Exploring the Magic of WAND

Matthias Petri[1,2] J. Shane Culpepper[1] Alistair Moffat[2]

1. School of Computer Science and Information Technology
RMIT University, Australia

2. Department of Computing and Information Systems
The University of Melbourne, Australia

ABSTRACT

Web search services process thousands of queries per second, and filter their answers from collections containing very large amounts of data. Fast response to queries is a critical service expectation. The well-known WAND processing strategy is one way of reducing the amount of computation necessary when executing such a query. The value of WAND has now been validated in a wide range of studies, and has become one of the key baselines against which all new top-k processing algorithms are benchmarked. However, most previous implementations of WAND-based retrieval approaches have been in the context of the BM25 Okapi similarity scoring regime. Here we measure the performance of WAND in the context of the alternative Language Model similarity score computation, and find that the dramatic efficiency gains reported in previous studies are no longer achievable. That is, when the primary goal of a retrieval system is to maximize effectiveness, WAND is relatively unhelpful in terms of attaining the secondary objective of maximizing query throughput rates. However, the BM-WAND algorithm does in fact help reducing the percentage of postings to be scored, but with additional computational overhead. We explore a variety of trade-offs between scoring metric and processing regime and present new insight into how score-safe algorithms interact with rank scoring.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and software—*performance evaluation*; H.3.1 [**Content Analysis and Indexing**]: Indexing methods

Keywords

Text indexing, information retrieval, performance

1. INTRODUCTION

Practical information retrieval systems must be both effective and efficient. Systems must generate answers that are good matches to queries, using as little computational resources as possible. The effectiveness of a system is determined by the choice of similarity scoring function; while the efficiency of the system is dictated by how quickly that similarity computation can be carried out.

Precomputed inverted indexes allow the documents that contain any particular query term to be quickly identified (see, for example Zobel and Moffat [21]). If the objective is to compute a similarity score for every document that contains any of the query terms, then exhaustive processing of the inverted list associated with each query term is sufficient. Two alternatives have been explored: *term-at-a-time* (TAAT) processing, where inverted lists are opened sequentially, and a state vector of partially computed similarity scores is maintained; and *document-at-a-time* (DAAT) processing, in which the inverted lists are opened concurrently, and each document is fully scored before any other document is considered. Document-at-a-time processing requires that the inverted lists be stored in document-sorted order; term-at-a-time allows other index orderings.

If only a subset of the documents matching the query are to be returned – as in the case of a *top k query* where only the k documents with the highest similarity score are to be identified – then more sophisticated processing regimes can be employed, with the objective of minimizing the amount of numeric computation that is required. For example, suppose that a query q has five terms, and that it is known (somehow) that for any document to become one of the k highest-scoring documents it must contain one or the other of the final two terms. Then it is unnecessary to compute scores for documents that only include some subset of the first three terms – there is a sense in which the set of documents to be scored is based on a disjunction of two of the terms (the last two) rather than of all five terms.

In 2003 Broder et al. [4] introduced a mechanism for establishing exactly such a *critical set* of terms, assuming that DAAT processing is in effect. Their WAND mechanism (standing for "weak, or weighted and") can be applied to any similarity scoring mechanism subject to relatively unrestrictive constraints, and a range of experimentation has shown that it is a useful way of eliminating some fraction of the query-processing costs [4, 9, 10]. A promising enhancement to WAND style processing is to group inverted lists into fixed sized blocks and process block-wise using the BM-WAND algorithm [9, 10].

Our Contribution:

Our primary objective in this paper is to demonstrate that WAND is relatively unhelpful when a highly-effective Language Model (LMDS) similarity computation is undertaken and show that the numeric characteristics of the document similarity scores that are generated do not provide any useful reduction in workload. We also show that LMDS consistently outperforms BM25 in a wide

ADCS '13, December 05 - 06 2013, Brisbane, QLD, Australia

variety of querying scenarios, offsetting the value of any potential performance gains when using models other than LMDS. Finally, we show that BM-WAND and LMDS can be used together to achieve a good effectiveness and efficiency trade-off.

The remainder of the paper proceeds as follows: Section 2 gives an overview of two alternative similarity scoring functions, and provides more detail of TAAT and DAAT processing. The WAND process is described in detail in Section 3. In Section 4 we explore a wide variety of interactions between scoring metric and processing regime. We conclude in Section 5 and discuss future work based on our preliminary findings outlined in this paper.

2. BACKGROUND

In this paper we use two fundamental scoring metric for bag-of-words processing: BM25 and LMDS.

2.1 Ranking Metrics

First, we use the well-known BM25 bag-of-words similarity scoring function as a reference point for $S(q, \mathcal{D}_i)$, defined as:

$$\text{BM25} = \sum_{t \in q} \log\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \cdot \text{TF}_{\text{BM25}} \qquad (1)$$

where

$$\text{TF}_{\text{BM25}} = \frac{f_{t,d} \cdot (k_1 + 1)}{f_{t,d} + k_1 \cdot ((1 - b) + (b \cdot \ell_d / \ell_{\text{avg}}))} .$$

Here, N is the number of documents in the collection; f_t is the number of documents containing t; $f_{d,t}$ is the number of occurrences of t in document d; ℓ_d is the number of symbols in document d; and ℓ_{avg} is the average of ℓ_d over the collection. The values of k_1 and b can be tuned for specific collections to improve effectiveness, but we use the standard Okapi parameters $k_1 = 1.2$ and $b = 0.75$ suggested by Robertson et al. [17].

We also use the LMDS bag-of-words similarity scoring function (see, for example, Zobel and Moffat [21]):

$$|q| \cdot \log\left(\frac{\mu}{|d| + \mu}\right) + \sum_{t \in q \wedge d} \left(f_{q,t} \cdot \log\left(\frac{f_{d,t}}{\mu} \cdot \frac{|C|}{F_t} + 1\right)\right) , \quad (2)$$

where $|q|$ is the length of the query, $|d|$ is the length of the document, $f_{q,t}$ is the frequency of term t in the query, $f_{d,t}$ is its frequency t in the document, $|C|$ is the total number of terms in the collection, F_t is the number of occurrence of t in the collection, and μ is a smoothing parameter set to 2,500 in this instance.

2.2 Efficient Index Processing

As mentioned previously, two main approaches to processing postings lists are commonly used – term-at-a-time (TAAT) and document-at-a-time (DAAT). Exhaustive TAAT processing uses a fixed number of accumulators, and each term / document rank contribution is calculated incrementally. The performance boost arising from sequential processing should not be underestimated, and applies at all levels of the memory hierarchy. The alternative approach is to process all of the terms simultaneously, one document at a time [8]. The advantage of this approach is that the final rank score is known as each document is processed, so it is relatively easy to maintain a heap containing exactly k scores. The disadvantage is that all of the term posting lists are cycled through for each iteration of the algorithm requiring non-sequential disk reads for multi-word queries.

In addition to the traversal ordering, a processing regime can also be classified as *score-safe* or *heuristic*. A score-safe processing algorithm guarantees that identical top-k ordering will occur relative to a set ranking metric. Score-safe algorithms are currently receiving a great deal of attention in the research community. Like many trade-offs, providing only a best-guess or approximation can often dramatically improve the efficiency of processing. It is also important to point out the using a heuristic processing approach does not necessarily mean that effectiveness will always be worse than a score-safe or exhaustive processing regime. The complex interaction between text processing and statistical scoring methods can often have difficult to explain outcomes. Nevertheless, being able to provide score safety relative to a fixed ranking metric remains an attractive property for a processing method to have, even if only for the theoretical benefits.

2.3 Heuristic Processing

Exhaustive TAAT processing can be very efficient. Buckley and Lewit [7] proposed one of the earliest heuristic processing methods. The key idea is to use a heap of size k to allow posting lists to be evaluated in TAAT order. Processing is terminated when the sum of the contributions of the remaining lists cannot remove an item from the min-heap. Moffat and Zobel [15] improved on this pruning approach with two heuristics: STOP and CONTINUE. The STOP strategy is somewhat similar to the method of Buckley and Lewit, but the terms are processed in order of document frequency from least frequent to most frequent. When the threshold of k accumulators is reached, processing stops. In contrast, the CONTINUE method allows the current accumulators to be updated, but new accumulators cannot be added. These accumulator pruning strategies only approximate the true top-k result list.

In addition to the early work of Buckley and Lewit, a wide variety of heuristic approaches to efficient top-k document retrieval have been reported over the years [1, 4, 6, 7, 16, 18]. The TAAT approach can be made even more efficient by eliminating the need to do the full similarity scoring calculation [1, 2, 16]. The key idea of *impact ordering* is to precompute the TF for each document a term appears in. Next, quantize the TF values into a variable number of buckets, and sort the buckets (or blocks) for each term in decreasing impact order. Now, the top-k representative can be generated by sequentially processing each of the highest ranking term contribution blocks until a termination threshold is reached. Anh and Moffat refer to this blockwise processing method as *score-at-a-time* processing. Despite not using the full TF contribution for each term, Anh and Moffat demonstrate that the effectiveness of impact ordered indexes is not significantly reduced, but efficiency is dramatically improved.

2.4 Score-Safe Processing

One attractive property of DAAT style processing is that the traversal method is easily amenable to a variety of score-safe processing enhancements. The most widely used pruning strategy for DAAT is MAXSCORE [19]. For example, the `Indri` search engine currently implements a variant of MAXSCORE referred to as *Term Bounded Max Score* [18]. Turtle and Flood [19] observed that the BM25 TF component can never exceed $k_1 + 1 = 2.2$. So, the total score contribution for any term is at most $2.2 \cdot \log(N/N_t)$. Using this observation, Turtle and Flood present an algorithm that allows posting values below the threshold to be skipped. As the minimum bounding score in the heap slowly increases, more and more postings can be omitted. Enhanced DAAT pruning strategies similar in spirit to MAXSCORE have been shown to further increase efficiency [4, 18].

Turtle and Flood also describe a similar approach to improve the efficiency of TAAT strategies. However, the TAAT variant is

more complex than the DAAT approach as it requires an ordered candidate list of k documents to be maintained. The candidate list is used to skip document postings in each term list which could not possibly displace the current top-k documents once the heap contains k items.

Fontoura et al. [11] compare several TAAT and DAAT based inverted indexing strategies. The authors present novel adaptations of MAXSCORE and WAND [4] to significantly improve query efficiency of inverted indexes. The authors go on to show further efficiency gains in DAAT style processing by splitting query terms into two groups: rare terms and common terms. The exact split is based on a fixed threshold selected at query time. All of the strategies explored by Fontoura et al. are score-safe.

In response to the growing interest in score-safe processing and the promising efficiency gains reported for WAND, Ding and Suel [10] investigated several practical enhancements to better support blockwise compressed list processing. Dimopoulos et al. [9] go on to present even more aggressive pruning strategies for the BLOCK-MAX WAND (BM-WAND) style indexes. In Section 3, we carefully examine WAND and BM-WAND score-safe processing strategies, and discuss the interaction between scoring metric and processing regime.

2.5 Other Related Work

Other recent work on query processing techniques includes that of Broschart and Schenkel [5], who precompute term contributions and term-pair contributions at index construction time, and store a pruned index that is guaranteed to not exceed a supplied maximum size. Queries are resolved by directly summing contributions stored in the index lists, and execution is very fast.

Even more recently, Konow et al. [13] have described the use of *treaps* to store postings lists, using a dual structure that allows efficient generation of postings both in increasing document order, when required, and also in decreasing weight order, when required. This structure is also amenable to fast query processing, but like the term-pair index of Broschart and Schenkel, requires early exit to achieve those gains, and hence cannot assure the correctness of the score computation. In contrast, the three WAND-based approaches summarized here ensure that the same ranking through to depth k is generated as would arise from full and exhaustive processing.

3. WAND PROCESSING

This section describes the WAND mechanism, first introduced by Broder et al. [4].

3.1 Standard WAND Processing

Algorithm 1 describes the WAND approach. A query q of $|q|$ terms is to be processed using an index $\mathcal{I}$ which contains a postings list $\mathcal{I}_t$ for each term t that appears in the collection. Each posting can be thought of as being a tuple $(d, w_{d,t})$ indicating that the weight or strength of term t in document d is given by $w_{d,t}$, and are assumed to be accessed in document-number order. Postings lists are accessed via three functions: *first_posting*$(\mathcal{I}_t)$, which creates an iterator for term t, and returns the first posting from it; *next_posting*$(\mathcal{I}_t)$, which advances the iterator for term t, and returns the next posting in $\mathcal{T}_t$; and *seek_to_document*$(\mathcal{I}_t, d)$ which advances the iterator for term t to the first document number greater than or equal to d, and returns that posting.

In DAAT processing, all postings lists are concurrently open, and one candidate posting (c_t, w_t) is maintained for each term. Step 9 is key to the WAND scoring process; it reorders the candidate postings by their document number. Steps 10–19 then process the candidates in that document-number order, and determine the

Algorithm 1 WAND processing.

```
    function WAND(q, I, k)
        for t ← 0 to |q| − 1 do
            U[t] ← max_d{w_d | (d, w_d) ∈ I_t}
            (c_t, w_t) ← first_posting(I_t)
 5:     end for
        θ ← −∞                                   // current threshold
        Ans ← { }                                // k-set of (d, s_d) values
        while the set of candidates (c_t, w_t) is non-empty do
            permute the candidates so that c_0 ≤ c_1 ≤ ··· c_{|q|−1}
10:         score_limit ← 0
            pivot ← 0
            while pivot < |q| − 1 do
                tmp_s_lim ← score_limit + U[pivot]
                if tmp_s_lim > θ then
15:                 break, and continue from step 20
                end if
                score_limit ← tmp_score_lim
                pivot ← pivot + 1
            end while
20:         if c_0 = c_pivot then
                s ← 0                            // score document c_pivot
                t ← 0
                while t < |q| and c_t = c_pivot do
                    s ← s + w_t                  // add contribution to score
25:                 (c_t, w_t) ← next_posting(I_t)
                    t ← t + 1
                end while                        // s is the score of document c_pivot
                if s > θ then                    // and is a possible top-k answer
                    Ans ← insert(Ans, (c_pivot, s))
30:                 if |Ans| > k then
                        Ans ← delete_smallest(Ans)
                        θ ← minimum(Ans)
                    end if
                end if
35:         else                                 // can't score c_pivot (yet)
                for t ← 0 to pivot − 1 do
                    (c_t, w_t) ← seek_to_document(I_t, c_pivot)
                end for                          // all pointers are now at c_pivot or greater
            end if
40:     end while
        return Ans
    end function
```

earliest document for which the sum of the terms' scores can possibly exceed θ, where θ is the lowest score associated with any of the k highest-scoring documents encountered until this point. Critical to the success of this computation is the definition of "possibly exceed"; the *score_limit* is computed by summing the maximum weight $U[t]$ associated with each term t, computed at step 3 as the maximum contribution made to any document's score by term t. The $U[t]$ values should also be permuted in conjunction with step 9, and need to be weighted if the similarity computation includes a non-unit term-frequency component; in that regard, Algorithm 1 contains a number of small simplifications. In an implementation, the values $U[t]$ are precomputed for every term t that occurs in the collection, and stored as part of the vocabulary or as part of the inverted index.

Compared to exhaustive evaluation (that is, processing every posting that appears, and computing a similarity score for every document in the union of the postings lists), Algorithm 1 only scores documents that have the possibility – however remote it may

Algorithm 2 WAND processing with query-independent document weights, replacing steps 10–21 in Algorithm 1.

score_limit $\leftarrow 0$
doc_limit $\leftarrow -\infty$
pivot $\leftarrow 0$
while *pivot* $< |q| - 1$ **do**
  tmp_s_lim $\leftarrow$ *score_limit* $+ U[$*pivot*$]$
  tmp_d_lim $\leftarrow \max($*doc_limit*$, UD[$*pivot*$])$
  if *tmp_s_lim* $+ |q| \cdot$ *tmp_d_lim* $> \theta$ **then**
    break, and continue from step 14
  end if
  score_limit $\leftarrow$ *tmp_s_lim*
  doc_limit $\leftarrow$ *tmp_d_lim*
  pivot $\leftarrow$ *pivot* $+ 1$
end while
if $c_0 = c_{pivot}$ **then**
  $s \leftarrow |q| \cdot$ *doc_weight*$[c_{pivot}]$ // score document c_{pivot}
  ... remaining computation as described in Algorithm 1
end if

be – of making the final top k. Documents which only contain low-weight terms are bypassed at step 37, when their iterators are stepped past them while seeking postings for documents that do have potential. Function *seek_to_document*$(\mathcal{I}_t, c_{pivot})$ returns the first posting in $\mathcal{I}_t$ for a document equal to or greater than c_{pivot}. If the cursor for term t is already at document c_{pivot}, then no change is made.

The postings processed at step 24 represent work done towards scoring a document. But, as more documents are processed and the top-k estimate maintained in *Ans* is refined, θ increases, a decreasing fraction of future documents will need to be scored. Over the whole query, the number of postings processed at step 24 as a fraction of the total number of postings in the terms' lists is an important predictor of efficiency.

3.2 Language Models and WAND

The presentation in Algorithm 1 assumes that each posting makes an independent contribution to the score of a document, an arrangement consistent with the BM25 similarity calculation shown in Equation 1, with the document length component of the similarity computation incorporated in to every terms' weight. In contrast, Equation 2 results in a score that is not just the sum of the term weights – there is also a separate offset arising from the document-dependent part of the computation, the component $|q| \cdot \log(\mu/(|d|+\mu))$. This value is always negative, taking on high (only slightly negative) values for short documents, and low (very negative) values for long documents, and replaces the document-length normalization component of other similarity functions. That is, all other things being equal, this static document-dependent component means that a long document that contains the query terms will have a lower score than a shorter document, since there is more scope for the terms to appear in it by chance.

Algorithm 2 shows how the WAND processing regime is adjusted to cater for this additional need, with

$$\textit{doc\_weight}[d] = \log \frac{\mu}{|d| + \mu}$$

and

$$\textit{UD}[t] = \max\{\textit{doc\_weight}[d] \mid (d, w_d) \in \mathcal{I}_t\}$$

computed when the index is constructed, so that *UD*$[t]$ records the maximum document weight associated with any document

Algorithm 3 BM-WAND processing with query-independent document weights, extending Algorithm 1.

compute *pivot* as described in steps 1–13 of Algorithm 2
score_limit $\leftarrow 0$
doc_limit $\leftarrow -\infty$
for $t \leftarrow 0$ **to** *pivot* **do** // establish localized bounds
  $(c_t, w_t) \leftarrow$ *seek_to_block*$(\mathcal{I}_t, c_{pivot})$
  score_limit $\leftarrow$ *score_limit* $+ B[t]$*.max_term_wgt*
  doc_limit $\leftarrow \max($*doc_limit*$, B[t]$*.max_doc_wgt*$)$
end for
$t \leftarrow$ *pivot* $+ 1$ // check remaining terms too
while $t < q$ **and** $c_t = c_{pivot}$ **do**
  score_limit $\leftarrow$ *score_limit* $+ B[t]$*.max_term_wgt*
  doc_limit $\leftarrow \max($*doc_limit*$, B[t]$*.max_doc_wgt*$)$
  $t \leftarrow t + 1$
end while
pivot $\leftarrow t - 1$
if *score_limit* $+$ *doc_limit* $> \theta$ **then**
  $s \leftarrow |q| \cdot$ *doc_weight*$[c_{pivot}]$
  for $t \leftarrow 0$ **to** *pivot* **do** // compute full score for c_{pivot}
    $(c_t, w_t) \leftarrow$ *seek_to_document*$(\mathcal{I}_t, c_{pivot})$
    if $c_t = c_{pivot}$ **then**
      $s \leftarrow s + w_t$
      $(c_t, w_t) \leftarrow$ *next_posting*$(\mathcal{I}_t)$
    end if
  end for
  execute steps 28 to 34 of Algorithm 1
else // move out of current block combination
  $c \leftarrow 1 + \min\{B[t]$*.max_doc* $\mid 0 \le t \le$ *pivot*$\}$
  for $t \leftarrow 0$ **to** *pivot* **do**
    $(c_t, w_t) \leftarrow$ *seek_to_document*$(\mathcal{I}_t, c)$
  end for
end if

in which term t appears. Other static score components – in particular, those arising from page-rank computations, or other document-dependent factors that are to be included, such as URL length or language assessment – can be similarly accommodated by including a further per-term bound that is not multiplied by $|q|$.

3.3 Block-Max WAND

The postings lists $\mathcal{I}_t$ are typically represented as blocks of compressed document-identifier differences, each an associated w_d component, or some value (such as term frequency in the document) from which w_d can be directly computed. In the BM-WAND approach [9, 10], the blocks are made of fixed length (typically 64 or 128 postings, compressed as a single entity), and a localized *block maximum* is added to each block, being the largest w_d of any document d that appears in that block of $\mathcal{I}_t$. The maximum static score for any document in the block is also maintained if LMDS processing is to be supported.

Algorithm 3 shows how the WAND approach is further modified to exploit the additional information. Once a pivot has been identified, a second sharper bound is computed from the localized maxima for the set of corresponding blocks. Only after this second check has been performed are the relevant blocks decoded, and a full evaluation undertaken.

In Algorithm 3, $B[t]$ represents the active block in $\mathcal{I}_t$, that is, the one that contains the current candidate c_t. In that context, $B[t]$*.max_doc*, $B[t]$*.max_term_wgt*, and $B[t]$*.max_doc_wgt* represent, respectively, the largest document number d that appears in that block, the largest term weight w_d that appears in that block,

and the largest value of *doc_weight*$[d]$ for any document that has a posting within that block.

One additional function is required: *seek_to_block*$(\mathcal{I}_t, c)$ advances the cursor in list $\mathcal{I}_t$ to the start of the block that, if it exists at all, must contain the posting for document c. This function makes use of the values $B[t]$.*max_doc*, and does not need to decode any of the postings within any of the blocks that are bypassed, nor of the block that causes the scan to halt. The values of are also used in the *seek_to_document*$(\mathcal{I}_t, c)$ calls at step 29 – that final loop moves all of the cursors for terms through the pivot term through to the first document falls outside of the current block combination. This is valid, since the current block combination has been demonstrated to be unable to supply a candidate that can enter the heap.

The next section examines how these various mechanisms interact with the scoring regimes used in the BM25 and LMDS similarity computations.

4. EXPERIMENTS

4.1 Experimental Setup

To compare the efficiency and effectiveness of the algorithms, two experimental collections were used. The TREC 7 and TREC 8 *Ad Hoc* datasets were combined to make a small collection consisting of 1.86 GB (528,000 documents) of NEWSWIRE data from the *Financial Times*, *Federal Register*, *Los Angeles Times*, and *Foreign Broadcast Information Service* [20]. A total of 250 test topics and associated relevance judgments are available for this collection (topic numbers 301–450 and 701–800 in the 2004 Robust Retrieval Track). We refer to this collection as NEWSWIRE.

We also use the TREC GOV2 collection. It contains 420 GB of web pages (reduced to 90 GB of processed plain text) crawled from the `.gov` domain. We used the Boilerpipe[1] software package to generate the plain text version of the collection, used as the input for all three search engines tested [12]. Boilerpipe aggressively removes excess templating material and identifies the plain text body of each web document using a variety of machine learning techniques. A total of 150 test topics and associated relevance judgments are available for this collection (topic numbers 701–850 for the TREC *Ad Hoc* Terabyte Track running from 2004–2006. When testing effectiveness with both collections, we use the *title* field of each topic; this gives a short query consisting of 2 to 3 keywords, on average, and is representative of a common web search engine query.

To demonstrate that our effectiveness numbers are competitive with other state-of-the-art retrieval systems, we use `Indri`[2] and `Atire`[3] as baselines. For all systems tested, we use Krovetz stemming. Only the plain text body from each web page is indexed using TREC plaintext formatting, and identical plaintext is used in all of the systems, seeking to minimize the effect that variation in HTML parsing or tokenization might have across different implementations.

As suggested by Armstrong et al. [3], we also report the best known run from the original 2004 Robust Retrieval Track and the 2006 Terabyte Track. These "best" results may or may not be reproducible, since the systems are not public and complete details of the processing carried out are unavailable; nevertheless, they represent the effectiveness frontier that all systems should be striving to exceed.

[1] `https://code.google.com/p/boilerpipe/`
[2] `http://www.lemurproject.org/`
[3] `http://atire.org/`

4.2 Effectiveness

We first compare the effectiveness of BM25 and LMDS for the collections. Table 1 lists the effectiveness of several retrieval system using a variety of scoring regimes on the GOV2 collection; and Table 2 compares effectiveness on the NEWSWIRE collection. For clarity, we refer to our implementation as NewSys. We break down the runs based on *factor* and *similarity*. The factor `bow` indicates a simple bag-of-words query; and the factor `fdm` denotes the full dependency model of Metzler and Croft [14]. As the size of the collection increases, the LMDS scoring regimes increase in effectiveness relative to BM25. Note that we are not attempting to find the optimal tuning parameters here. In our preliminary tests, tuning BM25 and tuning LMDS did not change the observed trend of LMDS scoring being more effective than BM25 scoring, often with statistical significance at $p < 0.01$. More specifically, we performed a like-against-like comparison of effectiveness for each system. A paired t-test comparing the BM25 and the LMDS variants of each of `Indri` and NewSys showed statistical significance with $p < 0.01$ for MAP and NDCG in both test collections. In addition, `Indri` with LMDS is significantly better than `Indri` with BM25 for P@10 and NDCG@10 on the GOV2 collection.

That is, within each single system, the LMDS ranking is consistently better than the BM25 scoring for the collections and queries tested, although we reiterate that we did not seek to tune the BM25 parameters for the individual collections, nor even try and identify a single combination of parameters that gave the best overall performance across the two collections. It is relatively difficult to compare the scoring metrics across systems since there can be subtle differences between the implementation of scoring, stemming, and tokenization. That is why we have chosen to focus the comparisons within each system. All other things being equal in `Indri` or in NewSys, the LMDS scoring consistently does better than BM25, and adding in proximity or phrase components further increases the effectiveness of LMDS scoring.

4.3 Efficiency

We now investigate the impact of using WAND to improve the overall efficiency of our scoring regimes. Table 3 show the fraction of pointers that are processed in NewSys using BM25 and LMDS scoring and phrase sequences to improve overall effectiveness. We report the percentages for $k = 10$ and $k = 1{,}000$ since these two values are commonly used in the literature and in TREC run submissions. One thing that becomes immediately apparent is that there is an order of magnitude difference in the number of postings that must be scored. If a traditional TREC run is done with $k = 1{,}000$ using WAND and LMDS, nearly every document is scored. In these cases, it is clear that the additional overhead of managing the pivot provides no performance gain, and in many cases *reduces* overall efficiency. So, what is really happening here? How can the similarity metric have such a dramatic influence on the efficiency of WAND?

Figure 1 plots the distribution of BLOCK-MAX scores in the GOV2 index for all distinct terms appearing in queries 701–850 and queries 301–450 and 701–800 for NEWSWIRE, expressed as a percentage of the corresponding term global maximum scores. The scores were computed over blocks of 64 elements. For BM25, the BLOCK-MAX scores of each block are skewed heavily towards the global list max score. Thus, the list max score is a reasonable approximation of the score contributions of the postings in the list. For the LMDS similarity computation the overall BLOCK-MAX score distribution follows a normal distribution, and only a fraction of the blocks have a maximum score close to the list maximum

System	Factors	Similarity	P@10	NDCG@10	MAP	NDCG
Indri	bow	BM25	0.479	0.469	0.210	0.454
NewSys	bow	BM25	0.552	0.456	0.245	0.519
NewSys	bow	LMDS	0.563	0.466	0.276	0.554
Indri	bow	LMDS	0.569	0.465	0.280	0.558
Indri	fdm	LMDS	0.611	0.500	0.317	0.596

(a) Using Boilerpipe and a Krovetz stemmer.

Atire	bow	BM25	0.526	0.439	0.265	0.558
Best submitted	unknown	unknown	0.603	0.493	0.334	0.683

(b) Other systems (with unknown parsing and tokenization regimes).

Table 1: Measured effectiveness on GOV2, using topics TREC 701–850. The rows in each section of the table are ordered according to the final column, which is the NDCG score. The `Atire` run used the default BM25 ranking metric with $k_1 = 0.9$ and $b = 0.4$ and Krovetz stemming. The row titled "Best submitted" was `uwmtFadTPFB` in the TREC 2006 Terabyte Track.

System	Factors	Similarity	P@10	NDCG@10	MAP	NDCG
Indri	bow	BM25	0.425	0.430	0.235	0.505
NewSys	bow	BM25	0.432	0.442	0.241	0.513
Indri	bow	LMDS	0.425	0.438	0.249	0.522
NewSys	bow	LMDS	0.429	0.440	0.250	0.526
Indri	fdm	LMDS	0.438	0.452	0.264	0.537

(a) Using Krovetz stemming.

Atire	bow	BM25	0.446	0.454	0.258	0.544
Best submitted	unknown	unknown	0.513	0.523	0.333	0.626

(b) Other systems (with unknown parsing and tokenization regimes).

Table 2: Measured effectiveness on NEWSWIRE, using topics TREC 301–450 and 601–701. The rows in each section of the table are ordered according to the final column, which is the NDCG score. The `Atire` run used the default BM25 ranking metric with $k_1 = 0.9$ and $b = 0.4$ and Krovetz stemming. The row titled "Best submitted" was `pircRB04t3` in the TREC 2004 Robust track.

score. For LMDS the list maximum score is not a good indicator of the overall score contributions in the list. This observation holds for both test environments.

Method	$k = 10$		$k = 1{,}000$	
	Avg.	Med.	Avg.	Med.
Exhaustive	100.0	100.0	100.0	100.0
BM25, bow	3.5	1.0	28.0	11.7
LMDS, bow	19.2	10.5	83.9	100.0

(a) Using WAND.

BM25, bow	2.9	0.8	27.4	11.3
LMDS, bow	7.5	3.8	45.0	36.4

(b) Using BM-WAND.

Table 3: Fraction of pointers processed as a percentage of the total number of pointers associated with each query, GOV2, using TREC topics 701–850. Across the set of queries, the average number of postings per query for exhaustive processing is 1,460,562, and the median number of postings is 1,080,008. The percentages shown in the table are relative to these two numbers.

The discrepancy between the score distributions is responsible for difference in usefulness of the WAND processing method for BM25 and LMDS. During processing, WAND uses the list maximum score $U[t]$ to approximate the contribution of a term to the score of a document. If, for a given document, the sum of the list max scores lists containing the document is larger then the current threshold, a full evaluation is performed, even though the actual score of the document can be much smaller. In the case of BM25 where the list maximum score is a reasonable approximation of the scores in the list which implies the actual document score is often close to the score approximated by the list score maxima. For LMDS, this is not the case, as the list max scores often overestimate the actual score contribution of a term to the similarity score of a document which causes more documents to be evaluated.

Figure 2 shows the fraction of the total pool of documents for each query (that is, the size of the union of the postings lists) scored when evaluation is carried out to different depths k. For the GOV2 collection the WAND approach in conjunction with BM25 allows skipping more than 90% of all documents for all sorting depths. When the LMDS similarity measure is used, this is no longer the case. For $k = 10$, 20% of all postings are evaluated; and when $k = 1{,}000$, almost all documents are evaluated. This is a result of the poor approximation of scores in each postings list by the list maximum discussed above. The use of BM-WAND processing only marginally improves the number of evaluated documents over WAND for BM25, as the block maxima used to provide a more accurate estimation of the similarity score of a document are similar in magnitude to the corresponding list maximum.

In contrast, BM-WAND processing strongly decreases the number of documents evaluated when using a LMDS based similarity measure. The secondary estimation of the pivot document similarity score using the BLOCK-MAX scores, which for LMDS

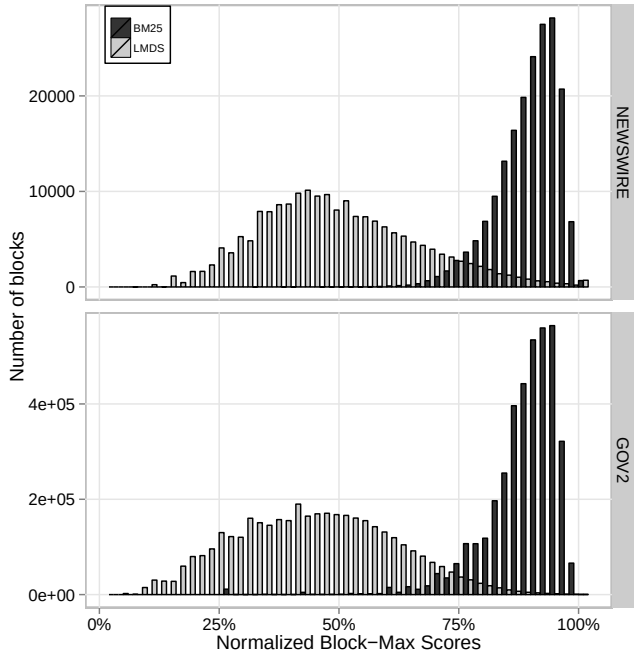

Figure 1: Distribution of BLOCK-MAX scores for all of the blocks associated with query terms arising in the two sets of test topics. The BLOCK-MAX score for each block in the postings list for term t was converted to a percentage of $U[t]$, the global maximum for that term, and then counted to determine the relative frequency of each fractional score.

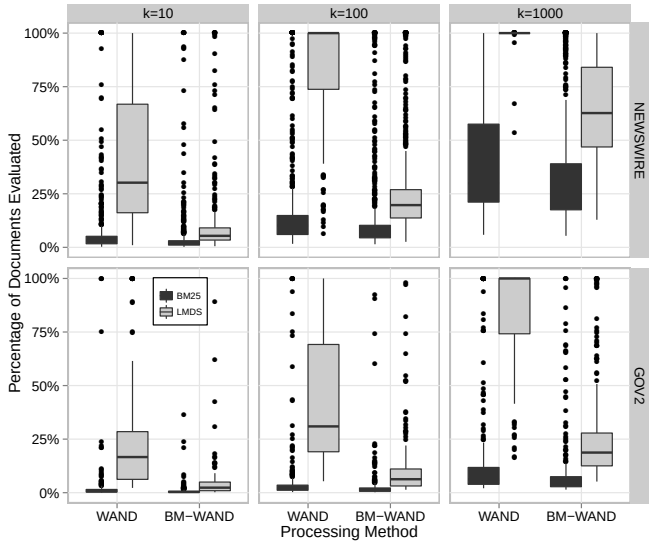

Figure 2: Number of documents scored, expressed as a fraction of the total number of documents containing any of the terms in that query. Three different retrieval depths are tested, for two different collections, for two different similarity computations, and for both WAND and BM-WAND processing.

are generally much lower than the list maximum score, allow additional documents to be skipped compared to WAND. For GOV2, this effect can be observed for all k. Interestingly, for the smaller collection (the NEWSWIRE collection of TREC-7/8) both methods perform worse for all k. Especially for $k = 1{,}000$, the performance of both WAND and BM-WAND degrades heavily. The small collection contains a total of $528,000$ documents. Thus, total number of relevant documents, and thus the size of the postings lists processed for each topic, is much smaller than for GOV2, which leads to a smaller fraction of the entries being skipped by both algorithms.

To further illustrate out the effect of different similarity measures, Figure 3 traces the evaluation of a single query, "`north korean counterfeiting`" (topic 808), against the GOV2 collection. Each dot in each pane represents a document that is scored, with the blue line indicating the current threshold θ at which scored documents qualify for entry to the heap. That is, documents whose points are above the line enter the heap and are potentially retained in to the top k; points below the line represent effort that is expended, but did not result in documents entering the heap. The objective of all pruning mechanisms is to reduce the number of evaluations that fall in to that second category. Documents that were not scored at all are represented by space; note that the horizontal axis is a proportional count relative to the size of the union of the postings lists.

Comparing the two left panes (WAND processing only) and the two right panes (BM-WAND processing) shows that the introduction of BLOCK-MAX information generates much greater savings in the LMDS scoring regime than in the BM25 scoring regime. Similar trends were observed for other queries and across both data sets. Note that scores cannot be compared between the top and bottom pairs of panes, because different computations are being undertaken.

5. CONCLUSION

We have explored the inherent trade-offs arising when two state-of-the-art top-k processing regimes are combined with effective similarity scoring models. We show that the dramatic efficiency gains reported in previous work that coupled BM25 scoring with the WAND approach can be eroded when the superior LMDS-based similarity scoring mechanism is employed. But when BM-WAND is employed substantial benefits to both BM25 and LMDS are provided for all values of k evaluated, with the benefits reducing similarly for both models as k increases relative to the collection size.

What we have not accurately measured yet is the impact that these complex tradeoffs have on actual querying time. In future work, we intend to delve deeper into these systems and measure the latency and overhead more precisely using an optimized implementation. Our initial experiments suggest that the length of the query may also affect the number of postings scored in BM25 and in LMDS scoring systems. In particular, it seems that long queries may gain greater relative advantage that short queries in BM25 scoring, but not in LMDS scoring, because LMDS normalizes score contributions relative to the number of terms in the query, and BM25 does not. It is also worth noting that BM-WAND reduces the number of documents that must be scored, but the additional arithmetic to make the pruning decisions has a cost that is not measured by simply counting the number of documents fully evaluated. We intend to continue our investigation, and extend our measurement regime to include the exact cost of these various score-safe processing regimes.

Acknowledgment

This work was supported by the Australian Research Council. Simon Gog participated in a number of helpful discussions. The referees provided a range of helpful comments.

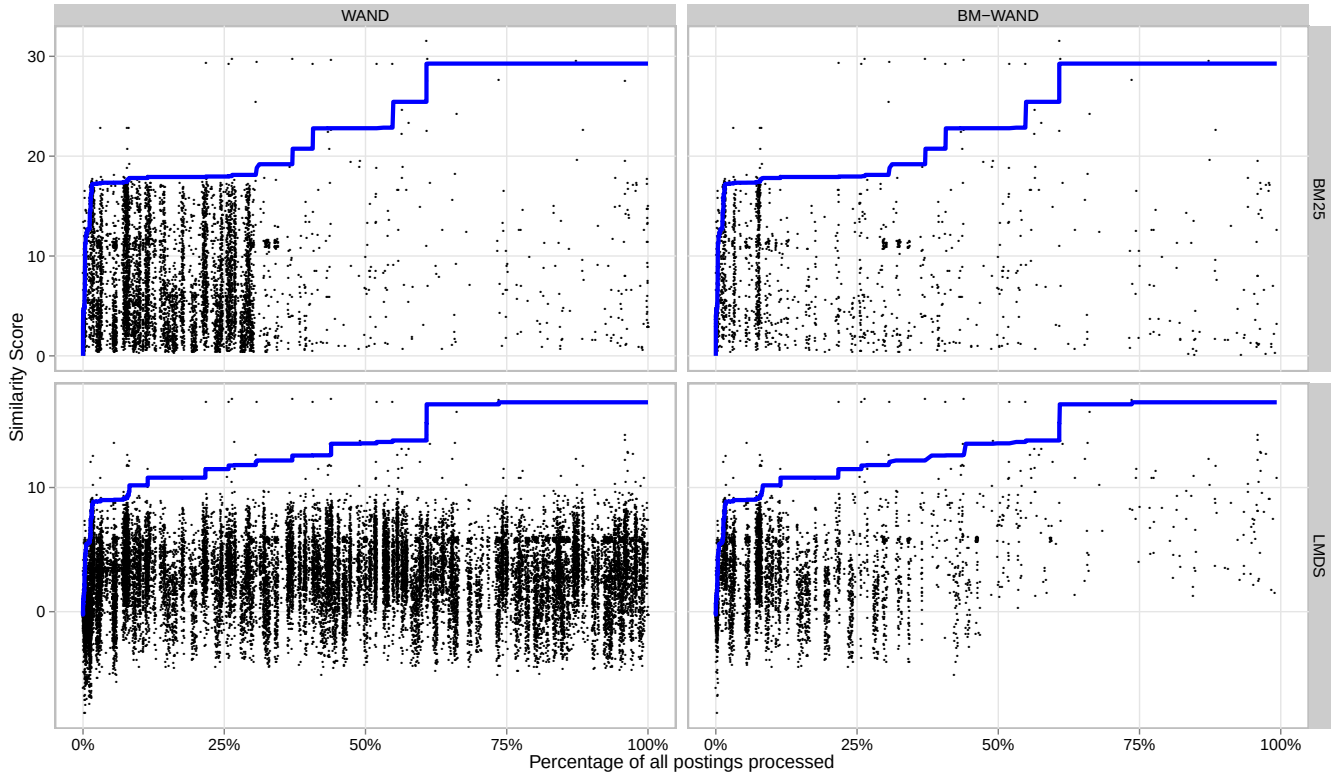

Figure 3: Distribution of evaluated documents as the postings lists for query "`north korean counterfeiting`" (topic 808) are evaluated for both WAND and BM-WAND processing using LMDS and BM25 similarity measures.

References

[1] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proc. SIGIR*, pages 372–379, 2006.

[2] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proc. SIGIR*, pages 35–42, 2001.

[3] T. G. Armstrong, A. Moffat, W. Webber, and J. Zobel. Improvements that don't add up: Ad-hoc retrieval results since 1998. In *Proc. CIKM*, pages 601–610, 2009.

[4] A. Z. Broder, D. Carmel, H. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. CIKM*, pages 426–434, 2003.

[5] A. Broschart and R. Schenkel. High-performance processing of text queries with tunable pruned term and term pair indexes. *ACM Trans. Information Systems*, 30(1):1–32, 2012.

[6] E. W. Brown. Fast evaluation of structured queries for information retrieval. In *Proc. SIGIR*, pages 30–38, 1995.

[7] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *Proc. SIGIR*, pages 97–110, 1985.

[8] S. Büttcher, C. L. A. Clarke, and G. V. Cormack. *Information Retrieval: Implementing and evaluating search engines.* MIT Press, Cambridge, Massachusetts, 2010.

[9] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. Optimizing top-k document retrieval strategies for block-max indexes. In *Proc. WSDM*, pages 113–122, 2013.

[10] S. Ding and T. Suel. Faster top-k retrieval using block-max indexes. In *Proc. SIGIR*, pages 993–1002, 2011.

[11] 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. VLDB Endowment*, 4(12):1213–1224, 2011.

[12] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proc. WSDM*, pages 441–450, 2010.

[13] R. Konow, G. Navarro, C. L. A. Clarke, and A. López-Ortíz. Faster and smaller inverted indices with treaps. In *Proc. SIGIR*, pages 193–202, 2013.

[14] D. Metzler and W. B. Croft. A Markov random field model for term dependencies. In *Proc. SIGIR*, pages 472–479, 2005.

[15] A. Moffat and J. Zobel. Self indexing inverted files for fast text retrieval. *ACM Trans. Information Systems*, 14(4):349–379, 1996.

[16] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency sorted indexes. *JASIST*, 47(10):749–764, 1996.

[17] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proc. TREC-3*, 1994.

[18] T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *Proc. SIGIR*, pages 219–225, 2005.

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

[20] E. M. Voorhees and D. K. Harman. Overview of the Eighth Text REtrieval Conference (TREC-8). In *Proc. TREC-8*, pages 1–24, 1999.

[21] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6–1 – 6–56, 2006.