The Potential of Learned Index Structures for Index Compression

Harrie Oosterhuis
University of Amsterdam
oosterhuis@uva.nl

J. Shane Culpepper
RMIT University
shane.culpepper@rmit.edu.au

Maarten de Rijke
University of Amsterdam
derijke@uva.nl

ABSTRACT

Inverted indexes are vital in providing fast key-word-based search. For every term in the document collection, a list of identifiers of documents in which the term appears is stored, along with auxiliary information such as term frequency, and position offsets. While very effective, inverted indexes have large memory requirements for web-sized collections. Recently, the concept of learned index structures was introduced, where machine learned models replace common index structures such as B-tree-indexes, hash-indexes, and bloom-filters. These learned index structures require less memory, and can be computationally much faster than their traditional counterparts. In this paper, we consider whether such models may be applied to conjunctive Boolean querying. First, we investigate how a learned model can replace document postings of an inverted index, and then evaluate the compromises such an approach might have. Second, we evaluate the potential gains that can be achieved in terms of memory requirements. Our work shows that learned models have great potential in inverted indexing, and this direction seems to be a promising area for future research.

1 INTRODUCTION

Search engines make large collections of documents accessible to users, who generally search for documents by posing key-word based queries. For the best user experience, the user should be presented relevant results as quickly as possible. Inverted indexes allow systems to match documents with key-words in an efficient manner [15, 22]. Due to their scalability, inverted indexes form the basis of most search engines that cover large document collections. They store inverted lists of the terms contained in each document in the collection; for a given term, an inverted list stores a list with all the documents in which it occurs. An important search operation performed on these lists is conjunctive Boolean intersection, as it is routinely used in search engines for early stage retrieval [1, 4, 8] and for vertical search tasks such as product or job search [19]. Boolean queries are computed by intersecting the inverted lists of the query terms [7], and the result set typically includes documents that contain *all* of the query terms, *including* the stopwords.

Despite consistent advances in compressed index representations over the years [11, 16, 20], the cost of storing all relevant data (such as stopword data) to effectively search sizable collections can be a bottleneck in scaling to increasingly larger collections. One interesting alternative is to use a bitvector to store the document vector of high frequency terms [9, 14]. However, there are limits to what compressing exact representations can achieve.

Recently the idea of learned index structures has been proposed by Kraska et al. [10]. Here, machine learned models are optimized to replace common index structures such as B-tree-indexes, hash-indexes, and bloom-filters. The benefit of learned index structures is that they require less memory and can be evaluated substantially faster than their traditional counterparts.

In this study we examine whether learned index structures can be used to reduce space in a Boolean search scenario, and investigate the effect this would have on exactness guarantees an index can provide. Using existing document collections, we estimate the space savings that such an approach could achieve. Our results show that a learned index structure approach has the potential to significantly reduce storage requirements, and still provide performance guarantees. The research questions we address are:

RQ1 How might learned indexes be used to support search based on Boolean intersection?
RQ2 Would learned indexes provide any space benefits over current compressed indexing schemes?

2 RELATED WORK

Boolean intersection has been a fundamental component of information retrieval systems for more than fifty years. In fact, early search systems were entirely reliant on Boolean retrieval models [5]. In recent years, ranked retrieval models have become more important, but Boolean operations are still a fundamental component in a variety of search tasks [8, 19].

One important application of Boolean intersection is as either a feature in multi-stage retrieval [12], or as a filtering stage in a multi-stage pipeline [8, 17]. In all of these, the key idea is to apply expensive feature extraction and machine learning models on a subset of the most promising candidate documents to ensure early-precision is maximized in the final result set [3, 21].

Machine learning has been applied in early-stage retrieval to predict the number of documents to pass through to the next stage [6] and even to predict which top-k processing algorithm should be used per query [13]. But in current cascaded and multi-stage retrieval models the use of machine learning algorithms is often deferred to later stages of the retrieval process as traditionally such algorithms were not optimized for efficiency. The recent introduction of learned index structures by Kraska et al. [10] is changing that perception. They have shown that common index structures such as B-tree-indexes, hash-indexes, and Bloom-filters can be replaced by learned models,

ADCS '18, December 11–12, 2018, Dunedin, New Zealand

ACM ISBN 978-1-4503-6549-9/18/12…$15.00
https://doi.org/10.1145/3291992.3291993

Algorithm 1 The Exhaustive Iterative Approach.

```
q_1,...,q_n ← receive_query
r ← []
for d ∈ D do
    if ∀q_i ∈ [q_1,...,q_n],  f(q_i,d) = 1 then
        append(r,d)
return r
```

Algorithm 2 The Two-Tiered Approach.

```
q_1,...,q_n ← receive_query
l_1,...,l_n ← truncated_lists_for_terms(q_1,...,q_n)
L ← ⋃_{i=1}^{n} l_i
r ← []
for d ∈ L do
    if ∀q_i ∈ [q_1,...,q_n],  f(q_i,d) = 1 then
        append(r,d)
return r
```

Algorithm 3 The Block Based Approach.

```
q_1,...,q_n ← receive_query
b_1,...,b_n ← block_lists_for_terms(q_1,...,q_n)
B ← ⋂_{i=1}^{n} b_i
r ← []
for b ∈ B do
    for d ∈ document_range_of_block(b) do
        if ∀q_i ∈ [q_1,...,q_n],  f(q_i,d) = 1 then
            append(r,d)
return r
```

while bringing both gains in memory and computational costs. Moreover, by applying recursive models a learned index structure can fallback on traditional structures for sub-cases where a learned model performs poorly. Consequently, learned models can provide the same correctness guarantees as their traditional counterparts. Given the potential advantages of learned models, we explore this line of research in a constrained early-stage retrieval scenario – specifically, can a learned indexing representation be used for conjunctive Boolean retrieval? If so, what are the performance implications?

3 APPLICABILITY FOR INVERTED INDEXES

In this section we answer **RQ1**: in what ways a learned index model can support Boolean intersection based search.

We will consider models that act as learned Bloom-filters because they have commonly been applied to conjuctive Boolean problems. Moreover, Kraska et al. [10] have shown that they can be applied to sizeable datasets and outperform traditional Bloom-filters in both speed and memory requirements. However, learned index structures have only been optimized for tasks involving a single set [10]. In contrast, each document in an inverted index could be seen as an individual set, thus making the problem substantially more complex.

For this study we will assume that a function $f(t,d) \in \{0,1\}$ can be learned perfectly so that for a term t and document d:

$$f(t,d) = \begin{cases} 1 & t \in d, \\ 0 & t \notin d. \end{cases} \tag{1}$$

In theory any deep neural network that is expressive enough could be optimized for an entire document collection without any errors. In practice, such a model has to be relatively sizeable and requires a very long period of optimization. Therefore, one may choose to compromise some correctness for practical reasons. In this study, we will not discuss the specifics of such a model or its optimization and instead focus on how it could be applied.

3.1 Exhaustive Iterative Approach

A straightforward approach to conjunctive Boolean functions using the model f would be to iterate over the entire document collection. Algorithm 1 displays what this approach could look like. It is clear that, per query, there is a huge computational cost proportional to the number of documents in the collection. However, this approach guarantees the correct results for conjunctive Boolean queries. Moreover, the only storage it requires is for the model f, thus it can provide the biggest gains in memory by completely replacing an inverted index. In practice this approach will most likely be avoided because of its computational costs, yet it provides an interesting example of how the storage requirements could be completely minimized.

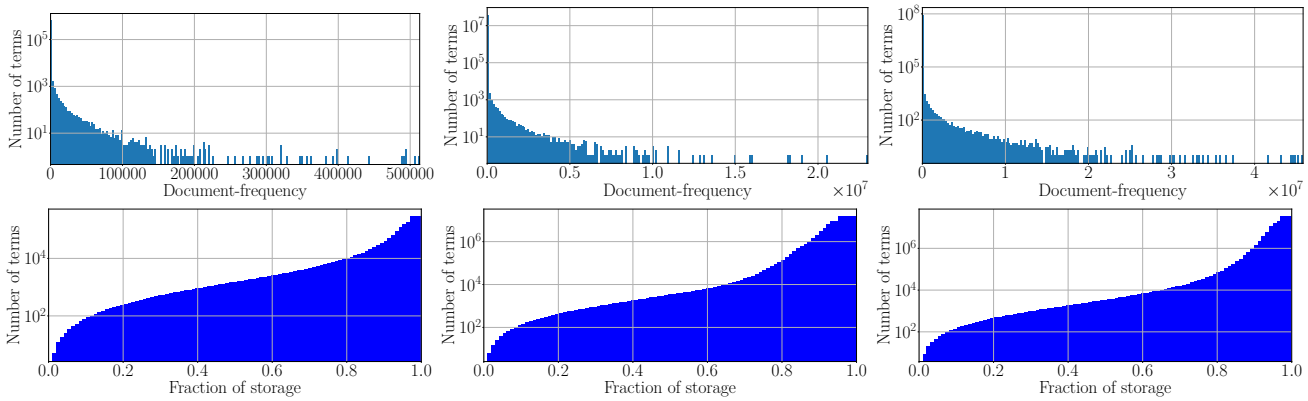
3.2 Two-Tiered Approach

The previous approach iterated over all documents in the collection, which has high computational costs. An existing method of speeding up retrieval is to use two-tier retrieval [18]. Here an index is divided in two partitions, one of which is of smaller size on which queries can be pre-processed quickly. We also propose a two-tiered approach where an inverted index is split into a smaller partition with truncated lists, and a larger partition with the remainder of the lists. We will assume that the size of the second partition is not important, but that the goal is to minimize the size of the first partition. The first partition consists of the inverted lists of each term but truncated to length k, the remainder of each list appears in the second partition. We will not make any assumptions about which parts of the lists are included in the truncations. Then Algorithm 2 displays how one may use the learned model f to search through the first partition. This approach only has to iterate over the intersection of the truncated lists, thus it is computationally more efficient than the previous approach. However, to retrieve all results the truncated lists will not always suffice. If *all* terms in a query have a document frequency greater than k then results may be missing after passing over the first partition. At this point, the algorithm could fallback by also considering the second partition. Conversely, correctness is guaranteed if *at least one* query-term appears in k or less documents. By applying the learned model f there is no need to use the second partition here. Thus for queries with at least one infrequent term the first partition and learned model are guaranteed to provide correct results. This approach may be particularly advantageous when the smaller size allows the first partition to fit in memory components with faster access.

3.3 Block Based Approach

Lastly, we introduce an approach inspired by existing signature files and partitioned approaches [8] in Algorithm 3. A document collection may be partitioned into multiple *blocks*, each containing a subset of documents. For every term, a list indicating the blocks in which their matching documents appear is stored. Then the intersection

Figure 1: Top: the distribution of document frequencies. Bottom: the minimum number of terms that appear at different fractions of the compressed inverted index. From left to right: results for the Robust, GOV2 and ClueWeb collections.



of the lists for every query-term provides restricted ranges in which results for a conjunctive Boolean query appear. Finally, these ranges can be traversed with the learned model f to retrieve all documents for the conjunctive Boolean query. The computational costs of this approach are limited by the size of the partitions. Reductions in storage can be achieved since only a list of partitions has to be stored. We note that for very infrequent terms, traditional inverted lists may still be stored resulting in hybrid presentations [14]. In addition to the storage gains, this approach still guarantees correct results for conjunctive Boolean queries.

Finally, we conclude our answer to **RQ1**: there are several methods by which a learned index structure could be applied to Boolean intersection. These approaches all make different tradeoffs between computational costs during retrieval and gains in the amount of storage space required. It may depend on the requirements of an application which approach is the most suitable.

4 ESTIMATING POTENTIAL GAINS

In the previous section we proposed several approaches to support conjunctive Boolean search with learned index structures. In this section we will answer **RQ2** by estimating the gains these approaches could make in terms of storage requirements.

For this analysis we will consider the two-tiered approach detailed in Section 3.2. This approach was chosen because it appears to produce the least storage gains, thus serving as a conservative bound, and, furthermore, for this approach we can accurately estimate the tradeoffs it makes. Three commonly used TREC document collections are considered for this study: Robust 2005 (Newswire), GOV2, and ClueWeb09B.[1] Figure 1 displays the distribution of document-frequencies in each collection. Additionally, to see the varying storage terms require, it also shows the minimum number of terms that can be stored in different fractions of the compressed inverted index. For this study we used OptPFOR compression [11]. From this figure it is clear that very few terms have a high document-frequency but that they can require a considerable percentage of total storage cost in the inverted index. For instance, in every collection we see that less than one percent of the terms take up forty percent of storage.

To estimate the gains of the two-tiered approach we will use truncated lists of a fixed size k in the first partition. Thus only terms with a higher document-frequency than k will have truncated lists. In addition this affects the optimization of f as it only has to consider terms for which not all documents are stored. The potential gains in storage of the first partition are then estimated as follows: First, we compute the amount of storage gained by removing the inverted lists of replaced terms from the inverted index. Second, we estimate the storage space required by a truncated list of length k; we take the average size of compressed lists of the same length in the complete compressed inverted index [11]. Then we estimate the size of the learned model f as linearly proportional to the vocabulary and collection size: $|T|\cdot|D|\cdot s$, where s is an unknown positive value. Lastly, we expect that for every term a bit has to be stored to indicate whether it has been replaced or not. By summing all these values we get the following formula for the expected gain in storage; with R as the set of terms to replace, the complete set of terms T, and complete set of documents D:

$$gain(R,s) = \left[\sum_{t\in R} size.full.list(t) - size.trunc.list(k)\right] - (|R|+|D|)\cdot s - |T|. \quad (2)$$

To account for the unknown value of s we compute a lower and upper bound by varying its value. For the upper bound, we estimate no cost from the model: $s=0$, this is the most gain this approach could potentially have. The lower bound is estimated with $s=512$ *bits*, this is equivalent to the cost of storing a compressed 128 unit embedding for every document and for every term as well. We expect this to be the worst-case scenario in terms of model size.

Figure 2 displays the estimated bounds for varying truncated list sizes; in addition, it also shows the number of terms that have to be replaced. For instance, on the Robust collection, a gain of at least 40% can be achieved by using a truncated list of 4,000 and replacing less than 4,000 term lists. Interestingly, the number of terms to replace grows exponentially as the truncated list size decreases, while the potential gain increases at a much smaller rate. This further shows that the highest gains can be made by replacing the most frequent terms. Moreover, replacing extremely rare terms could even require more storage depending on the model costs. Regardless, we see that even with high model costs substantial gains are possible by choosing an appropriate truncated list size.

Lastly, on a set of 40,000 queries from the TREC Million Query Track [2], we verified the number of queries with results that can be guaranteed correct on the first partition. *With* a learned model

[1] https://trec.nist.gov

Figure 2: Top: The estimated upper and lower bounds in terms of storage space required. Bottom: The number of terms that need to be replaced. From left to right: results for the Robust, GOV2 and ClueWeb09B collections.

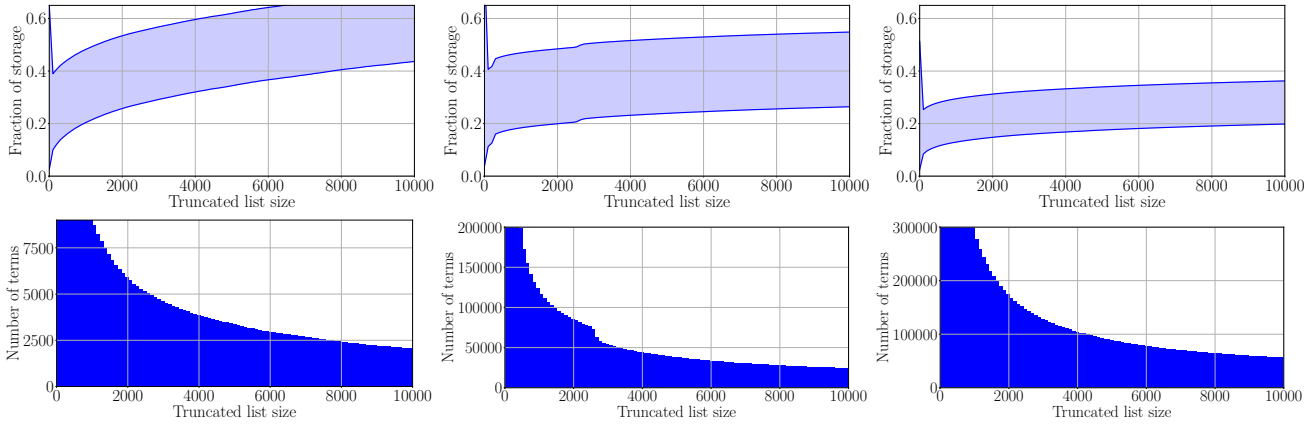
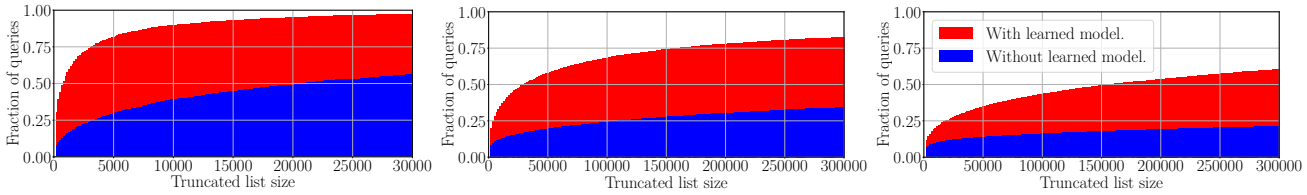


Figure 3: Percentage of queries with guaranteed correct results in the first-tier by varying truncated list sizes. From left to right: Robust, GOV2 and ClueWeb09B.



a query is guaranteed correct if the list of *at least one* term is not truncated. In contrast, *without* a learned model *all* query-terms need complete lists for guaranteed correct results. Figure 3 displays the difference between the two-tiered approach *with* and *without* the learned model. As expected, the learned model considerably increases the correctness of the results in the first stage.

Finally, we answer **RQ2** positively: our results show that even the most storage inefficient approach with high model costs can produce substantial reductions in storage requirements.

5 CONCLUSION

In this study, we have explored how search based on Boolean intersection may benefit from the usage of learned index structures. We have proposed several approaches by which a learned model can produce substantial reductions in storage requirements. Each approach makes a tradeoff between storage requirements and computational costs. Our results show that even conservative estimates on the potential gains w.r.t. space benefits are considerable. We expect that combining learned index structures with inverted indexes will be a fruitful research direction in the near future.

Acknowledgements. This work was partially supported by the Netherlands Organisation for Scientific Research (NWO) under project nr. 612.001.551 and the Australian Research Council's *Discovery Projects* Scheme (DP170102231).

REFERENCES

[1] N. Asadi and J. Lin. 2013. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proc. SIGIR*. 997–1000.
[2] B. Carterette, V. Pavlu, H. Fang, and E. Kanoulas. 2009. Million Query Track 2009 Overview. In *Proc. TREC*.
[3] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. 2017. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proc. SIGIR*. 445–454.
[4] C. L. A. Clarke, J. S. Culpepper, and A. Moffat. 2016. Assessing efficiency–effectiveness tradeoffs in multi-stage retrieval systems without using relevance judgments. *Inf. Retr.* 19, 4 (2016), 351–377.
[5] W. B. Croft, D. Metzler, and T. Strohman. 2010. *Search Engines: Information Retrieval in Practice*. Vol. 283. Addison-Wesley Reading.
[6] J. S. Culpepper, C. L. A. Clarke, and J. Lin. 2016. Dynamic cutoff prediction in multi-stage retrieval systems. In *Proc. ADCS*. 17–24.
[7] J. S. Culpepper and A. Moffat. 2010. Efficient set intersection for inverted indexing. *ACM Trans. Inf. Sys.* 29, 1 (2010), 1.
[8] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He. 2017. BitFunnel: Revisiting signatures for search. In *Proc. SIGIR*. 605–614.
[9] A. Kane and F. Tompa. 2014. Skewed partial bitvectors for list intersection. In *Proc. SIGIR*. 263–272.
[10] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. 2018. The case for learned index structures. In *Proc. SIGMOD*. 489–504.
[11] D. Lemire and L. Boytsov. 2015. Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.* 45, 1 (2015), 1–29.
[12] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. 2013. About learning models with multiple query-dependent features. *ACM Trans. Inf. Sys.* 31, 3 (2013), 11.
[13] J. Mackenzie, J. S. Culpepper, R. Blanco, M. Crane, C. L. A. Clarke, and J. Lin. 2018. Query driven algorithm selection in early stage retrieval. In *Proc. WSDM*. 396–404.
[14] A. Moffat and J.S. Culpepper. 2007. Hybrid bitvector index compression. In *Proc. ADCS*. 25–31.
[15] A. Moffat and J. Zobel. 1996. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Sys.* 14, 4 (1996), 349–379.
[16] G. Ottaviano and R. Venturini. 2014. Partitioned Elias-Fano indexes. In *Proc. SIGIR*. 273–282.
[17] J. Pedersen. 2010. Query understanding at Bing. In *SIGIR Industry Day*.
[18] C. Rossi, E.S. de Moura, A.L. Carvalho, and A.S. da Silva. 2013. Fast document-at-a-time query processing using two-tier indexes. In *Proc. SIGIR*. ACM, 183–192.
[19] T. Russell-Rose and P. Gooch. 2018. 2dSearch: A visual approach to search strategy formulation. In *Proc. DESIRES*. 90–96.
[20] A. Trotman. 2014. Compression, SIMD, and postings lists. In *Proc. ADCS*. 50–57.
[21] L. Wang, J. Lin, and D. Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In *Proc. SIGIR*. 105–114.
[22] J. Zobel and A. Moffat. 2006. Inverted files for text search engines. *ACM Comp. Surv.* 38, 2 (2006), 6.