Document Reordering is Good, Especially for e-Commerce

Vishnusaran Ramaswamy
eBay Search
visramaswamy@ebay.com

Roberto Konow
eBay Search
rkonow@ebay.com

Andrew Trotman
University of Otago
andrew@cs.otago.ac.nz

Jon Degenhardt
eBay Search
jdegenhardt@ebay.com

Nick Whyte
eBay Search
nwhyte@ebay.com

ABSTRACT

Document id reordering is a well-known technique in web search for improving index compressibility and reducing query processing time. We explore and evaluate the benefits of document id reordering for large-scale e-Commerce search. We observe that many e-Commerce sites organize offerings according to an ontology (i.e. product category). We then present a simple extension to document-id reordering: ordering based on item category. Our results show that we not only achieve the expected index size reduction, but also achieve a latency reduction of over 20% (on average) per query.

KEYWORDS

E-Commerce search, Document id Re-ordering, Compression

ACM Reference format:
Vishnusaran Ramaswamy, Roberto Konow, Andrew Trotman, Jon Degenhardt, and Nick Whyte. 2017. Document Reordering is Good, Especially for e-Commerce. In *Proceedings of ACM SIGIR Workshop on eCommerce, Tokyo, Japan, August 2017 (SIGIR 2017 eCom),* 6 pages.

1 INTRODUCTION

The search engine plays an essential role in e-Commerce: it connects the user's need with a set of relevant items based on a query. This is not a simple task, millions of queries per second need to be processed over possibly billions of items. Moreover, it is expected that every query will be executed in just a few hundred milliseconds.

In order to solve this problem, search engines are implemented as large distributed systems where there is a limited budget in CPU and memory that every machine can use. Any improvement in efficiency could potentially be translated into a reduction in hardware, networking, and operating costs. Tremendous research and engineering efforts has gone into addressing performance challenges: reduction of memory requirements by improving data compression, reduction the CPU use by implementing early termination techniques, and massively parallel execution engines are just a few of the techniques that have been extensively studied in the past. In this paper, we focus on one technique originally designed to improve data compression and reduce the size of the data that is loaded into main memory, document id reordering [3].

The *inverted index* is an old and simple, yet very efficient data structure that is at the heart of most search engines and is used to support various search tasks. From a collection of documents, an inverted index stores for each unique term (or word) a list of *postings*. Each posting stores pairs $\langle d, w(t,d)\rangle$, where d is a unique document identifier (doc_id) and $w(t,d)$ is a relevance measure of the term t with respect to document d (often the number of times term t occurs in document d). These *posting lists* can be extended to store additional information such as the positions of the term within the document. Posting lists are usually stored sorted by doc_id and processed document at a time. To help with compression, doc_ids are often stored *difference* encoded - the value stored in the list is the difference (or *d-gap*) between this id and the preceding id. These d-gaps are further compressed using a variable-width integer codec such as variable byte encoding, PForDelta [15], QMX [13], or similar.

The final compression ratio depends on the number of bits required to represent the d-gaps, which depends on how the doc_ids are assigned to documents. If the d-gaps are small they compress better than if they are large. That is, if all the documents containing a given term have document ids that are close to each other then the index is smaller that if they are randomly distributed throughout the collection, simply because the d-gaps are smaller and so compress better.

This has motivated several authors in the past [2, 3, 7, 14] to study the doc_id assignment process in such a way as to optimize compression. In practice, search engines can assign doc_ids in a number of different ways: at random, based on document similarity, in the order documents are indexed (collection order), based on a global measure of quality such as pagerank, or for web documents, URL order. Others have noted [7] that document reordering not only reduces index size, but can also decrease query processing time.

A popular e-Commerce search technique to improve precision is to constrain a query to a particular set of categories in a category tree. This can be done automatically by a trained classifier, or manually by the user. For example, the results of the query *iphone case* can be constrained so that all the resulting items belong to the category "Cell Phones & Accessories Cases, Covers & Skins". These categories also form a natural partitions, clustering items according to popular search dimensions.

In this paper we explore a document id reordering technique for structured and categorized data that both improves compression and decreases query latency. Our results show that document id ordering on category substantially reduces the size of the index. It also reduces mean latency per query by 27% and 99th percentile latency by 45%. Latency improvements are seen both with and without category constraints applied.

2 BASIC CONCEPTS & RELATED WORK

2.1 Inverted Index

Given a collection $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ of text documents, a document d_i can be considered to be a sequence of terms (or words), the number of words, and an unique *document identifier* (doc_id) $\in [1, D]$. The number of words in a document is represented by $|d_i|$, and the total number of words in the collection is then $\sum_{i=1}^{D} |d_i| = n$.

An inverted index maintains the set of distinct terms of the collection (the *vocabulary*), which in most cases is small compared to the total number of words contained in the collection. More precisely, it is of size $O(n^\beta)$, for $0 < \beta < 1$ depending on the text type (β is normally small).

For every distinct word, the inverted index stores a list of *postings*. Each posting stores the *document identifier* (doc_id), the *weight* of the term in the document $w(t, d)$ and, if needed, the positions of the term within the document. The weight $w(t, d)$ of term t in d is a utility function that represents the importance of that word inside the document. That utility function is often just the number of times the term occurs in the document – the term frequency.

There are many different ways to encode posting lists including term-frequency ordered, impact ordered, and so on, but the most common way appears to be ordering on increasing document id. Either way, in order to improve efficiency the inverted index is typically loaded into main memory when the search engine starts up, and parts of it are compressed in order to reduce the memory footprint.

The weights are hard to compress and usually small so they are often stored uncompressed in a fixed number of bytes (often 1). The document ids, however have been the subject of much research.

The list of doc_ids $\langle d_1, d_2, d_3, \ldots d_n \rangle$, is a strictly monotonically increasing sequence. These integers can be decreased in size by calculating the differences between each document id and the preceding document id, resulting in a list of d-gaps $\langle d_1, d_2 - d_1, d_3 - d_2, \ldots, d_n - d_{n-1} \rangle$. The list of d-gaps is then encoded using a variable-length encoding scheme.

Bit-aligned codes were used in the past, but proved to be inefficient when decoding. Byte-aligned codes [11] and word-aligned codes [14] are now preferred as decoding speed is of concern. A simple, yet efficient byte-aligned technique is *Variable Byte Encoding* (VByte), but an alternative approach is to word-align blocks of integers using an encoding such as PForDelta. Integer compression techniques for monotonic integer sequences have been studied for decades. We recommend the reader to see the work of Trotman [13] for a comprehensive study and comparison of techniques.

Merging lists can be done by traversing the lists from the start to finish, but in order to support more complicated query processing techniques random access to postings in a list is needed. A recent approach is postings list encoding using Elias-Fano, but a more common approach is the use of skip-lists.

A skip-list for a postings list is generated by dividing the postings list into *blocks*. Each block starts with a given doc-id and is at a given offset from the start of the postings list. These $\langle$*doc-id, offset*$\rangle$ tuples provide entry points into the postings [6, 10].

To implement random access to a posting, a binary search is performed on the skip list, then the appropriate block is decompressed and searched linearly.

2.2 Query Processing

Query processing involves a number of processes such as query parsing, query rewriting and the computation of complex machine-learned ranking functions that may include hundreds or thousands of features derived from the documents, the query, and the user. To rank efficiently, it is common to separate the query processing into multiple stages. The first stage is responsible for identifying which documents must be ranked and the subsequent stages rank those documents. In the first phase, a simple and fast retrieval filtering such as WAND [4] and BlockMax-WAND [8] are often used. We do not consider these algorithms further as Boolean is common as the first stage in e-Commerce search.

A conjunctive Boolean query of the form "*Last* **AND** *Jedi*" requires an *intersection* calculation, whereas a disjunctive query of the form "*Episode* **OR** *VIII*" requires a *union* calculation. Union queries are solved by linearly traversing all postings lists for all terms in the expression and returning all documents containing either term. Efficiently resolving intersection queries requires complicated traversal of the postings lists and has been examined for decades [1, 5, 9]. It has been proven that the optimal intersection algorithm for two sets of length m and n with $m \leq n$ is $O(m \log \frac{n}{m})$. The most popular algorithm for solving intersections is *Set Versus Set* (and also known as *Small Versus Small*) [1].

In the second and subsequent phases, an increasingly expensive set of ranking functions is used to identify the most relevant documents. Each stage provides to the next stage a better ranking of the recall set, but also reduces the number of documents that are ranked (each time the top-k, for decreasing k, are re-ranked). The final stage might just rank the top-10 using thousands of features derived from the document, the query, meta-data (the price), and a user profile (for e-Commerce, a buyer profile and a seller profile).

In general, improving the efficiency of the first stage frees more resources for the subsequent stages, and thus increases the overall performance of the search engine.

2.3 Document Reordering

Document reordering is a well studied technique in web search [2, 3, 7, 14]. Most prior work, has focused on reordering documents to achieve better compression. The approach is to perform text clustering on the collection to find similar documents and then assign doc_ids to minimize the d-gaps in the posting list. Silvestri [12] explored a much simpler idea that takes advantage of an implicit organization of the documents in the Web. In his work he proposes to assign doc_ids sequentially according to the document URL and showed that this simple technique achieved competitive results when compared to text clustering techniques. In essence, he roughly clustered on topic because different sites and different parts of the same sites are usually topically related.

Yan et al. [14] studied integer encoding for the special case of optimally re-ordered d-gaps and introduced variants of well-known

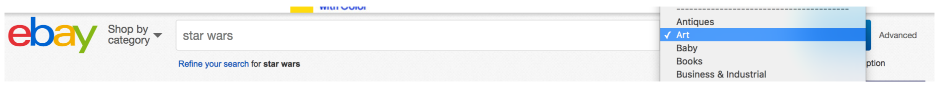

Figure 1: User category constrained query on eBay.com

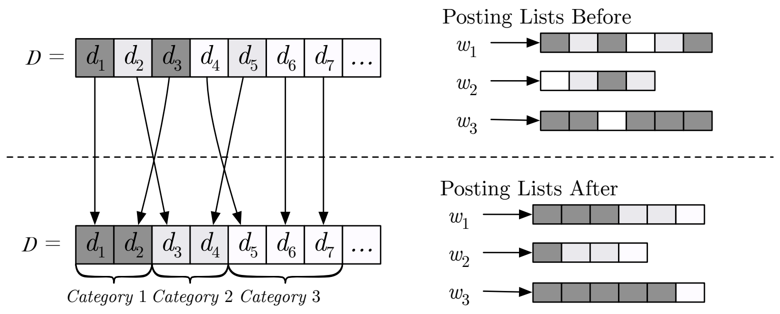

Figure 2: Document reordering diagram. Different gray levels represent different categories. On the left, we show a sketch of how documents get new doc_id assignment. On the right, the effect on posting lists.

encoding techniques for the purpose. They also showed that, when using document reordering, it is possible to effectively compress term frequencies since similar frequency values are also (consequently) grouped together. Finally, they evaluated different query processing schemes and showed that document reordering can help the search engine performance by reducing the number of doc_ids that requires to be decompressed (because fewer encoded blocks are decompressed).

Our approach is motivated by the work of Silvestri [12] – we present a simple but non-optimal document id reordering technique. It takes advantage of the structured nature of documents in e-Commerce, specifically that items that are for sale are usually classified and categorized before being listed. We evaluate this from both compression and query efficiency perspectives. We analyze the benefits of employing document id reordering in our search engine for the different stages in query processing, taking into consideration special constraints that are common in e-Commerce search.

3 DOCUMENT REORDERING IN E-COMMERCE

E-Commerce search is a much more structured and constrained scenario than web search. In e-Commerce, much of the document content is given by the item properties such as price, brand, model, category, color, and so on. It is natural to consider these features as filtering components of a search and it is consequently common practice to generate posting lists for those kind of features. For example, by generating posting lists for each instance of the feature "brand" (i.e brand:apple, brand:samsung, etc) the user can easily constrain their search to just the items made by a given manufacturer – and indeed they expect to be able to do so.

Category is a particularly interesting property of e-Commerce items (and queries), because it is not only used to divide the inventory into distinct types of products but it is also commonly used to improve the precision of the results. Given an user query, the search engine can constrain the results to just those from the most relevant category. This can be done in an automatic way by training query to category classifiers, or by allowing the user to manually select a category. Figure 1 shows an example user query "star wars" being constrained to the category "art" on eBay.com.

If the search engine creates posting lists for each category, the first stage of query processing can be improved significantly, since it can perform a direct boolean intersection between the (possibly term expanded) user query and the posting list for the given category. Since this cuts down the size of the recall base it increases the efficiency of the search engine, but since it restricts to the most likely category it also removes noise from the results list so increases precision. And this is the motivation for document id reordering based on item category.

We re-order the collection so that the doc_ids are assigned in such a way that items belonging to the same category are given contiguous doc_ids. This reduces the size of the d-gaps in posting lists which leads to better compression. However, this is not the only benefit, since posting lists are sorted by doc_id, it creates implicit category "divisions" within each posting list.

Figure 2 illustrates this. On the top left, the collection of documents is shown in "collection order", where the distinct shades of gray represent different categories. The top right gives example posting lists for words (w_1, w_2, w_3). The bottom left of the figure shows the collection category reordered where, for example, collection ordered d_2 becomes category ordered d_3. The bottom right shows the effect of document reordering on the posting lists, they are now implicitly divided by category.

This document reordering scheme not only helps compression, but also decreases latency: as the d-gaps are smaller the decompression of the integers is faster since, in general, a smaller number of operations is required to decode a smaller integer. Equally, since similar documents are stored together, fewer accesses to the skip-lists are needed. It is obvious that when a query is category constrained the results must lie within a consecutive part of the postings list.

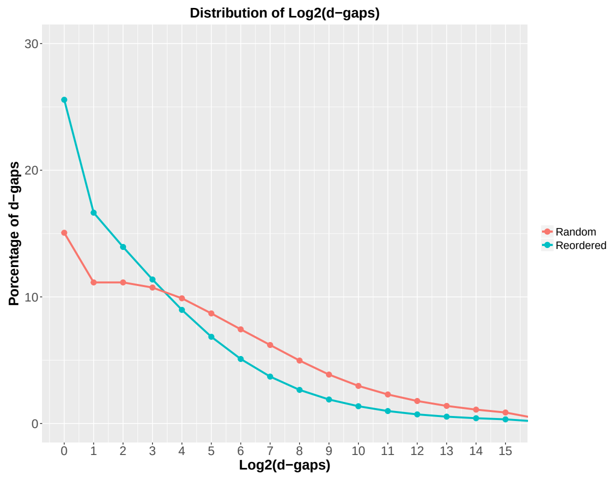

Figure 3: Distribution of d-gaps. The x-axis corresponds to the number of bits required to represent the d-gaps in binary plus one. The y-axis is the percent of such d-gaps in the collection.

4 EXPERIMENTS

4.1 Experimental Setup

We conducted our experiments on eBay's search engine. We selected 12 million random items from our dataset and constructed two indexes:

- Random Index: documents were assigning doc_ids at random.
- Reordered Index: documents were sorted by category and then assigning doc_ids.

All of our experiments were conducted on a dedicated server with two 10-core Intel Xeon E5-2670 v2 CPU running at 2.50Ghz, 25 MB of cache, and 128 GB RAM running Ubuntu 16.04 on Linux Kernel 4.4.0-57. Our search engine is written in C++ and was compiled using g++ 5.4.0 and maximum optimization level.

To evaluate the performance of our document reordering technique we use two sets of queries:

- General Queries: approximately 3 million queries from production logs of one eBay data center. These queries included user issued queries as well as system-issued queries (such as those from the eBay public APIs). No country-specific filtering was performed, so queries are in many languages.
- User Queries: a random sample of 57,058 queries from eBay.com exactly as entered into the search box by ebay.com users.

4.2 Index Space Results

Since compression improvements will depend on the d-gaps values, we first analyzed the differences in the distribution of the d-gaps between random doc_id assignment and category-based doc_id assignment. Figure 3 shows on a log-linear scale the distribution of g-gaps. The x-axis is $\lceil \log_2(gapsize) \rceil$ while the y-axis is the percentage of d-gaps of that size. It can be seen that the reordered distribution has many more d-gaps on the left – the reordered index has substantially more small d-gaps than the random index.

	Random	Reordered	Change (%)
Avg. $\log_2$**(d-gaps)**	5.73	4.11	-28%
d-gaps = 1	890×10^6	$1,510 \times 10^6$	+70%
Avg. d-gaps	1966	639	-67.5%
Avg. Bytes/d-gap (vByte)	1.30	1.22	-6.1%
Index Size	29.67 GB	28.72 GB	-3.2%

Table 1: Space savings and bits per d-gap obtained before and after applying document reordering.

Table 1 presents a summary of the figure. It shows that the number of d-gaps equal to 1 has increased by 70%, that the average d-gap has decreased by 67.5%, and that the average number of bits required to represent d-gaps is reduced by 28%. In practice, the actual size reduction in the index will depend on the integer encoding scheme. For the purpose of this paper, we constructed a special index that uses variable byte encoding to compress doc_ids. We see a reduction of 6.1% in the average number of bytes required to represent a doc_id using this encoding scheme, while this represents a 3.2% space reduction of the complete index.

4.3 Query Processing Results

In order to evaluate the improvement of query processing, we executed two sets of experiments with the two different query sets.

The first experiment considered throughput using the General Queries – that is, its a mirror of the production environment. We computed the average number of queries per second (QPS) that could be resolved when the CPU was held at 80% utilization (the other 20% is used in eBay for processes such as index maintenance). We found that on average the Reordered Index could process about about 30% more queries per second than the Random Index.

Table 2 shows the average search time per query (in milliseconds) at the mean, median, 95th percentile and 99th percentile. In all cases we see a substantial improvement, the mean latency improved by 26.9% while 95th percentile latency is almost halved when compared to the random document Reordering.

	Random	Reordered	Latency Reduction
Mean	22.43	16.4	26.9%
Median	4.35	3.21	28.3%
95th Pct.	57	30.2	47%
99th Pct.	375	224	40%

Table 2: Comparison of random doc_id assignment versus category doc_id reordering. Mean, median, 95th and 99th percentiles of query processing times in milliseconds for general queries.

For the second experiment, with user queries, we evaluated the impact of document reordering depending on the recall set size (as output by the first stage of the search process) and the latency

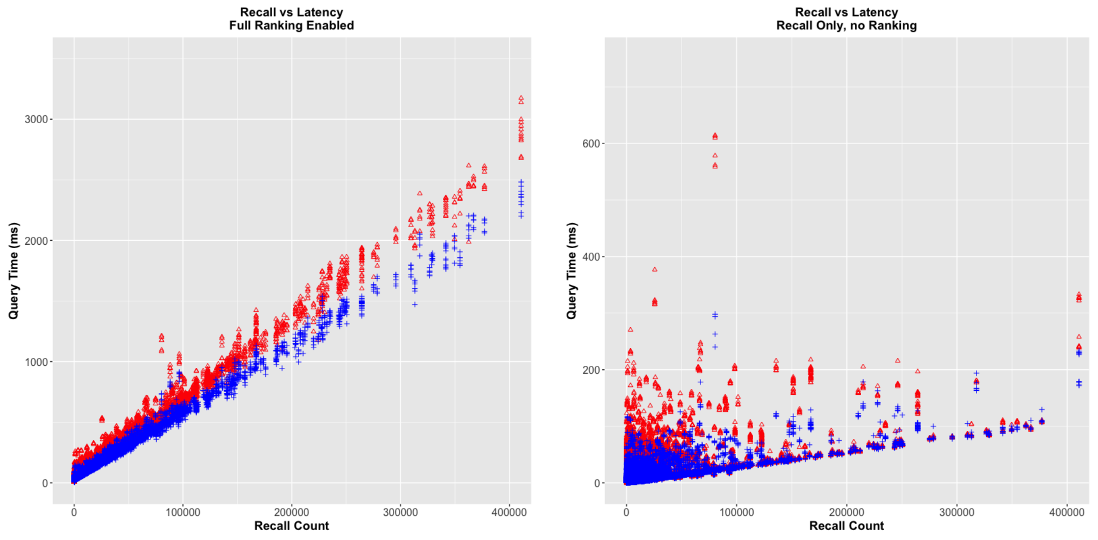

Figure 4: Recall size vs latency with full ranking enabled and without ranking. Blue triangles represent data points with document reordered index, while red plus signs represents random document reordering.

	Without Category Constraint			With Category Constraint		
	Random	**Reordered**	**Latency Reduction**	**Random**	**Reordered**	**Latency Reduction**
Mean	26.8	14.3	47%	18.9	8.4	55%
Median	5.9	3.5	42%	3.2	1.7	48%
95th Pct.	85.8	50.8	41%	61.6	27.6	55%

Table 3: Execution time in milliseconds for user queries with and without category constraint enforcement.

before ranking and after ranking. The results are presented in figure 4. The blue plus signs represent the Reordered Index, while the red triangles represent the Random Index. On the left we show the impact post ranking, where it can be seen that the largest latency improvements are obtained when the queries generate a large set of results. On the right we show the recall versus latency results when ranking is disabled, in other words, just recall identification. It can be seen that there is a strict boundary at the bottom, and there is no significant improvement for the queries that are located within that boundary. Latency improvements can be seen overall, but are large for expensive queries.

We also evaluated the impact of applying category constrains to the queries. The results are shown in table 3. The left side shows the latency (in milliseconds) when category constraint is not used. In this case the Reordered index improved mean query latency by 47% and the 95th percentile by 41%. The right shows the effect of enabling category constrain on the queries. There the mean query latency has reduced by 55% when the Reordered Index is used, and a similar effect is observed for the 95th percentile. Clearly both unconstrained and category constrained queries are materially improved.

4.4 Latency Improvement Analysis

Categories naturally cluster both document terms and query terms. Items satisfying a multi-term AND query will tend to come from a small number of categories. Ordering posting lists by category will put documents satisfying these queries near each other both in posting lists and in the forward index. This should improve CPU cache hit rates and even improve the inherent efficiency of the posting list processing algorithms. The latter effect would result from having larger clusters of posting list entries that are either matched or skipped than in a random assignment of the documents.

Query expansion is likely to compound these effects, especially in an e-Commerce environment. As in most search engines, we make extensive use of query expansion to improve recall and precision. Rewritten queries often form a complex Boolean expression involving many posting lists and nested AND / OR constructs. Expansions involve not only text keywords, but also the structured meta-data associated with products. For example, the term “black” may expand to “color:black” and “samsung” to “brand:samsung”.

To examine these effects we used the CPU usage profiling tool PERF, while processing a portion of a General Queries collection, to analyze and identify the exact locations where the improvement was more noticeable. We observed the functions performing the task of iterating (and skipping) through posting lists was consuming about 5% of the total CPU time, and we observed a noticeable improvement especially in these parts. We also saw improvements in the doc_id variable byte decoding section. Finally we analyzed the effect of how cache misses were affected by the document reordered index. We observed a 7% reduction in overall cache misses and a

13% reduction in last-level cache misses (LLC). These numbers show that that document ordering by category yielded a significant improvement in overall CPU cache hit rates. These numbers are consistent with our hypothesis for the improvements in latency. Additional analysis is still needed to quantify the effects on posting listing processing.

The probability that any given cluster of posting list entries will be referenced by a query is far from uniform in the General Queries collection, and more likely following something like a zipfian curve. This should reduce the number of CPU cache entries filled with posting list data during processing of a query, and thus reducing the CPU cache worked set size for the portion of query processing dedicated to processing posting lists. The reduction in CPU cache working set size for posting lists allows a larger amount of CPU cache to be used for other functions performing query processing work, which improves the CPU cache hit rate for those other functions.

The above discussion focuses on the recall component of query processing. As mentioned earlier, document reordering also better co-locates forward index data for items satisfying the recall expression for a query. Forward index data is used extensively in the ranking component of query processing. As such, this is also has potential to improve CPU cache hit rates. We have not measured this effect directly, but it is consistent with the results shown in figure 4. These graphs show a latency improvement from ranking beyond the improvement from recall processing alone.

5 CONCLUSIONS AND FUTURE WORK

We presented a simple, yet efficient, document reordering technique that takes advantage of structured component from the queries and the data. This is particularly common in e-Commerce search engines. We showed that by performing this simple re-arrangement of the doc_ids we can improve the capacity of the system by 30%, and process category-constrained queries in about half the time that it took without the re-arrangement.

As future work, we plan to add more document reordering criteria to other dimensions such as item aspects like color, brand and also the location of the item by ordering by country, another important task is to measure the behavior of the cache in this scenario. We also plan to perform a deeper analysis and quantification of the latency improvements seen in both boolean only queries and when ranking is enabled. A topic for future study is whether there is a material difference between these two cases when considering per-document latency benefits. The current analysis is insufficient to answer this question.

REFERENCES

[1] Jeremy Barbay, Alejandro López-Ortiz, Tyler Lu, and Alejandro Salinger. 2009. An experimental investigation of set intersection algorithms for text searching. *ACM Journal of Experimental Algorithmics* 14 (2009), art. 7.

[2] Roi Blanco and Álvaro Barreiro. 2005. Document identifier reassignment through dimensionality reduction. In *European Conference on Information Retrieval*. Springer, 375–387.

[3] Dan Blandford and Guy Blelloch. 2002. Index compression through document reordering. In *Proceedings of the Data Compression Conference. DCC 2002.* IEEE, 342–351.

[4] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on Information and knowledge management.* ACM, 426–434.

[5] Shane Culpepper and Alistair Moffat. 2010. Efficient Set Intersection for Inverted Indexing. *ACM Transactions on Information Systems* 29, 1, Article 1 (2010), 25 pages.

[6] Shane J. Culpepper and Alistair Moffat. 2007. Compact Set Representation for Information Retrieval. In *Proceedings of the 14th International Symposium on String Processing and Information Retrieval (SPIRE).* 137–148.

[7] Shuai Ding, Josh Attenberg, and Torsten Suel. 2010. Scalable Techniques for Document Identifier Assignment in Inverted Indexes. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10).* ACM, 311–320.

[8] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval.* ACM, 993–1002.

[9] Andrew Kane and Frank Tompa. 2014. Skewed Partial Bitvectors for List Intersection. In *Proceedings of the 37th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR) (SIGIR '14).* 263–272.

[10] Peter Sanders and Frederik Transier. 2007. Intersection in Integer Inverted Indices. In *Proceedings 9th Workshop on Algorithm Engineering and Experiments (ALENEX).*

[11] Falk Scholer, Hugh Williams, John Yiannis, and Justin Zobel. Compression of inverted indexes for fast query evaluation. In *Proceedings of the 25th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR).* 222–229.

[12] Fabrizio Silvestri. 2007. Sorting out the Document Identifier Assignment Problem. In *Proceedings of the 29th European Conference on IR Research (ECIR'07).* 101–112.

[13] Andrew Trotman. 2014. Compression, SIMD, and Postings Lists. In *Proceedings of the 2014 Australasian Document Computing Symposium (ADCS).* Article 50, 8 pages.

[14] Hao Yan, Shuai Ding, and Torsten Suel. 2009. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World wide web.* ACM, 401–410.

[15] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. 2006. Super-scalar RAM-CPU cache compression. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE'06.* IEEE, 59–59.