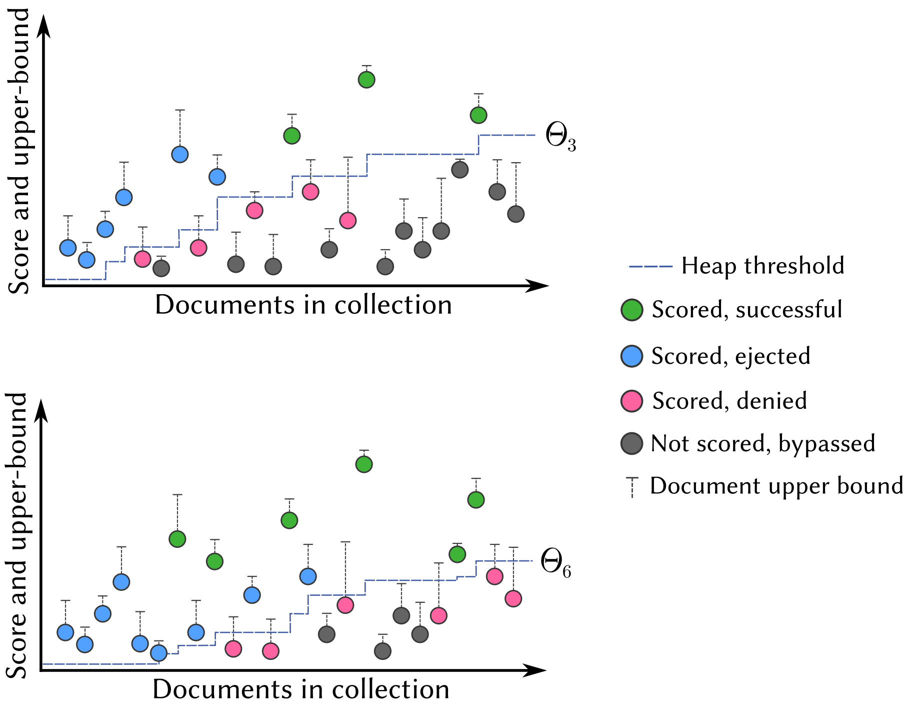

Fig. 2: An example of top-k retrieval with $k = 3$ (top) and $k = 6$ (bottom), with four classes of documents: successful (green); ejected (blue); denied (pink); and bypassed (dark gray). In this example, all but one of the documents in the top $k = 6$ results were scored during $k = 3$ traversal.

3 Efficient Next-Page Retrieval

We now describe a range of techniques for second-page retrieval, noting for each whether or not it is safe. The first two are the obvious baselines already mentioned in Section 1, to set a context.

On-Demand Computation (Baseline OD). The simplest baseline mechanism is to compute more results only when, and exactly when, the need arises. That is, if a second page is requested, the top-$2k$ documents are computed afresh, and the second half of that ranking is presented to the user. This approach correctly returns the documents occupying ranks $k+1$ to $2k$, and is thus safe. Each further "next page" request similarly results in a new query being issued. This mechanism can be expected to be efficient when the next-page rate ρ is very small, but not when ρ is high.

Full Precomputation (Baseline FP). A second baseline is to always compute the top-nk results for some $n > 1$, so that the next $n - 1$ pages are cached following the initial query. The value of n might be tuned based on a log analysis of past user behavior, providing a trade-off between the initial overhead of precomputing more detailed results, and the likely net savings. This approach is also safe; the difference is that it can be expected to be efficient if the next-page rate ρ is high, but a poor choice when ρ is low and the majority of users are satisfied after viewing just a single page of results.

The remainder of this section describes three new methods, employing increasing amounts of information calculated during the computation of the first page containing the top-k answer.

Retaining Ejected Documents (Method 1). The second results page must be composed of ejected, denied, or bypassed documents; of these, the ejected documents (the blue points in Figure 2) are the easiest set to exploit. They had their correct scores computed during the first-page computation, and were in the heap as "within the top-k so far" items at some point, suggesting that they could be good candidates for the second results page. Moreover, they are ejected from the heap in monotonically increasing score order, and hence all that is required to track the highest-scoring ones is an array of k items (or nk, if more than one possible followup page is being allowed for), cyclically overwriting the oldest element in the array as each new element is ejected from the heap (Figure 3). This additional record-keeping adds a minuscule overhead to the first-page computation.

There is no basis to expect that a second results page built solely from the k highest scoring ejected documents will be the same as would be constructed by a top-$2k$ retrieval (indeed, there might not even be k ejected documents arise for any given query), and this approach is thus approximate.

Adding Denied Documents (Method 2). The denied documents have also had their scores fully computed, and are another source of attractive candidates for the second results page – for example, denied documents from near the end of the computation can be expected to have higher scores than documents ejected early. The denied document scores are not generated in score order, so to capture them a secondary top-k min-heap is used rather than the previous cyclic array, with that second heap holding both the ejected and the denied documents. That is, the primary heap, with entry value θ, builds the first page of results; at the same time, the secondary heap, with entry threshold $\theta' < \theta$ concurrently builds a second page of results.

The need for the secondary heap adds a small overhead to the first page computation, but ensures that a better-quality pool of documents is retained, should they be required. However, the second SERP is still approximate rather than safe, because documents bypassed during the first-page computation could have had scores high enough to be part of the second results page (see Figure 2).

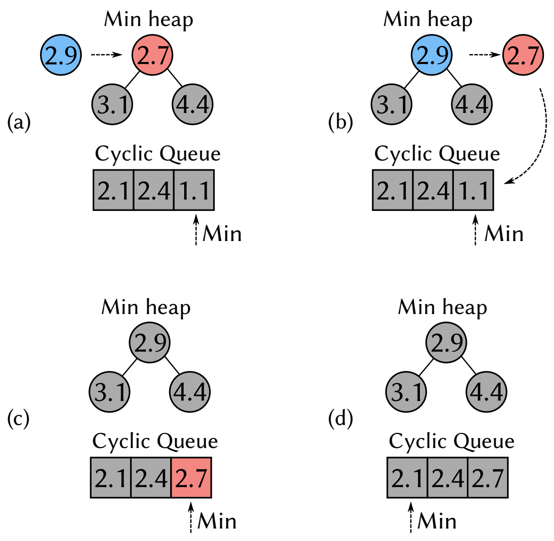

Fig. 3: Using a cyclic queue to track the k most recent heap ejections. The pointer into the cyclic buffer records the current minimum value (part (a)). When a document is ejected from the heap (part (b)), that minimum is overwritten with information about the ejected document and its score (part (c)), and the pointer is incremented so that it addresses the new minimum value (part (d)).

Adding Bypassed Documents (Method 3). Both previous methods generate a second results page very rapidly, based solely on information retained during the first-page computation. By the end of that first-page computation the primary heap threshold θ has risen to become Θ_k; and at the same time the secondary heap threshold θ' has reached a smaller terminal value, $\Theta'_{2k} \le \Theta_k$. If it could be demonstrated that $\Theta'_{2k} = \Theta_{2k}$, then the approximate second page could be certified as being safe. Unfortunately, all that can be said in general is that $\Theta'_{2k} \le \Theta_{2k} \le \Theta_k$, with Θ_{2k} unknown, and with the first inequality holding because it is certain that there are $2k$ or more documents with scores of Θ'_{2k} or more. Hence, if a safe second page is to be generated, any bypassed documents that could score between Θ'_{2k} and Θ_k must be identified and considered. To achieve that goal, the query must be re-processed against the collection.

Two key observations allow the work required by that re-execution to be substantially reduced compared to the OD baseline. The first is that the value of Θ'_{2k} generated during the first-page execution provides a legitimate bound with which to prime the heap threshold for the second-page re-computation. That is, second-page query execution can be commenced with a reasonably accurate initial under-estimate $\theta \leftarrow \Theta'_{2k}$ of the true second-page threshold Θ_{2k}, thereby extracting the greatest benefit from whichever dynamic pruning mechanism is being used.

The second observation is that it is not necessary to consider the entire document range during the second-page execution. To see why, consider some document with ordinal identifier d_j that is a member of the first-page ejected set. When d_j joined the primary heap during that first-page execution, the heap threshold at that time, θ_j, was less than the score computed for d_j, that is, $\theta_j < \mathcal{S}(\mathcal{Q}, d_j)$, otherwise d_j would have been a denied document. Because of that relationship, every document d_i that precedes d_j in the collection (that is, when $i < j$) either had a score less than d_j, with $\mathcal{S}(\mathcal{Q}, d_i) < \mathcal{S}(\mathcal{Q}, d_j)$; or, if document d_i had a score greater than document d_j, then d_i was in the primary heap when d_j was added, and, because its score was higher, remains in the heap immediately after d_j's ejection. That is, only documents that come *after* document d_j in the collection ordering can have both a score greater than d_j and also be in the bypassed set.

Now consider the moment in the first-page execution at which d_j is ejected from the primary top-k heap. Suppose that the document that causes the eviction is d_m, where $j < m$. Immediately prior to the eviction we must have had $\theta_m = \mathcal{S}(\mathcal{Q}, d_j)$, and thus (because θ_ℓ is non-decreasing in ℓ) that $\theta_\ell \leq \mathcal{S}(\mathcal{Q}, d_j)$ for all $j < \ell \leq m$. In combination with the point made in the previous paragraph, this second relationship means that any documents that both have scores greater than $\mathcal{S}(\mathcal{Q}, d_j)$ and that are also in the first-page bypassed set can only appear after d_m. That is, d_m, the cursor position in the document-at-a-time processing scan at the moment d_j is ejected from the primary top-k heap, is the earliest that a bypassed document could have a score greater than $\mathcal{S}(\mathcal{Q}, d_j)$.

Finally, consider the set of at most k documents identified via the secondary heap during the first-page execution, which is composed of a mixture of ejected and denied documents with scores greater than or equal to Θ'_{2k}. These documents can be added to the first-page top-k to initialize the second-page execution heap of size $2k$, and the heap entry threshold Θ'_{2k}. If the lowest scoring document of that joint set, d_j, is a denied document, then the most recently ejected document with a similarity score $< \Theta'_{2k}$ is identified, and the original top-k execution can be safely "resumed" at the point in which that document was ejected, denoted d_m, as all higher scoring documents that are potentially missing in the top-$2k$ must occur *after* that point. Similarly, if d_j is itself an ejected document, then the original top-k execution can be resumed at the point in which d_j was ejected, d_m. Thus, it is sufficient to track only the most recent k ejected documents from the first-page heap to locate this resumption point. To do so, we re-introduce the cyclic array of k elements during the first-page execution, now storing tuples $\langle \theta_j, d_m \rangle$ corresponding to the most recent k ejected documents. Once a suitable restart point is identified, the second-page execution can proceed as a "resumption" of the first-page execution, with an initialized heap of size $2k$ and an initial threshold of Θ'_{2k}.

Figure 4 illustrates the configurations being discussed. The first-page computation is shown in the top part, with Θ_k and Θ'_{2k} being computed as by-products of the standard top-k computation. As well, document d_j is identified during that first-page execution as the least-score ejected element in the sec-

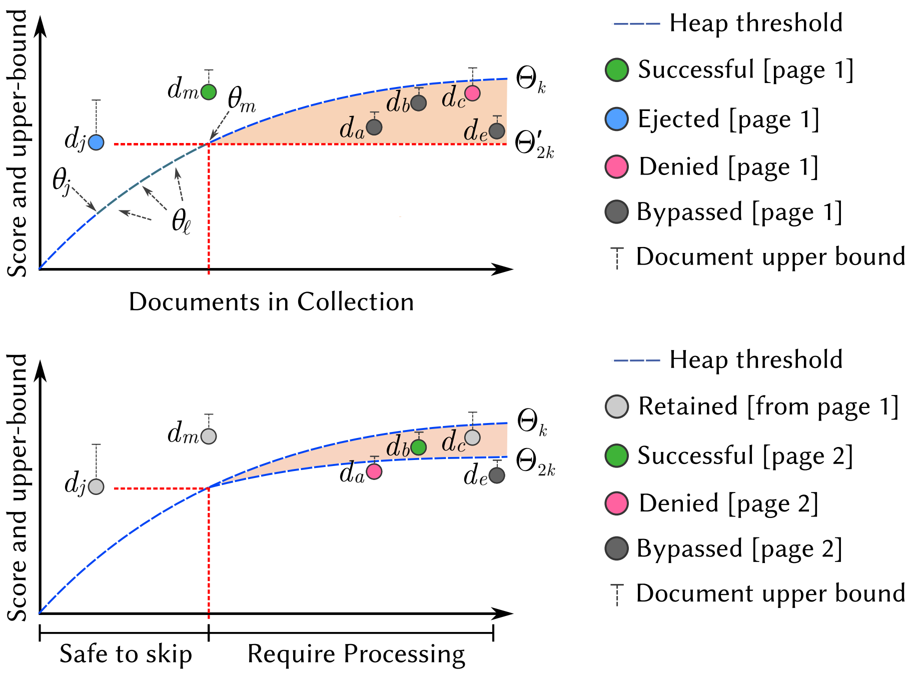

Fig. 4: First-page retrieval (top) and accelerated second-page retrieval (bottom). Documents d_a and d_b were bypassed during the first-page retrieval, and need to be scored during the second-page computation. Document d_c would have already be considered during the first page computation and retained as a denied document; it does not get re-scored. Document d_e is bypassed during both first-page and second-page retrieval.

ondary heap, and it's partner d_m, the one that resulted in its ejection, is also noted. Hence, document d_m marks the earliest point in the document sequence at which any documents with scores greater than $\mathcal{S}(\mathcal{Q}, d_j)$ might have been bypassed.

The corresponding Method 3 second-page query execution is shown in the bottom part of Figure 4. All documents prior to d_m in the collection ordering can be skipped without further consideration, because all previous documents that were bypassed are known to also be bypassable in the top-$2k$ retrieval process that builds a safe second results page. And any required documents that precede d_m in the collection ordering that were not bypassed during the first-page execution must have remained in the secondary heap that emerged from it, and that is used to seed the second-page computation. Only documents that were bypassed after d_m was processed need to be re-examined and tested against that secondary heap during the second-page computation.

As a final addition to this jigsaw, a bitvector over the document space can also be added, to keep track of the documents that have already been scored in the first-page execution. There is no need for them to be scored again during

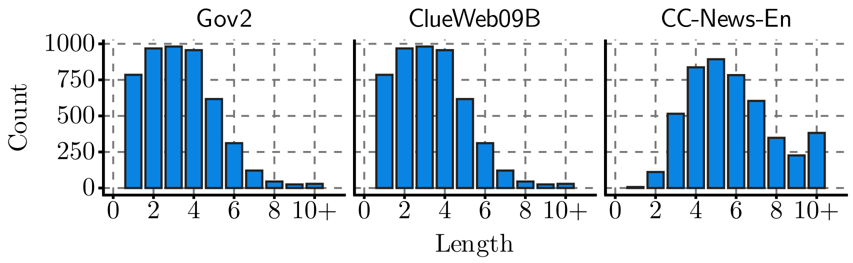

Fig. 5: Distribution of query length, in terms, across the three test collections. Note that Gov2 and ClueWeb09B utilize the same query log.

accelerated second-page execution, and only documents that were bypassed during the first-page process need to be scored, further reducing processing costs.

4 Experiments

We now carry out a detailed experimental evaluation of the five second-page retrieval mechanisms described in Section 3: two safe baselines (OD and FP); two approximate second-page procedures (Method 1 and Method 2); and one safe second-page mechanism (Method 3).

Data and Queries. Three collections are employed; Gov2 (approximately 25 million `.gov` documents); ClueWeb09B (around 50 million web documents); and CC-News-En (around 43.5 million English news documents (Mackenzie et al., 2020)). A query log was generated for the Gov2 and ClueWeb09B datasets by selectively sampling 5,000 random queries from the TREC *Million Query Track* (Allan et al., 2007, 2008; Carterette et al., 2009), such that there were 1000 queries of each length from one to four terms, and an additional 1000 containing five or more terms. For CC-News-En, a random sample of 5,000 queries was taken from its temporally matched query log. After that initial sampling stage, any queries with out-of-vocabulary terms were removed. Figure 5 shows the distribution of the lengths of the test queries after sampling and then filtering. The final query subsets are provided as part of the experimental codebase (see Section 5).

Experimental Setup. The collections were indexed using Anserini with the default stoplist and Porter stemming (Yang et al., 2018). The indexes were next converted to the common index file format (CIFF) (Lin et al., 2020); then reordered using the recursive graph bisection approach (Dhulipala et al., 2016; Mackenzie et al., 2019); before being ingested by the PISA search system (Mallia et al., 2019a), which was used for the remainder of the experimentation. For all experiments, documents are ranked according to the BM25 (Kamphuis

	Gov2			ClueWeb09B			CC-News-En		
	WAND	BMW	VBMW	WAND	BMW	VBMW	WAND	BMW	VBMW
Ejected	6.46	6.47	6.47	6.43	6.43	6.43	6.58	6.59	6.59
Denied	3.50	2.75	2.89	3.54	2.56	2.76	3.41	3.10	3.08
Bypassed	0.04	0.78	0.64	0.03	1.00	0.81	0.01	0.31	0.33

Table 1: Average number of documents in ranks 11 to 20 of exact top-20 rankings across the three categories "ejected", "denied", and "bypassed" defined with respect to the corresponding exact top-10 ranking for each query when computed using three different dynamic pruning regimes.

et al., 2020; Robertson and Zaragoza, 2009; Trotman et al., 2014) model with parameters $k_1 = 0.4$ and $b = 0.9$ (Trotman et al., 2012). The exact BM25 formulation employed can be found in the PISA overview from Mallia et al. (2019a). All experiments were performed entirely in-memory on a Linux machine with two 3.50 GHz Intel Xeon Gold 6144 CPUs and 512 GiB of RAM. All reported query timings are the mean value of three independent runs, and all our results pertain to top-10 evaluation, the most likely situation in which "next page" requests must be processed.

Proof of Concept. The primary interest is in regard to query execution time, which will be reported shortly. To set the scene for those results, Table 1 shows the breakdown of document types across the three collections and three dynamic pruning techniques, categorizing the documents at ranks 11 to 20 in safe top-20 rankings according to what happened in each corresponding top-10 execution, using three different dynamic pruning methods.

As can be seen, around two-thirds of the documents needed for a safe top-20 ranking are from the ejected sets; and another quarter are from the denied sets. Only a minority of the top-20 documents were bypassed during the top-10 first-page retrieval. The more precise upper score bound estimates generated by the BMW and VBMW approaches mean that a greater fraction of the documents needed for second-page safety are bypassed by top-10 retrieval, an observation which has also been made in connection with threshold prediction for accelerated top-k retrieval (Petri et al., 2019). Indeed, their ability to bypass more documents is exactly the attribute that makes those two methods faster for first-page retrieval. But even when that is taken into account, Method 2 correctly captures, on average, more than nine of the ten documents required for the second SERP, indicating that it can be expected to provide efficient-yet-effective next-page retrieval regardless of which pruning technique is used.

Figure 6 shows the scaled relative positions across the document space associated with each query at which the documents from each of these three categories appear during processing, using the same three test environments. Ejected documents are more likely to appear early in the traversal, when the heap threshold is low, whereas denied documents tend to occur later in the

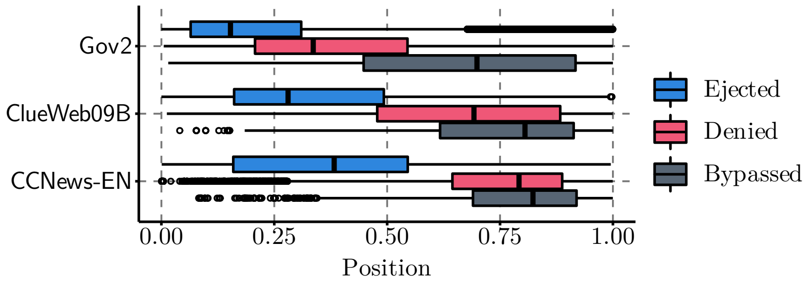

Fig. 6: Relative positions in the query document space of documents in ranks 11 to 20 of the exact top-20 rankings across three categories defined by the corresponding top-10 rankings when using VBMW dynamic pruning, expressed as fractions of the union of the documents in the postings lists of the terms associated with each query. Since the heap threshold increases as more documents are evaluated, most ejections occur early on, with more documents being denied and bypassed later in the traversal.

traversal, once the heap threshold is larger. Finally, bypassed documents occur even later, when the heap threshold is close to its terminal value. By safely skipping to the first point at which a valid second-page document may have been bypassed, Method 3 is intended to avoid scoring many potential candidates, accelerating the second-page retrieval process.

Quantifying Safeness. Both baselines, and Method 3, are safe. To measure the relative safeness of Methods 1 and 2 in a meaningful way, and to avoid issues associated by score ties (Lin and Yang, 2019; Yang et al., 2016), the following process was used. For each query, the documents in positions 11 to 20 of the safe ranking were divided into two groups: those with scores greater than that of the 20th document; and those with scores equal to it. Each document in the first group was counted as a "+1" if it appeared anywhere within ranks 11 to 20 of the approximate ranking, and zero otherwise. In addition, a further "+1" was recorded for each document in the second group if there was a corresponding document with the same score in the approximate ranking, with the restriction that each document in the approximate ranking could only be counted once.

The count for the query was then divided by ten, to get an equality-insensitive overlap ratio, with 1.0 corresponding to "safe except for perturbations in connection with score ties", and 0.0 representing complete second page mismatch. Those per query overlaps were then averaged over the query sets to get system overlaps.

Selection of Pruning Mechanism. In a range of preliminary experiments we measured all of WAND, BMW, and VBMW-based retrieval for top-10 and top-20 retrieval across the three collections. In results that agree with similar

Mechanism	Gov2			ClueWeb09B			CC-News-En		
	Page 1	Page 2	Ovlp.	Page 1	Page 2	Ovlp.	Page 1	Page 2	Ovlp.
Baseline OD	3.6	4.2	–	11.8	13.6	–	24.5	28.8	–
Baseline FP	4.2	0.0	–	13.6	0.0	–	28.8	0.0	–
Method 1	3.6	0.0	0.66	11.8	0.0	0.66	24.6	0.0	0.67
Method 2	3.6	0.0	0.94	11.9	0.0	0.93	24.6	0.0	0.97
Method 3	3.8	1.6	1.00	12.3	4.9	1.00	25.1	9.9	1.00

Table 2: First-page retrieval time and second-page retrieval time for five methods, and equality-insensitive overlap for the two non-safe methods. Times are average milliseconds per query, with VBMW used throughout.

experiments carried out by others, we found that VBMW was consistently fastest, and it is used throughout the results reported here.

First and Second Page Retrieval. Table 2 reports the mean latency for each technique, split across the cost of first and second page retrieval. Of the two baseline techniques, the OD (on demand) approach is the fastest to generate a first page of results, and the baseline FP (full pre-computation) the slowest, providing bookends to the palette of three further approaches described here. Method 1 and Method 2, both approximate rather than safe for the second page, have negligible additional cost compared to the OD baseline, and are equally fast to generate a first (safe) page of results. Both then generate second pages without further computation being required, with Method 2 giving a "closer to safe" outcome than Method 1, which only retains ejected documents. These results confirm the outcomes presented in Table 1.

Method 3 generates a safe second page. To achieve that, it takes slightly longer than baseline OD when processing the first page, but still much less than baseline FP, and recoups that cost – plus more – via accelerated second-page processing. The same patterns of relative performance occur in all three test collections.

Jump-Starting the Second Page. Figure 7 helps explain the speed of the Method 3 second-page computation. To generate the graphs, the set of document numbers involved in each query was generated from the union of the terms' posting lists, and document numbers mapped in a uniform manner into the range zero to one. The "resumption" document d_m for each query was similarly mapped to the same range, and expressed as a decimal value. For example, a value of 0.5 indicates that half of the document range was completely avoided during the Method 3 second page computation – that d_m (see Figure 4) was halfway through the union of the query's postings lists.

As can be seen in Figure 7, for the small Gov2 collection, only about a quarter of the document space is skipped, but for the two larger collections it is respectively around one third, and around one half. Figure 7 also shows how the more accurate document upper score bound estimates achieved by

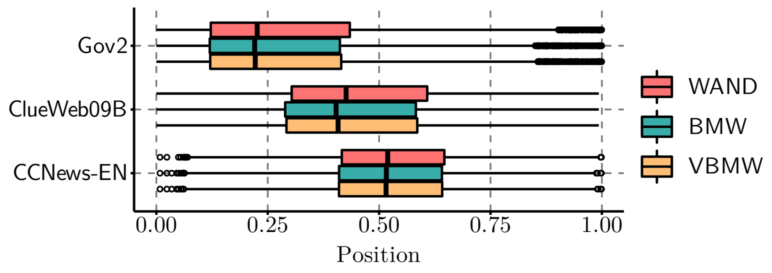

Fig. 7: Distribution of jump-start second page query resumption positions, expressed as scaled fractions of the set of documents involved in the union of the terms associated with each query. Higher resumption positions indicate a larger relative volume of documents being safely skipped.

Collection	Mechanism	Query second page request rate, ρ					
		1%	2%	4%	10%	20%	40%
Gov2	Baseline OD	3.7	3.7	3.8	4.0	4.5	5.3
	Method 3	3.8	3.8	3.8	3.9	4.1	4.4
	Baseline FP	4.2	4.2	4.2	4.2	4.2	4.2
ClueWeb09B	Baseline OD	11.9	12.1	12.3	13.2	14.5	17.3
	Method 3	12.4	12.4	12.5	12.8	13.3	14.3
	Baseline FP	13.6	13.6	13.6	13.6	13.6	13.6
CC-News-En	Baseline OD	24.8	25.1	25.7	27.4	30.3	36.0
	Method 3	25.2	25.3	25.5	26.1	27.1	29.1
	Baseline FP	28.8	28.8	28.8	28.8	28.8	28.8

Table 3: Combined querying cost, first two pages, for the three safe methods, and a range of values of ρ. Times are calculated from the same data as used to construct Table 2, and are expressed as average milliseconds per query, with VBMW used throughout. Similar patterns were also observed for the other two pruning approaches.

BMW and VBMW give rise to the need for a slightly greater fraction of the document range to be checked during second page computation.

Overall Querying Cost. The total time spent processing each query depends on both the cost of first and second page retrieval, and also on ρ, the fraction of queries for which a second page is requested. Table 3 uses the same data as already presented in Table 2, but combines the first page cost with a varying fraction of the second page cost, to get an estimated total per query cost. Only the three second-page-safe methods are shown, as a basis for comparison; with the best method shown in blue for each different value of ρ.

For low values of ρ, the cheapest overall approach remains to compute (only) the top-10 at first, and not invest in any preparation for a possible second page request. If, and only if, that request arrives, it is costly to process, but when ρ is low the gamble pays off. Similarly, for high values of ρ, the best overall approach is to pre-compute two pages, and have the second k documents ready to dispense when the request arrives. In between, and slightly better than either baseline for all three collections when ρ is in the range $\approx$ 4–20%, is Method 3, which adds a small investment to each first page computation in order to greatly accelerate second page computations.

Keeping It Simple. One potential drawback of Method 3 is the volume of state data retained between every first page computation and a possible second page request: the secondary heap; the cyclic array; and the bitvector showing which documents have already been scored. The first two structures are very small, at just ten elements each. But the bitvector might be large. For 50 million documents it requires that around 6 MiB per query be allocated and held for some interval of time (perhaps minutes), a significant space imposition that might well be better employed for other purposes.

If that space expenditure is regarded as being excessive but it is important that the second-page ranking be safe (that is, if Method 2 cannot be used), then an additional option is to compute the first page using Method 2, but then only retain the kth largest score Θ'_{2k} from the secondary heap, a single floating-point value per query. Then, if/when a second page is required for that query, the floating point value can be used as an initial heap priming threshold $\theta \leftarrow \Theta'_{2k}$ for a full top-$2k$ computation. This simpler approach also outperforms both of the baseline approaches, but over a more restricted range of ρ. For example, knowledge of the Method 2 Θ'_{2k} value allows (see Table 2) the VBMW/CC-News-En second page computation time to be reduced from 28.8 milliseconds to 20.9 milliseconds, and hence if $\rho = 10\%$, results in an average per-query cost of 26.7 milliseconds, slightly better than both the OD and FP baselines shown in Table 3.

Beyond Page Two. A natural question to ask is whether these techniques extend to third (and further) pages. Clearly both baselines can continue to be applied, albeit with mounting costs. It might thus be wiser to blend them, and if/when a third page is requested, compute the documents at ranks $2k + 1$ to $4k$ inclusive, so that the third and fourth pages are both ready; and if then a fifth page is requested, to compute ranks $4k + 1$ to $8k$ in another pass through the index; and so on. Regardless of how many documents are required, the techniques described here – use of a cyclic array, a secondary heap, a bitvector of evaluated documents, jump-start resumption, and heap threshold priming – can be used in conjunction with any such processing, provided only that the maximum depth k' required to serve the next page (or set of pages) is known at the time the current query is being processed.

In practice, a decreasingly small fraction of searches involve the user requesting a third page or beyond (see, for example, Figure 1), and diminishing marginal gains can be expected from any acceleration mechanism. In particular, if third page requests occur with probability of the order of ρ^2, and fourth page requests with probability approximately ρ^3, and so on, then avoiding the retention of ever-increasing amounts of state information might be more important than saving further slivers of computation time.

5 Discussion and Conclusion

Limitations and Future Work. The mechanisms we have proposed are not without drawbacks. As already discussed, Method 1 and Method 2 produce a second result page with negligible further work required and with only small amounts of extra space being required, but those second pages are not guaranteed to be safe; and that fact alone might be enough to rule these two options out of contention for some applications.

Method 3 is safe, and for a range of values of ρ is faster than either of the two baselines, but involves a substantial commitment of memory to each query for the bitvector, and it is unclear whether that same memory could be deployed in a different manner to achieve equal or greater query throughput savings. For example, memory could equally well be used for answer caching, thereby completely avoiding computation of some repeat queries; or could be used to store certain frequently-accessed postings lists in partially decompressed form. If so, it might be that simply retaining the secondary heap threshold Θ'_{2k} from the first-page execution for each query, and using it to prime the second-page heap, is the best option.

Another factor that affects the usefulness of this work is the increasing tendency for retrieval systems to be implemented as a cascade of phases, with a fast-but-blunt first phase based on BM25 and dynamic pruning used to select (say) 1,000 documents, and then further phases of increasing complexity applied to refine the document ranking and determine the final top-k listing that gets presented to the user (Chen et al., 2017; Matveeva et al., 2006; Wang et al., 2011). In such arrangements the user is not directly connected to the underlying first-phase similarity computation, and a user-initiated request for a second page of results might or might not translate into a fresh round of activity in the first-phase retrieval engine. That is, our techniques are primarily applicable only to situations in which a BM25-like computation is being used in a single phase of retrieval to supply results pages directly to users.

We have focused on the usual DaaT retrieval mode. Other processing options are also possible, including Term-at-a-Time (TaaT) traversal and Score-at-a-Time (SaaT) traversal. The techniques we have described do not directly apply to these modes. However, the TaaT and SaaT processing strategies may provide their own opportunities for accelerating next-page retrieval. For example, both techniques generally store per-document scores in a large accumulator table, and retention of that table seems likely to provide a boost to safe second page retrieval. It would be interesting to compare these different arrangements head-to-head to improve the understanding of the intricate trade-off space between them (Crane et al., 2017). Similarly, there are likely to be index-independent optimizations which could be explored. One such example is to build a classifier which can predict whether a user is likely to continue to a second or third results page, and adjust the initial top-k computation accordingly, similar to the notion of *pre-fetching* links which are likely to be clicked (White et al., 2017). We leave this idea for future investigation.

Conclusion. We have explored a range of interesting algorithmic issues connected with the question of retrieving a second page of top-k results in search applications. Three methods have been introduced, two of them simple ways of approximating the second search page, and a third that undertakes a restricted re-scan to ensure that the second page of results is faithful to what would be generated by a full top-$2k$ mechanism. The key to bounding the effort required in the re-scan is to retain strategic state information during the generation of the first page, thereby providing the second page retrieval with a useful jump-start.

Experiments with three large text collections have shown that the new score-safe approach fits neatly between two alternative baseline mechanisms, and for a range of continuation probabilities ρ offers the solution that involves the least total computational cost when first page and second page retrieval are combined. The two non-score safe approaches can also be used if required, with Method 2 in particular providing retrieval effectiveness (in terms of which documents get presented to the user) that on average is in close agreement to the second-page-safe alternatives.

We have also discussed the limitations that apply to our mechanism, and acknowledge that these are restrictive. Nevertheless, we believe that there will be situations in which users directly interact with a single-phase search system. We also believe that innate algorithmic curiosity is served by this work, in terms of responding to the question we posed in Section 1 as to whether or not second-page retrieval can be accelerated by retaining information from the corresponding first-page computation.

Software. The software that was developed during this work, and additional data and tools, are available at `https://github.com/jmmackenzie/next-page`.

Acknowledgements. This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP200103136). We thank the referees for their helpful comments.

References

J. Allan, B. Carterette, J. A. Aslam, V. Pavlu, B. Dachev, and E. Kanoulas. Million query track 2007 overview. In *Proc. Text Retrieval Conf. (TREC)*, 2007.

J. Allan, J. A. Aslam, B. Carterette, V. Pavlu, and E. Kanoulas. Million query track 2008 overview. In *Proc. Text Retrieval Conf. (TREC)*, 2008.

L. Azzopardi and G. Zuccon. Two scrolls or one click: A cost model for browsing search results. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 696–702, 2016.

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

B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas. Million query track 2009 overview. In *Proc. Text Retrieval Conf. (TREC)*, 2009.

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

M. Costa and M. Silva. A search log analysis of a Portuguese web search engine. In *INForum – Simpósio de Informática*, pages 525–536, 2010.

M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. A comparison of Document-at-a-Time and Score-at-a-Time query evaluation. In *Proc. Conf. on Web Search and Data Mining (WSDM)*, pages 201–210, 2017.

L. L. S. de Carvalho, E. S. de Moura, C. M. Daoud, and A. S. da Silva. Heuristics to improve the BMW method and its variants. *Journal of Information and Data Management*, 6(3):178–191, 2015.

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

S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 993–1002, 2011.

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

B. J. Jansen and A. Spink. How are we searching the world wide web? A comparison of nine search engine transaction logs. *Information Processing & Management*, 42(1):248–263, 2006.

B. J. Jansen, A. Spink, and T. Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing & Management*, 36(2):207–227, 2000.

C. Kamphuis, A. P. de Vries, L. Boytsov, and J. Lin. Which BM25 do you mean? A large-scale reproducibility study of scoring variants. In *Proc. European Conf. on Information Retrieval (ECIR)*, pages 28–34, 2020.

A. Kane and F. W. Tompa. Split-lists and initial thresholds for WAND-based search. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 877–880, 2018.

D. Kelly and L. Azzopardi. How many results per page? A study of SERP size, search behavior and user experience. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 183–192, 2015.

J. Lin and P. Yang. The impact of score ties on repeatability in document ranking. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 1125–1128, 2019.

J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. de Vries. Supporting interoperability between open-source search engines with the common index file format. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 2149–2152, 2020.

J. Mackenzie and A. Moffat. Examining the additivity of top-k query processing innovations. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 1085–1094, 2020.

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

J. Mackenzie, R. Benham, M. Petri, J. R. Trippas, J. S. Culpepper, and A. Moffat. CC-News-En: A large English news corpus. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 3077–3084, 2020.

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

A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. PISA: Performant indexes and search for academia. In *Proc. OSIRRC at SIGIR 2019*, pages 50–56, 2019a.

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

A. Mallia, M. Siedlaczek, M. Sun, and T. Suel. A comparison of top-k threshold estimation techniques for disjunctive query processing. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 2141–2144, 2020.

B. Mansouri, M. S. Zahedi, R. Campos, and M. Farhoodi. Online job search: Study of users' search behavior using search engine query logs. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 1185–1188, 2018.

I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 437–444, 2006.

M. Petri, A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. Accelerated query processing via similarity score prediction. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 485–494, 2019.

S. E. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations & Trends in Information Retrieval*, 3:333–389, 2009.

C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

D. Spina, M. Maistro, Y. Ren, S. Sadeghi, W. Wong, T. Baldwin, L. Cavedon, A. Moffat, M. Sanderson, F. Scholer, and J. Zobel. Understanding user behavior in job and talent search: An initial investigation. In *Proc. SIGIR Workshop on eCommerce*, 2017. URL `http://ceur-ws.org/Vol-2311/paper_2.pdf`.

A. Spink, D. Wolfram, B. J. Jansen, and T. Saracevic. Searching the web: The public and their queries. *Journal of the American Society for Information Science*, 52(3):226–234, 2001.

A. Trotman, X.-F. Jia, and M. Crane. Towards an efficient and effective search engine. In *Proc. OSIR at SIGIR 2012*, pages 40–47, 2012.

A. Trotman, A. Puurula, and B. Burgess. Improvements to BM25 and language models examined. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 58–65, 2014.

L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*, pages 105–114, 2011.

R. W. White, F. Diaz, and Q. Guo. Search result prefetching on desktop and mobile. *ACM Trans. on Information Systems*, 35(3), 2017.

A. F. Wicaksono and A. Moffat. Empirical evidence for search effectiveness models. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 1571–1574, 2018.

E. Yafay and I. S. Altingovde. Caching scores for faster query processing with dynamic pruning in search engines. In *Proc. ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pages 2457–2460, 2019.

P. Yang, H. Fang, and J. Lin. Anserini: Reproducible ranking baselines using Lucene. *ACM Journal of Data and Information Quality*, 10(4):1–20, 2018.

Z. Yang, A. Moffat, and A. Turpin. How precise does document scoring need to be? In *Proc. Asia Information Retrieval Societies Conf. (AIRS)*, pages 279–291, 2016.

Y. Zhang and A. Moffat. Some observations on user search behavior. In *Proc. Australasian Document Computing Symp. (ADCS)*, pages 1–8, 2006.

J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6.1–6.56, 2006.