Fast Bag-Of-Words Candidate Selection in Content-Based Instance Retrieval Systems

Michał Siedlaczek*
Tandon School of Engineering
New York University
Brooklyn, NY, USA
michal.siedlaczek@nyu.edu

Qi Wang*
Tandon School of Engineering
New York University
Brooklyn, NY, USA
qw376@nyu.edu

Yen-Yu Chen
Blippar, Inc.
Mountain View, CA, USA
yyc211@gmail.com

Torsten Suel
Tandon School of Engineering
New York University
Brooklyn, NY, USA
torsten.suel@nyu.edu

***Abstract*—Many content-based image search and instance retrieval systems implement bag-of-visual-words strategies for candidate selection. Visual processing of an image results in hundreds of visual words that make up a document, and these words are used to build an inverted index. Query processing then consists of an initial candidate selection phase that queries the inverted index, followed by more complex reranking of the candidates using various image features. The initial phase typically uses disjunctive top-k query processing algorithms originally proposed for searching text collections.**

Our objective in this paper is to optimize the performance of disjunctive top-k computation for candidate selection in content-based instance retrieval systems. While there has been extensive previous work on optimizing this phase for textual search engines, we are unaware of any published work that studies this problem for instance retrieval, where both index and query data are quite different from the distributions commonly found and exploited in the textual case. Using data from a commercial large-scale instance retrieval system, we address this challenge in three steps. First, we analyze the quantitative properties of index structures and queries in the system, and discuss how they differ from the case of text retrieval. Second, we describe an optimized term-at-a-time retrieval strategy that significantly outperforms baseline term-at-a-time and document-at-a-time strategies, achieving up to 66% speed-up over the most efficient baseline. Finally, we show that due to the different properties of the data, several common safe and unsafe early termination techniques from the literature fail to provide any significant performance benefits.

***Index Terms*—inverted index; bag-of-visual-words; image retrieval; candidate selection; top-k search; cascade ranking**

I. Introduction

The bag-of-words retrieval model represents text as a multi-set—or a bag—of words. Documents and queries form sparse vectors in the vocabulary space. An inverted index is a widely used structure under such assumptions [1], [2], [3], [4]. Results satisfying a search condition can be ranked based on a distance score between a query and a document. Typically, due to abundance of matching results, the user receives only a limited number of top k documents. We refer to such approach as top-k retrieval.

Modern search engines employ highly complex ranking functions, such as those based on learning-to-rank models [5], [6]. However, using them to rank matching documents is infeasible due to time restrictions. A widely used solution is to build a cascade ranking architecture [7], [8], which breaks down ranking into several cascades. The first cascade, also known as the candidate selection phase, ranks all matching documents using a very fast ranking function, and then selects a limited number of highest scoring results as candidates. The following cascades rerank smaller and smaller sets of candidates using increasingly better, but also more expensive, ranking functions. In this paper, we deal with the candidate selection phase.

It has been shown that the bag-of-words model generalizes beyond text search applications. In particular, Sivic and Zisserman showed how to extract visual words from video frames and store them in an inverted index [9]. They used standard text retrieval techniques to produce a set of candidates to be reranked by more precise methods. Since then, similar bag-of-visual-words (BoVW) approaches have been widely used for first-phase candidate retrieval in image search engines [10], [11], [12]. Nevertheless, previous work has focused on the effectiveness rather than efficiency of this phase, leaving many questions concerning performance of the BoVW model open.

State-of-the-art image detection methods employ convolutional neural networks (CNN) to generate feature vectors used to compare images in the database with incoming queries [13], [14], [15], [16]. While this yields good results, calculating distances between a query and a large number of images is undesirable in a number of scenarios. It can be found infeasible with respect to resource or latency requirements [12]. Therefore, a bag-of-words method might be necessary in the initial phase to limit the number of candidates that are then reranked using more advanced features. Local BoVW features can be retrieved from a CNN's activations to achieve this. Although CNNs have been shown to outperform SIFT in many applications, many cases still merit using SIFT, including retrieving: generic objects, gray-scale images, or images having rich textures or intense color changes [17].

In this paper, we deal with the following augmented reality scenario, which is currently deployed in a commercial setting: We are given a set of carefully curated images representing certain objects or categories of objects. A mobile app is provided to the user who can continuously point their phone camera at different objects and then receive information about these objects in real time. This requires a system that, given a

*Work performed during an internship at Blippar, Inc.

picture from a camera, retrieves its best match. As the system works in real time, it must have extremely low latencies. Thus, pictures are submitted to an inverted index, which generates candidates that are passed along to the next cascade, where the top result is selected.

We experiment with a BoVW inverted index containing images provided to us by Blippar[1] (later referred to as the BoVW index). These documents are a subset of images deployed in the production environment. Our goal is finding the most efficient first-phase query processing technique in the scenario described above.

Our contributions are as follows:

1) We analyze the quantitative properties of the BoVW index and associated queries, and compare them to those of the Clueweb09-B [18] text collection, a standard collection used for textual IR.
2) We compare the efficiency of the term-at-a-time and document-at-a-time strategies in our image search setting, and show the former to outperform the latter; we further describe optimizations that significantly improve the efficiency of the term-at-a-time approach.
3) We discuss the properties of both safe and unsafe early termination techniques, and show that they perform poorly on the BoVW index, or indices with similar properties.

The remainder of the paper follows the following structure. Section II provides background on inverted indices, retrieval strategies, and content-based image retrieval. Section III describes our experimental setup. In Section IV, we analyze the structure of the BoVW index and discuss differences between text and image search. Section V compares document-at-a-time and term-at-a-time strategies. In Section VI, we introduce a set of term-at-a-time optimizations, which significantly improve efficiency. In Section VII, we analyze the performance of common early termination techniques on the BoVW index. Finally, Section VIII provides a summary.

II. Background and Related Work

This section outlines top-k retrieval methods and puts them in the context of content-based instance retrieval (CBIR). We refer to the survey by Zobel and Moffat [4] for a more exhaustive description of text retrieval structures and algorithms.

A. Inverted Index

An inverted index (or inverted file) indexes a collection of documents, such as web pages, to enable fast document retrieval. Every distinct term contributes a list of postings (or pointers), each referring to a single document containing the term. Along with a unique ID of the document, it may contain (or point to) additional information, such as the term-document frequency, a pre-computed impact score, or a list of the occurrences within a document. Each posting list is commonly sorted by increasing document IDs, for fast joining of lists and effective compression. However, there are other ordering strategies to support a range of query processing techniques. We discuss those in the following sections.

B. Bag-Of-Visual-Words Model

Although inverted indices primarily deal with text corpora, they generalize to other applications, including CBIR. In this section, we briefly describe how an inverted index can be used in such a setting. We refer to the survey Zheng et al. [17] for a more in-depth explanation.

The first step is to extract local features from an image, using methods such as Scale-Invariant Feature Transform (SIFT) or Convolutional Neural Networks (CNN). The former detects a set of key points and their support regions in an image [19]. Then, a local descriptor is identified for each key point [20], [21], [22]. Both key points and their descriptors are intended to be robust in the presence of most geometric and photometric changes. Alternatively, the features can be retrieved from activations of the fully connected layers or the lower-level convolutional filters of a CNN. The latter are used to detect local visual patterns, and are more robust to image transformation [17], similar to the local invariant detectors used in SIFT. Various CNN architectures have been used, such as AlexNet [23], VGGNet [24], GoogleNet [25], and ResNet [26].

Since the total number of distinct features might be too large, they are quantized to a set of visual words (or codewords) by identifying cluster centroids to generate the final vocabulary. Each feature is assigned to one (hard assignment) or several (soft assignment) codewords. Soft assignment has been shown to improve retrieval effectiveness [27] but at the same time increases the index size. Determining the size of the vocabulary poses a challenge: a small vocabulary might fail to distinguish between distant descriptors, while a large one might generalize poorly. In consequence, the size of the vocabulary varies between implementations and applications [28], [29], [30].

The extracted codewords serve as a corpus for building an index. Typically, queries consist of codewords extracted from an input image by approximating its detected descriptors to the closest word, or words if soft assignment is used for queries, in the vocabulary. From that point forward, the search engine exploits text retrieval techniques during the candidate selection phase.

BoVW search can be used as a building block in many applications, such as finding similar images, or recognizing objects in images. The index we used in our work is a subset of an index deployed as part of an augmented reality (AR) system, where one of the subtasks is to recognize objects in front of a camera, in order to then retrieve additional information about these objects. This is achieved by querying an inverted index to find candidate matches in a set of images of known objects, followed by additional more specialized retrieval and recognition systems. We note that Hu et al. recently used a similar architecture for visual search in Bing [12], where a set of inverted indices distributed over many

[1] http://blippar.com

nodes reduces the size of the candidate set by a factor of 10,000.

C. Text Retrieval Strategies

We now describe common techniques for candidate selection in textual search engines. We focus on disjunctive queries, as these are necessary in image search scenarios due to the fact that a document rarely contains all query terms. We distinguish two major groups of query processing approaches. Exhaustive methods traverse all postings that satisfy the Boolean condition (in our case, a disjunction). Early termination (or dynamic pruning) techniques may choose to ignore some postings to speed up processing. These further divide into safe and unsafe methods. The former always return the same top-k results as an exhaustive search on the same ranking function, while the latter might yield different results.

The *term-at-a-time* (TAAT) strategy is arguably the most straightforward. Each posting list is traversed separately to accumulate scores for all documents, which determine the top k results. This simple, sequential nature allows for highly efficient implementations. On the other hand, accumulating all partial scores requires time as well as significant memory for score accumulators (which in turn might result in additional costs due to cache misses, as the accumulators usually do not fit in L1).

The *document-at-a-time* (DAAT) strategy [31] introduces an orthogonal approach: All posting lists are sorted in order of increasing document IDs and traversed simultaneously one document at a time, as if merged together. Since all relevant postings of one document are fully processed before those of the next, score accumulators become unnecessary, and the top-k documents can be collected in a small priority queue. This technique works especially well for conjunctive queries, where only documents containing all query terms need to be evaluated.

The *score-at-a-time* (SAAT) strategy is facilitated by impact-ordered indices, where postings are sorted by pre-computed partial scores (impacts), which are quantized by fixed-length integers. Thus, postings can be traversed in order of decreasing scores. This is particularly important for a type of safe early termination, which processes as many leading postings as allowed by the allocated budget or a stopping condition [32]. For simplicity, we refer to this technique as SAAT, implicitly assuming early termination.

D. Safe Early Termination

In this section, we describe three well known safe early termination approaches: Threshold Algorithm, WAND, and Max-Score.

a) Threshold Algorithm: Introduced by Fagin et al. [33], the Threshold Algorithm (TA) traverses all posting lists, sorted in descending order by their impact scores, in parallel: at step i, it considers the i-th posting from each list. The sum of the impact scores of these postings is the current threshold T, which either decreases or remains the same with each step. Any newly discovered document is fully scored using random access lookups into all of the lists. If at least k documents scored so far have a score of T or higher, then the algorithm can be terminated, as no new document can make it into the top-k results. The algorithm is well studied and analyzed in terms of query cost. It is known to work well in some scenarios, but can perform poorly in others due to the large number of random lookups. In particular, efficiency significantly drops as the number of query terms increases.

b) WAND: This algorithm, as proposed by Broder et al. [34], is based on a DAAT approach, and therefore keeps pointers to the currently processed posting for every query term. Additionally, the terms themselves are kept sorted by their current document IDs. Each step starts by accumulating the score upper bounds for the posting lists in order of increasing IDs until their sum reaches the current threshold T—the lowest score that can still possibly end up in the top-k results. The last term in the sum is called the pivot term. If all the terms up to the pivot point to the same document, we calculate its score. Otherwise, we advance all pointers to the pivot's document (or to the next greater document ID existing in that list), as no documents before that one can make it into the top-k. We repeat these steps until all of the terms are fully processed.

WAND improves upon DAAT by skipping documents that are known to have a score below the threshold. If enough of them exist, the processing can be sped up significantly. On the other hand, WAND introduces additional overhead for sorting terms and selecting the pivot, which can prove significant for large numbers of terms.

Block-Max WAND (BMW) [35] is a generalization of the WAND algorithm in which the score upper bounds are considered per block of a posting list. Recent work has shown that allowing variable length blocks can further improve query efficiency [36].

c) Max-Score: The family of Max-Score algorithms [31] relies on the score upper bounds for all query terms to partition them into essential and non-essential terms: Given a list of terms sorted by increasing upper bounds, the non-essential terms are those comprising the longest prefix of that list such that the sum of all lists' upper bounds is less than T—the current threshold for top-k results. As a consequence, no top-k result can occur only in the non-essential lists. The Max-Score approach comes in both DAAT and TAAT variants.

The DAAT version uses the above property of non-essential lists as follows: Since any top-k document must occur in at least one essential list, a DAAT union operation is performed first only within essential lists. Once a document's score for essential terms is known, we calculate its overall upper bound by adding all the non-essential score upper bounds. If it exceeds T, then it potentially belongs to the top-k results, and we must perform lookups in the non-essential lists to calculate the actual full score of the document. Otherwise, we ignore it and proceed to the next one. Every time T changes, we update the essential and non-essential lists to reflect the new threshold.

The TAAT variant simply processes terms—sorted by decreasing maximum score or by increasing posting list length, depending on the implementation—until the remaining upper bounds drop below T, which is increasing as we accumulate postings. After that, we only need to accumulate the scores of the documents that have already been scored. As with the DAAT version, we can perform lookups to the remaining lists utilizing block-skipping. Furthermore, after accumulating a posting, we can remove its document from the list of candidates if its score increased by the sum of the remaining upper bounds is less than T.

E. Long Queries

The queries we encounter in this work are much longer than those usually seen in text search engines. There has been some amount of previous work on longer text queries. One line of work has focused on longer queries issued by users to search engines, and how such queries should be processed [37], [38], [39], [40]. The focus in that work is on understanding and exploiting the characteristics of user text queries, which rarely have more than about 20 terms, and thus the conclusions are not very relevant to our scenario of image search queries derived from query images, where we have hundreds of query terms.

Another line of work going back more than two decades focuses on very long text queries that may derive from longer topic descriptions used as queries, or from automatic query expansion techniques, with evaluations taking place on older TREC collections. In particular, Kaszkiel and Zobel [41] showed that term-at-a-time processing is more efficient than document-at-a-time for longer queries—though these are still significantly shorter than our queries. Performance results for queries with up to about 300 terms on older TREC collections were for example reported by Ahn et al. [42] and Persin et al. [43]. The latter propose a filtering algorithm working on a frequency-ordered index that limits the number of accumulated postings by over 80% with little decrease in accuracy. Even though many improved early termination techniques have been proposed since, including SAAT strategies on impact-ordered indices [44], [45], we are unaware of any recent work that evaluates them on such long queries.

F. Alternative Approaches

Content-based image retrieval bears a resemblance to *nearest neighbor search*: based on visual features, we look for the indexed image (or images) that matches the query as closely as possible. Some algorithms based on indices such as k-d trees [46] and R-trees [47] address the issue of finding the nearest neighbors in an n-dimensional space. However, the complexity of those depends on the number of dimensions, and thus they perform poorly when n is large [48], [49], [50]. In fact, in practice it is quite difficult to obtain significant performance improvements over a simple sequential scan in the high-dimensional case [51], [52]. Since image retrieval is a high-dimensional problem, these approaches are unlikely to be efficient.

Fine-tuned single-pass convolutional neural networks reign supreme among object recognition systems due to their precision [17]. However, they can prove inefficient or ineffective for a large, generic domain [17], [12]. On the other hand, this approach can be preferable for search in limited, specialized domains. For example, Blippar implements a range of targeted neural networks, such as cars, logos, or dog breeds recognition, among others. Here, a single shot detector [53] detects categories and passes queries along to specialized engines, and an inverted index is used whenever no specialized engine can be selected by the detector.

III. Data and Setup

Our collection consists of roughly 2.6 million images made available to us by Blippar, which is a subset of the images uploaded to their production database. We use the OpenCV (opencv.org) implementation of SIFT [20] for key point and descriptor detection. Query image descriptors are converted to vocabulary codewords using an approximate k-nearest neighbors algorithm [54]. Unless stated otherwise, we use hard descriptor assignment.

When evaluating unsafe algorithms in Section VII, we report N-S scores based on the UK Bench [55] collection, a standard data set used for instance retrieval, which was incorporated into the collection for evaluation purposes. This collection consists of a set of photographs of real-life objects, where each object is photographed four times from different angles. The N-S score is the average number of correct, i.e., representing the queried object, images in the top 4 results (thus the score always falls between 0 and 4). However, since we want to evaluate the first-phase candidate retrieval independently of the re-ranking phase, we report the results assuming a perfect re-ranker that always puts the correct images first, provided they are in the candidate set. Therefore we count the number of the correct results in the top $k = 30$ documents.

In Section IV, we compare our collection to Clueweb09-B, a standard textual data set used by the Information Retrieval community. It contains roughly 50 million documents in English retrieved by a web crawler. We assume BM25 scoring for text retrieval.

Although many index compression methods exist [56], we leave our index uncompressed. This decision is dictated by a relatively small size (see Section IV), simplicity, and speed requirements. The index stores document IDs in a contiguous memory region for each list, along with another region containing consecutive partial impact scores, acquired from the visual processing of the indexed images (with tf-idf type weights). Both IDs and scores are represented by 4-byte integers. The the entire index is in main memory.

We ran our experiments on a server with an Intel Xeon E5-2670 v2 CPU @ 2.50GHz. We executed queries sequentially in a single thread. The latency is reported in milliseconds. We exclude time of tasks common to all methods, such as query parsing, fetching posting lists, and printing results.

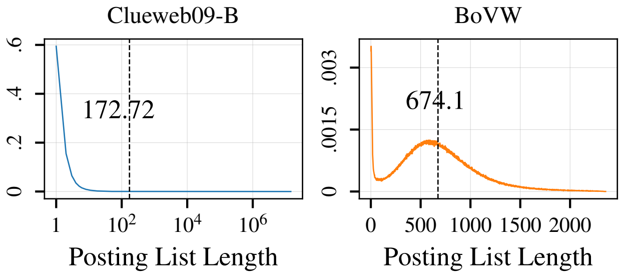

Fig. 1. The density of the distributions of posting list lengths for both Clueweb09-B and BoVW indices. The former is presented on a logarithmic length scale for improved readablility. The vertical line indicates the mean.

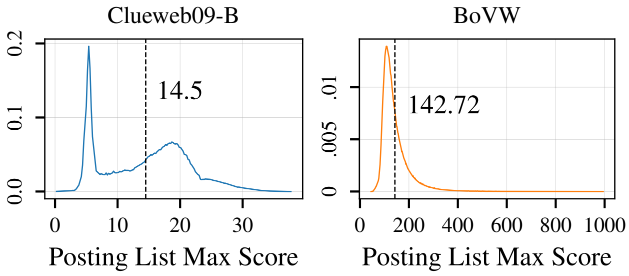

Fig. 2. The density of the distributions of the maximum impact score in a posting list, for both Clueweb09-B and BoVW indices. BM25 was used to score documents for Clueweb09-B, while the scores in BoVW are those acquired from the visual analysis of the indexed images. The vertical line indicates the mean.

The following type of box and whiskers plot is used throughout the paper[2]: the bar indicates the median; the diamond within the box, if displayed, indicates the mean average; the lower and upper hinges correspond to the first and the third quartile, respectively; the upper whisker corresponds to the largest value that is no further than 1.5 × IQR (interquartile range, i.e., the distance between the first and the third quartiles) from the upper hinge; the lower whisker corresponds to the lowest value that is no further than 1.5 × IQR from the lower hinge; outliers, if shown, are represented by dots outside of the whiskers.

IV. Quantitative Analysis

The quantitative properties of data can influence efficiency of retrieval algorithms to a great extent. In this section, we look into differences and similarities between text and image data distributions and discuss their potential impact.

a) Observation 1. Query Lengths: Image queries are on average two orders of magnitude longer (2.8 and 272 for Clueweb09-B and BoVW, respectively). Such long queries cause a large overhead of selecting the list from which to take the next document (as done in DAAT and SAAT strategies). The queries become even longer with soft assignment. For instance, Blippar also uses 3-descriptor assignments to improve precision, which increases the average query length to over 800 terms.

b) Observation 2. Posting List Lengths: The distributions of posting list lengths differ significantly. Short posting lists dominate the lexicon of the text index. In fact, unique words make up more than a half of all terms, and 75% of the terms occur at most 3 times. However, the most common words form extremely long posting list. Some early termination techniques can leverage this skew by effectively avoiding processing the long lists, or big parts of them, especially when the longer lists have lower impact scores. However, the distribution of the BoVW index is much more uniform (Figure 1), which may impact early termination efficiency.

c) Observation 3. Posting List Max Scores: Techniques such as WAND and Max-Score utilize maximum term scores for early termination. As seen in Figure 2, the maximum scores of visual words are more consistent than those of terms in the textual index. This is likely to have similar effect as uniformity of list lengths.

d) Observation 4. Length and Max Score Correlation: In textual indices, rare query terms, and therefore short posting lists, typically have high maximum scores due to the inverted document frequency component of most popular scoring functions, such as BM25 and cosine similarity. For example, for TREC evaluation queries run on ClueWeb09B, posting list lengths and maximum scores exhibit a strong negative correlation (Pearson correlation coefficient equal to -0.66). However, we found these to be largely uncorrelated in the queries submitted to the BoVW index. In fact, they exhibit a slight positive correlation (0.06), and thus higher frequency terms have higher maximum scores. This suggests potentially less advantage for techniques such as Max-Score.

e) Observation 5. Query Term Footprint: While the top textual results contain most of the query terms, content-based image search exhibits a very small *query term footprint*—the fraction of the query terms actually contained in an average top-k result. We found that the average term footprint for UK Bench queries is only 1.5% for $k = 30$. (We exclude the result identical to the query image, as it is implausible for a user to take a picture identical to one in the index.) It is even lower for our production queries, barely reaching 1.1%, as these include many queries that describe no existing object in the index. This essentially renders conjunctive queries useless. Moreover, we expect it to have negative impact for Max-Score type early termination methods, as there will be few non-essential lists to skip.

f) Observation 6. Index Size: We index relatively few images. Clueweb09-B contains over 50 million web pages, and commercial web engines index considerably more, whereas our collection consists of less than 3 million images. While we operate on a subset of an industrial collection, even the size of the full production instance is nowhere near the size of a large-scale web index. This gives us the advantage of fitting the index in main memory.

g) Observation 7. Accumulator Sparsity: We analyzed how many documents are scored on average (assuming exhaustive processing). When running a random TREC Web

[2] This is the default box and whiskers plot produced by the matplotlib library: https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.boxplot.html

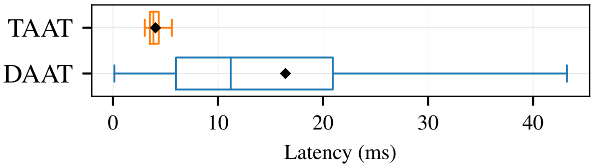

Fig. 3. Latency comparison between TAAT and DAAT strategies. Experiments were run for 10,000 randomly sampled production queries. Outliers are ignored for improved readablility.

Track query on the ClueWeb09B index, 15% of all documents have non-zero scores, while the same is true for only 8% of documents for the BoVW index on a random sample of production queries. The sparsity suggests that accumulating and aggregating results could be improved beyond the regular TAAT strategy.

V. DAAT and TAAT Comparison

This section compares two strategies of posting list traversal: DAAT and TAAT. We briefly describe our implementation and analyze the experimental results.

Due to *Observation 5*, we must implement the disjunctive version of the DAAT algorithm. To handle large numbers of terms, we keep pointers to current document IDs for each term in an optimized priority queue. Until the queue empties, we retrieve all pointers with the lowest current document ID, accumulate the document score, move these pointers to the next document ID, and insert them back to the queue. The accumulated scores are stored in another priority queue of size k. Therefore, the time complexity is $O(p \log q + m \log k)$, where p is the number of postings, q is the number of query terms, and m is the number of documents with a score above the top-k threshold at the moment of its accumulation.

Our TAAT implementation allocates memory for a full accumulator array, and sets its values to 0 before each query. Then, it sequentially traverses each posting list, accumulating scores in the array. Once finished, a priority queue aggregates the top-k documents in a single traversal of the accumulator array.

On average, this simple implementation significantly outperforms the DAAT strategy, mostly due to the large number of query terms (*Observation 1*), what has been also shown for long queries in case text retrieval [41]. We profiled both methods on a sample of 1,000 queries using cachegrind[3]. Indeed, we found that retrieving posting lists from and pushing them into the heap accounts for 75% of the instructions in DAAT. Furthermore, it suffers from a similar cache miss issue as TAAT: irregular posting list access along with large list numbers often causes the processor to miss the L1 cache during posting traversal.

As shown in Figure 4, DAAT performs better for short queries due to initialization and aggregation phases in TAAT, which are linear with respect to the collection size (we address this issue in the following section). However, short queries make up only a small fraction of the production query log. The large majority of the queries are longer than 70 codewords; for these, TAAT performs better: overall, its average latency is $4\,\text{ms}$ compared to $16.4\,\text{ms}$ for DAAT.

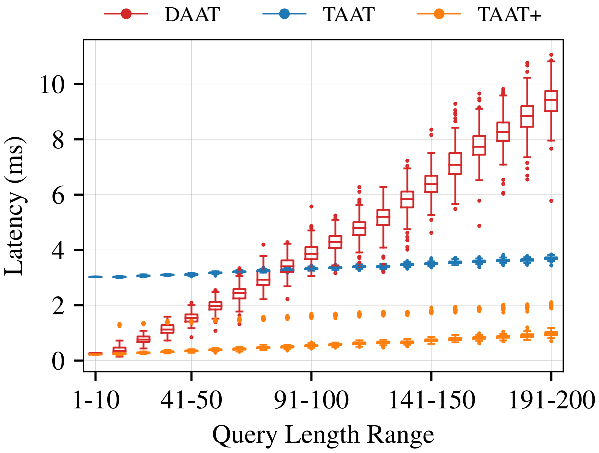

Fig. 4. Efficiency of TAAT and DAAT strategies for varying query length ranges based on a sample of production queries. The TAAT+ method includes the optimizations proposed in Section VI.

Having established the performance benefit, memory usage remains our main concern. Fitting the entire accumulator array in the main memory poses no issue in our scenario (*Observation 6*). Indeed, one accumulator array currently uses a little over 10 MB, which makes up barely a fraction of the entire index (11 GB). Since the index is kept in main memory, its size would become an issue much faster than the accumulator array. In that case, partitioning of the entire index would most likely take place.

However, a larger accumulator array might exceed the L2 (or L3) cache size, slowing down processing. In that case, we could implement a document-range-at-a-time (DRAAT) strategy: (1) partition the documents into ranges that fit in the L2 cache, (2) process them one by one in TAAT fashion, and (3) aggregate the results from all ranges. We could potentially take it one step further and decrease the range sizes to fit in the L1 cache. However, L1 is orders of magnitude smaller than L2, and therefore the ranges would have to be so small as to contain only a few postings from each list, making the traversal inefficient. Preliminary experiments with such small document ranges showed a significant slowdown for query processing, so we did not pursue this further.

VI. TAAT Optimizations

In this section, we describe further improvements to the TAAT strategy. We summarize the experimental results in Figure 5. Table I breaks down the three subsequent stages of improvements into the major TAAT processing phases: (1) accumulator array initialization, (2) posting list traversal and score accumulation, and (3) top-k result aggregation. In addition to hard assignment results, we include results obtained using term query soft assignment: each key point is assigned to

[3] http://valgrind.org/docs/manual/cg-manual.html

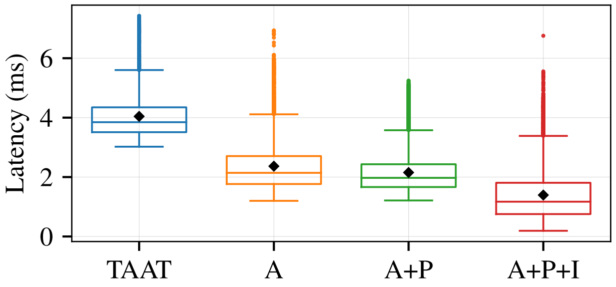

Fig. 5. Efficiency of the proposed TAAT optimizations.

Hard Asgmt.	**Init.**	**Traversal**	**Aggregation**	**Total**
TAAT	1.072	1.019	1.936	4.027
TAAT+A	1.074	1.136	0.133	2.343
TAAT+A+P	1.066	0.886	0.129	2.081
TAAT+A+P+I	0.135	1.059	0.180	1.374
Soft Asgmt.	**Init.**	**Traversal**	**Aggregation**	**Total**
TAAT	1.076	3.062	1.951	6.089
TAAT+A	1.078	3.381	0.129	4.588
TAAT+A+P	1.081	2.809	0.130	4.020
TAAT+A+P+I	0.137	3.317	0.181	3.635

TABLE I

Average elapsed time in the three TAAT phases, for consecutive stages of the optimizations. In the soft assignment case, we assign 3 codewords to each local descriptor of an image query.

3 visual words, and thus the queries are effectively three times longer. However, we still use hard assignment for indexed documents.

We only consider direct-mapped array-based accumulators. Although using a hash table eliminates the initialization phase, our experiments showed both traversal and aggregation phases to be significantly slower.

A. Block Aggregation (A)

Finding the top-k results among all processed documents accounts for a major portion of processing time. We aim to improve this by partitioning the accumulator array into blocks of documents of some fixed same size b. Throughout the traversal phase, we also maintain the maximum score in each block. When aggregating results, a block can be skipped whenever its maximum score is lower than the k-th result aggregated so far. Even though this requires some overhead during posting traversal, we experienced an overall efficiency improvement due to very fast top-k aggregation in the final phase (over 10 times faster on average).

Two main factors determine the efficiency of this method: the number of processed postings and the size of the accumulator array. We gain the most when the ratio of the number of postings to the collection size is the lowest. We see this in Table I: we improve overall latency by 43% when using hard assignment but only 25% once we triple the number of terms using soft assignment. Furthermore, the efficiency depends on the size b of the blocks. We empirically found $b = 1024$ to yield the best results among the tested values.

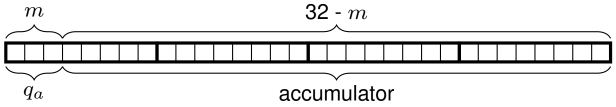

Fig. 6. An illustration of the lazy initialization described in Section VI-C. Each accumulator is shrunk to make room for the query counter. Whenever the accumulator's counter q_a is smaller than the current query counter q, the entire accumulator is erased and q_a is updated to q. Whenever the current query counter q cycles back to 0, all accumulators are cleared.

B. Prefetching During Traversal (P)

Due to random access to the accumulator array, TAAT traversal experiences many L1 cache misses—almost 50% of all the accumulator access instructions. We use pre-fetching directives to ensure that the accumulators are fetched by the processor in advance, eliminating some of the cache misses.

Effectively, we hint the processor to fetch the accumulator associated with a document l postings ahead of the currently processed one (we empirically selected l to be equal to 3). Therefore, the accumulator is usually already in L1 when we need it. Additionally, we inform the processor that the pre-fetched memory will be used for both reading and writing, and can be evicted right after. As shown in Table I, we were able to speed up posting traversal by approximately 20%.

C. Accumulator Initialization (I)

Many accumulators are unused during query processing (*Observation 7*), yet they still must be initialized—a task taking over $1\,\mathrm{ms}$ per query. To avoid it, we attempted to leverage our accumulator blocks from Section VI-A, and only re-initialize those blocks that have non-zero maximum score. This closely resembles the method proposed by Jia et al. [57], except that we use the existing structure to partition the accumulator array. However, the scored documents are scattered across the accumulator array, and we are rarely able to skip any blocks during the initialization. Therefore, we observed no improvements for this method. Even with the blocks as small as 128 documents, we found that only 2% of the accumulators could be skipped during initialization, at which point the comparison overhead significantly outweighed the negligible speedup.

As another attempt to speed up the initial phase, we introduce a lazy initialization scheme. We reserve an m-bit prefix of each accumulator for a query counter: a sequential query index modulo 2^m, denoted as q (Figure 6). We fully erase the accumulator array only when $q = 0$. Otherwise, we proceed to posting traversal right away. However, before an accumulator update, we must erase it if its prefix q_a is lower than q (i.e., the existing accumulator value was calculated for another query) and update the prefix to q. Otherwise, we simply add the partial score.

The larger the value of m, the less often we have to do a full initialization. However, it introduces a minor overhead for each posting that is traversed, and also during aggregation. Furthermore, we need to store the query index q, which uses m bits of the accumulator, and therefore m must be relatively small. We experimentally found no further improvement for

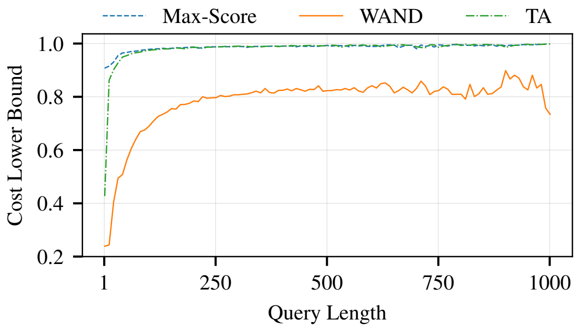

Fig. 7. Conservative lower bounds on the relative cost of early termination techniques averaged for each query length. The costs are defined as follows. *Max-Score*: the fraction of the postings that are in the essential lists; *WAND*: the fraction of the postings that must be visited; *TA*: the fraction of the postings that must be visited, excluding random lookups.

$m > 3$, and therefore set $m = 3$ in our experiments, as reported in Table I.

On average, the initialization phase now takes only an eighth of what it used to, as the accumulators are fully erased every eighth query—we can observe these queries as the outliers in Figure 4. In case we require execution time to distribute more evenly, we could initialize a different subset of the accumulators each time.

On the other hand, for every posting, we take an additional step to determine whether to clear the score; similar extra work is required in aggregation. Nevertheless, we observe an overall decrease in latency, as this overhead is small.

D. *Summary*

We have experimented with four optimization techniques for the TAAT approach, and found three of them to decrease query time. We substantially leveraged accumulator sparsity (*Observation 7*) to speed-up initialization and aggregation phases, and utilized the compiler's prefetch instructions to eliminate many L1 cache misses, thus improving the performance of posting traversal. Overall, we decreased average query time by 66% for hard and 40% for soft query term assignment.

VII. Early Termination

In this section, we examine a list of well-known early termination concepts in terms of the efficiency of querying the BoVW index. We first analyze safe approaches: TA [33], WAND [34], and Max-Score [31], and then evaluate the time-quality trade-off of an unsafe SAAT strategy. We chose these approaches as they represent a wide spectrum of early termination methods. We believe that such algorithms as those proposed by Anh and Moffat [32], or Strohman and Croft [58], or the Block-Max WAND algorithm [35], [36] would make little difference in the overall picture, as they are based on similar basic ideas as TA and WAND. In this section, we show that known early termination techniques seem to provide little benefit on the BoVW index. We chose to report results of a simulation in order to avoid having to go into too many implementation details. Instead, we aim to show that no implementation can obtain a significant improvement.

A. *Threshold Algorithm (TA)*

We simulated TA [33] to calculate when the stopping condition occurs and found that 98% of the postings had to be traversed (excluding random lookups) before the k-th score grew above the threshold. Therefore, we are unable to improve the speed we achieved with the TAAT strategy.

TA performs poorly for a few major reasons: First, each document exist in only few out of many posting lists. Therefore, the scores of the top-k documents are low relative to the threshold T. Furthermore, the scores within a posting list decrease slowly: we found that, on average, the minimum score is only 47% smaller than the median. Finally, TA is known to slow down with increasing numbers of terms. In fact, its time complexity becomes closer and closer to linear in the number of indexed documents for extremely long queries. Figure 7 shows the average fraction of the postings that must be processed for different query lengths.

B. *WAND*

Under favorable circumstances, WAND [34] can significantly decrease the number of postings that are processed. The best improvements can be made when the following occurs frequently during the processing: (1) many posting lists point to a lower document than the pivot, and (2) the difference between the lowest document ID and the pivot document ID is large. When these two hold, many postings can be skipped. Otherwise, the improvements might be insufficient to make up for the overhead of selecting the pivot, which can be significant.

In our scenario, we found no cost benefit, due to very few postings being skipped during traversal (Figure 7). The average number of posting lists preceding the pivot is only 3.3, while the average number of terms is over 200. Almost 80% of the postings had to be visited on average, and over 70% of them had to be evaluated (in our case, evaluation means reading the partial score, multiplying it by the query-term weight, and adding it to the accumulator variable). Considering the additional overhead of pivot selection, we found these savings to be insufficient to improve upon the DAAT baseline.

C. *Max-Score*

Let us assume that, given a query, we know the final score of the k-th retrieved document. We denote it as T and use it as a threshold during query processing: no document can be returned if we know its score to be lower than T. Following the Max-Score [31] approach, we partition the posting lists into essential and non-essential ones based on T. Doing so, we found 97% of terms, accounting for 98% of the postings, to be essential on average. Figure 7 shows the results for specific query length ranges.

Like TA and WAND, Max-Score suffers from the low scores of the top-k results, relative to the maximum scores in the posting lists. It is caused by very few of the query terms actually occurring in the top-k results. Furthermore, both the lengths and maximum scores of posting lists vary little, as

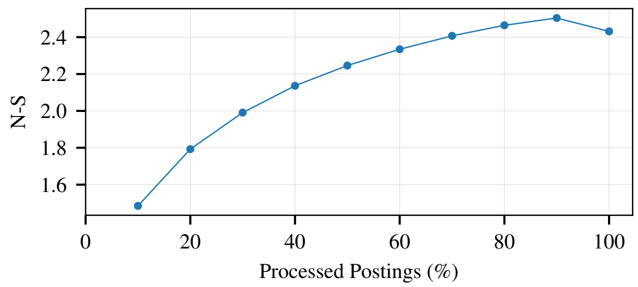

Fig. 8. Precision of the anytime SAAT method for different fractions of the processed postings.

stated in *Observations 2 and 3*. As a result, the Max-Score approach is unable to improve query efficiency.

D. SAAT

In the SAAT strategy [32], postings are traversed in order of decreasing impacts. The power of this method lies in its straight-forward application as an unsafe early termination technique. We maximize the collective impact of the processed postings by considering the ones with the largest partial impacts, discarding the rest. Although no guarantee regarding result quality can be made, it is a reasonable and well-known heuristic. Ideally, we would like to traverse only a small portion of all postings, yet experience little decrease in result quality. This approach was found to be very effective on text collections [44], [45].

Several potential SAAT implementation exist. Anytime ranking [44] selects the next highest impact among the posting lists at each step, until the maximum number of processed postings is reached. This ensures that the highest scored postings are always accumulated. However, it introduces a significant overhead for long queries, similar to that of the DAAT approach.

We note that some of the overhead of selecting the next posting from the lists of the query terms should be avoidable, say, by choosing an impact threshold, or a percentage of posting to be processed, and then performing a quick computation to identify the relevant postings. However, it is not clear how to select the right cutoffs, and the idea runs counter to the anytime-ranking approach. Nevertheless, our results show that even in this case any possible improvements would be very limited at best.

We implemented the anytime ranking algorithm to analyze the time-precision trade-off. Figure 8 shows average N-S scores for different fractions of processed postings. We found that at least 75% must be processed on average to retain the precision of the exhaustive methods. This is a significantly higher threshold compared with the reported 10% for textual indices [44], [45]. We found these savings to be insufficient to make up for the overhead introduced in the anytime ranking approach. Faster methods, such as those mentioned above, could be employed to remove the overhead, but it is unclear whether they can offer an acceptable trade-off, and even in the best case improvements would be limited to 25%.

VIII. Conclusion

Candidate selection is an important part of a CBIR system. Due to its high efficiency, an inverted index is often used in this phase. However, the quantitative properties of such an index differ significantly from those of a textual one, and therefore different query processing approaches might be needed.

We analyzed image and text data, and pointed out differences that may influence the efficiency of retrieval algorithms. We made several important observation, which later were backed up by experiments.

We compared TAAT and DAAT strategies and found the former to significantly outperform the latter due to high numbers of query terms. We further introduced a set of optimizations to the TAAT strategy. They reduced the overall query time by 66% and 40% with hard and soft query term assignment, respectively.

It was recently shown by Crane et al. [59] that some well-known safe early termination methods perform poorly for large k. We analyzed their efficiency in the image retrieval scenario and concluded that they are also inefficient for very long queries, even at low values of k. Furthermore, we showed that the SAAT strategy offers a trade-off between speed and quality on our data that is significantly worse than the previously reported results for textual data.

Nevertheless, some question remain open. Despite thorough analysis, we failed to find a suitable early termination technique. Further attempts could be made to propose novel approaches that increase the efficiency of a BoVW index (or indices of similar properties), or to provide more definite evidence that effective early termination is impossible under the presented circumstances. Furthermore, additional data analysis may unveil properties that could be exploited to further optimize the TAAT strategy.

Acknowledgement

This research was partially supported by NSF Grant IIS-1718680 ”Index Sharding and Query Routing in Distributed Search Engines”.

References

[1] D. Harman, E. Fox, R. Baeza-Yates, and W. Lee, “Inverted files,” *Information Retrieval, Data Structures & Algorithms*, pp. 28–43, 1992.

[2] J. Zobel, A. Moffat, and K. Ramamohanarao, “Inverted files versus signature files for text indexing,” *ACM Transactions on Database Systems (TODS)*, 1998.

[3] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine,” *Computer networks and ISDN systems*, 1998.

[4] J. Zobel and A. Moffat, “Inverted files for text search engines,” *ACM Computing Surveys*, 2006.

[5] L. Wang, J. Lin, and D. Metzler, “Learning to efficiently rank,” in *Proc. of the 33rd Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2010.

[6] T.-Y. Liu, “Learning to rank for information retrieval,” *Foundations and Trends in Information Retrieval*, 2009.

[7] C. L. A. Clarke, J. S. Culpepper, and A. Moffat, “Assessing efficiency–effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgments,” *Information Retrieval Journal*, 2016.

[8] L. Wang, J. Lin, and D. Metzler, “A cascade ranking model for efficient ranked retrieval,” in *Proc. of the 34th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2011.

[9] J. Sivic and A. Zisserman, “Video google: Efficient visual search of videos,” in *Toward Category-Level Object Recognition*, 2006.

[10] R. Shekhar and C. V. Jawahar, “Word image retrieval using bag of visual words,” in *Proc. of the 10th IAPR Intl. Workshop on Document Analysis Systems*, 2012.

[11] J. Liu, “Image retrieval based on bag-of-words model,” *arXiv preprint arXiv:1304.5168*, 2013.

[12] H. Hu, Y. Wang, L. Yang, P. Komlev, L. Huang, J. Huang, Y. Wu, M. Merchant, A. Sacheti *et al.*, “Web-scale responsive visual search at Bing,” in *Proc. of the 24th ACM SIGKDD Intln. Conf. on Knowledge Discovery & Data Mining*, 2018, pp. 359–367.

[13] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, “Learning compact binary descriptors with unsupervised deep neural networks,” in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2016.

[14] W.-L. Ku, H.-C. Chou, and W.-H. Peng, “Discriminatively-learned global image representation using cnn as a local feature extractor for image retrieval,” in *Visual Communications and Image Processing*, 2015.

[15] L. Gao, J. Song, F. Zou, D. Zhang, and J. Shao, “Scalable multimedia retrieval by deep learning hashing with relative similarity learning,” in *Proc. of the 23rd ACM Intl. Conf. on Multimedia*, 2015.

[16] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, and J. Li, “Deep learning for content-based image retrieval: A comprehensive study,” in *Proc. of the 22nd ACM Intl. Conf. on Multimedia*, 2014.

[17] L. Zheng, Y. Yang, and Q. Tian, “Sift meets cnn: A decade survey of instance retrieval,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[18] J. Callan, M. Hoy, C. Yoo, and L. Zhao, “Clueweb09 data set,” 2009. [Online]. Available: http://lemurproject.org/clueweb09/

[19] Y.-G. Jiang, C.-W. Ngo, and J. Yang, “Towards optimal bag-of-features for object categorization and semantic video retrieval,” in *Proc. of the 6th ACM international Conf. on Image and Video Retrieval*, 2007.

[20] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proc. of the 7th IEEE Intl. Conf. on Computer Vision*, 1999.

[21] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Proc. of the 9th European Conf. on Computer Vision*, 2006.

[22] T. Ojala, M. Pietikäinen, and D. Harwood, “A comparative study of texture measures with classification based on featured distributions,” *Pattern Recognition*, 1996.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, 2012.

[24] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proc. of the Conf. on Computer Vision and Pattern Recognition*, 2015.

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. of the Conf. on Computer Vision and Pattern Recognition*, 2016.

[27] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Lost in quantization: Improving particular object retrieval in large scale image databases,” in *2008 IEEE Conf. on Computer Vision and Pattern Recognition*, 2008.

[28] A. Mikulik, M. Perdoch, O. Chum, and J. Matas, “Learning vocabularies over a fine quantization,” *Intl. Journal of Computer Vision*, 2013.

[29] S. Lazebnik, C. Schmid, and J. Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories,” in *2006 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, 2006.

[30] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, “Local features and kernels for classification of texture and object categories: An in-depth study,” INRIA, Research Report RR-5737, 2005.

[31] H. Turtle and J. Flood, “Query evaluation: Strategies and optimizations,” *Information Processing & Management*, 1995.

[32] V. N. Anh and A. Moffat, “Pruned query evaluation using pre-computed impacts,” in *Proc. of the 29th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2006.

[33] R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” *Journal of computer and system sciences*, 2003.

[34] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, “Efficient query evaluation using a two-level retrieval process,” in *Proc. of the 12th Intl. Conf. on Information and Knowledge Mgmt.*, 2003.

[35] S. Ding and T. Suel, “Faster top-k document retrieval using block-max indexes,” in *Proc. of the 34th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2011.

[36] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini, “Faster blockmax wand with variable-sized blocks,” in *Proc. of the 40th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2017.

[37] M. Bendersky and W. B. Croft, “Analysis of long queries in a large scale search log,” in *Proc. of the 2009 Workshop on Web Search Click Data*, 2009.

[38] M. Gupta and M. Bendersky, “Information retrieval with verbose queries,” *Foundations and Trends in Information Retrieval*, 2015.

[39] S. Huston and W. B. Croft, “Evaluating verbose query processing techniques,” in *Proc. of the 33rd Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2010.

[40] J. Mackenzie, F. Scholer, and J. S. Culpepper, “Early termination heuristics for score-at-a-time index traversal,” in *Proc. of the 22nd Australasian Document Computing Symposium*, 2017.

[41] M. Kaszkiel and J. Zobel, “Term-ordered query evaluation versus document-ordered query evaluation for large document databases,” in *Proc. of the 21th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 1998.

[42] V. N. Anh, O. de Kretser, and A. Moffat, “Vector-space ranking with effective early termination,” in *Proc. of the 24th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2001.

[43] M. Persin, J. Zobel, and R. Sacks-Davis, “Filtered document retrieval with frequency-sorted indexes,” *Journal of the American Society for Information Science*, 1996.

[44] J. Lin and A. Trotman, “Anytime ranking for impact-ordered indexes,” in *Proc. of the 2015 Intl. Conf. on The Theory of Inf. Retrieval*, 2015.

[45] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna, “Toward reproducible baselines: The open-source ir reproducibility challenge,” in *Proc. of the 38th European Conf. on IR Research*, 2016.

[46] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, 1975.

[47] N. Roussopoulos, S. Kelley, and F. Vincent, “Nearest neighbor queries,” in *Proc. of the 1995 ACM Intl. Conf. On Management of Data*, 1995.

[48] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is nearest neighbor meaningful?” in *Proc. of the 7th Intl. Conf. on Database Theory*, 1999.

[49] A. Hinneburg, C. C. Aggarwal, and D. A. Keim, “What is the nearest neighbor in high dimensional spaces?” in *Proc. of the 26th Intl. Conf. on Very Large Databases*, 2000.

[50] D. Tunkelang, “Making the nearest neighbor meaningful,” in *SIAM Workshop on Clustering High Dimensional Data and its Applications*, 2002.

[51] R. Weber, H.-J. Schek, and S. Blott, “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces,” in *Proc. of the 24th Very Large Data Bases Conf.*, 1998.

[52] U. Shaft, J. Goldstein, and K. Beyer, “Nearest neighbor query performance for unstable distributions,” Department of Computer Science, University of Wisconsin, Tech. Rep., 1998.

[53] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *Proc. of the 14th European Conf. on Computer Vision*, 2016.

[54] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, “Object retrieval with large vocabularies and fast spatial matching,” in *Proc. of 2007 IEEE Conf. on Computer Vision and Pattern Recognition*, 2007.

[55] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” in *2006 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, 2006.

[56] I. H. Witten, T. C. Bell, and A. Moffat, *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1994.

[57] X.-F. Jia, A. Trotman, and R. OKeefe, “Efficient accumulator initialisation,” in *Proc. of the 15th Australasian Document Computing Symposium*, 2010.

[58] T. Strohman and W. B. Croft, “Efficient document retrieval in main memory,” in *Proc. of the 30th Ann. Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2007.

[59] 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. of the 10th ACM Intl. Conf. on Web Search and Data Mining*, 2017.