From MaxScore to Block-Max WAND: The Story of How Lucene Significantly Improved Query Evaluation Performance

Adrien Grand,[1] Robert Muir,[2] Jim Ferenczi,[1] and Jimmy Lin[3]

[1] Elastic NV [2] Ntrepid Corporation
[3] David R. Cheriton School of Computer Science, University of Waterloo

This is the preprint of an accepted ECIR 2020 reproducibility paper.

Abstract. The latest major release of Lucene (version 8) in March 2019 incorporates block-max indexes and exploits the block-max variant of WAND for query evaluation, which are innovations that originated from academia. This paper shares the story of how this came to be, which provides an interesting case study at the intersection of reproducibility and academic research achieving impact in the "real world". We offer additional thoughts on the often idiosyncratic processes by which academic research makes its way into deployed solutions.

Keywords: open-source software · technology adoption

1 Introduction

We share the story of how an innovation that originated from academia—block-max indexes and the corresponding block-max WAND query evaluation algorithm of Ding and Suel [6]—made its way into the open-source Lucene search library. This represents not only a case study in widespread reproducibility, since every recent deployment of Lucene has access to these features and thus their performance benefits can be easily measured, but also of academic research achieving significant impact. How did these innovations make their way from the "ivory tower" into the "real world"? We recount the sequence of events, including false starts, that finally led to the inclusion of block-max WAND in the latest major version of Lucene (version 8), released in March 2019.

We see this paper as having two main contributions beyond providing a narrative of events: First, we report results of experiments that attempt to match the original conditions of Ding and Suel [6] and present additional results on a number of standard academic IR test collections. These experiments characterize the performance of Lucene's implementation and show the extent to which performance improvements are retained when moving from a research prototype to a production codebase. Second, we offer a number of observations about the adoption of academic innovations, perhaps providing some insight into how academics might achieve greater real-world impact with their work.

2 Setting the Stage

From its very beginnings in 1999, Lucene has mostly existed in a "parallel universe" from academic IR researchers. Part of this can be attributed to its "target audience": developers who wish to build real-world search applications, as opposed to researchers who wish to write papers. Academic IR researchers have a long history of building and sharing search engines, dating back to the mid 1980s with Cornell's SMART system [4]. The tradition continues to this day, with Lemur/Indri [12, 13] and Terrier [14, 8] being the most successful examples of open-source academic search engines, still popular with many researchers today. Until recently, there has been little exchange between Lucene and these systems, other than a few academic workshops [21, 16].

Lucene has, for the longest time, been somewhat of a laughingstock in the academic IR community. For much of its existence, its default ranking model was a variant of TF-IDF that was not only *ad hoc*, but demonstrably less effective than ranking models that were widely available in academic systems [18]. Okapi BM25 was not added to Lucene until 2011,[1] more than a decade after it gained widespread adoption in the research community; the consensus had long emerged that it was more effective than TF-IDF variants. This lag has contributed to the broad perception by researchers that Lucene produces poor search results and is ill-suited for information retrieval research.

This negative perception of Lucene, however, began to change a few years ago. In 2015, an evaluation exercise known as the "open-source reproducibility challenge" [7] benchmarked seven open-source search engines and demonstrated that Lucene was quite competitive in terms of both effectiveness and efficiency. It was the fourth fastest system (of seven) in terms of query evaluation, beating all the systems that were better than it in terms of effectiveness.

Since then, there has been a resurgence of interest in adopting Lucene for information retrieval research, including a number of workshops that brought together like-minded researchers over the past few years [2, 1]. Anserini [19, 20] is an open-source toolkit built on Lucene that was specifically designed to support replicable information retrieval research by providing many research-oriented features missing from Lucene, such as out-of-the-box support for a variety of common test collections. The project aims to better align IR researchers and practitioners, as Lucene has become the de facto platform used in industry to build production search solutions (typically via systems such as Elasticsearch and Solr). The experiments in this paper were conducted with Anserini.

3 From MaxScore to Block-Max WAND

At Berlin Buzzwords in 2012, Stefan Pohl gave a presentation about MaxScore [17] to raise awareness about efficient retrieval techniques in the Lucene community [15]. The presentation was accompanied by a working prototype.[2]

[1] `https://issues.apache.org/jira/browse/LUCENE-2959`

[2] `https://issues.apache.org/jira/browse/LUCENE-4100`

This contribution was exciting but also challenging to integrate as it conflicted with some of the flexibility that Lucene provides, requiring an index rewrite. There were ideas on how to address these issues, but they entailed a lot of effort, and so the issue remained stalled for about five years.

Five years is a long time and many changes occurred meanwhile. The switch from TF-IDF to BM25 as Lucene's default scoring function in 2015 created a natural upper bound on scores due to BM25's saturation effect, which made it possible to implement retrieval algorithms that reasoned about maximum scores without changes to Lucene's index format. This led to an effort to implement a general-purpose Wand [3], based on a previous implementation for `BooleanQuery`. Lucene received support for Wand at the end of 2017 (although it wasn't released until version 8.0 with block-max indexes).

Implementing Wand introduced two new issues. First, the total hit count would no longer be accurate, since not all matches are visited. Common analytics use cases depend on this count, and many search engines display this value in their interfaces (see additional discussion in Section 5). Second, the fact that some Lucene queries could produce negative scores became problematic, so Lucene now requires positive scores.[3]

Support for block-max indexes was the final feature that was implemented, based on the developers' reading of the paper by Ding and Suel [6], which required invasive changes to Lucene's index format. Note that the paper describes directly storing the maximum impact score per block, which fixes the scoring function at indexing time. To provide flexibility in being able to swap in different scoring functions, the Lucene implementation stores all tf (term frequency) and dl (document length) pairs that might yield the maximum score. If we have one such pair $(\mathrm{tf}_i, \mathrm{dl}_i)$ then we can remove all other pairs $(\mathrm{tf}_j, \mathrm{dl}_j)$ where $\mathrm{tf}_j \leq \mathrm{tf}_i \wedge \mathrm{dl}_j \geq \mathrm{dl}_l$, since they are guaranteed to yield lower (or equal) scores—based on the assumption that scores increase monotonically with increasing tf and decreasing dl. This is implemented by accumulating all such pairs in a tree-like structure during the indexing process. These pairs are stored in skip lists, so the information is available to groups of 8, 64, 512, 4096, ... blocks, allowing query evaluation to skip over more than one block at a time.

An interesting coda to this story is that academic researchers were exploring alternatives to per-block impact scores circa 2017, for exactly the same reason (to allow the scoring model to be defined at search time). For example, Macdonald and Tonellotto [10] showed how to derive tight approximate upper bounds for block-max Wand, based on work that dates back to 2011 [9]. Similarly, the recently-released PISA research system stores flexible block-level metadata [11]. Unfortunately, the Lucene developers were not aware of these developments during their implementation.

The journey from MaxScore to block-max Wand concluded in March 2019, with the rollout of all these features in the version 8.0 release of Lucene. They are now the out-of-the-box defaults in the world's most popular search library.

[3] `https://issues.apache.org/jira/browse/LUCENE-7996`

Table 1. Per-query latency (ms), comparing Ding and Suel [6] with Lucene under similar experimental conditions, but on different hardware ($k = 10$).

	TREC 2005	TREC 2006
Ding and Suel [6]: exhaustive OR	369	226
Ding and Suel [6]: WAND	64	78
Ding and Suel [6]: BMW	21	28
Lucene: exhaustive OR	98	188
Lucene: BMW	32	55

4 Experimental Evaluation

During the implementation of block-max WAND, performance improvements were quantified in terms of Lucene's internal benchmark suite, which showed a 3× to 7× improvement in query evaluation performance. As part of a formal reproducibility effort, we present experiments that attempt to match, to the extent practical, the original conditions described by Ding and Suel [6].

According to the paper, experiments were conducted on the Gov2 web collection, on a randomly-selected subset of 1000 queries from the TREC 2005 and 2006 Efficiency Tracks, which we were able to obtain from the authors. For their experiments, the inverted index was completely loaded into main memory and query evaluation latency was measured to retrieval depth ten.

Our experiments were conducted with the Anserini IR toolkit,[4] comparing v0.5.1, which depends on Lucene 7.6 and uses an optimized exhaustive OR query evaluation strategy [5] with v0.6.0, which depends on Lucene 8.0 and uses block-max WAND. We used Anserini's standard regression test settings on the different collections, as described on its homepage. Results represent averages over three trials on a warm cache. While the indexes were not explicitly loaded into memory, Lucene benefits from caching at the OS level.

All experiments were conducted using a single thread on an otherwise idle server with dual Intel Xeon E5-2699 v4 processors and 1TB RAM running RHEL (release 7.7). Results are shown in Table 1, where figures in the top three rows are copied from Table 1 in the original paper. It is interesting that Ding and Suel report a much larger increase in performance comparing exhaustive OR to BMW (18× on TREC 2005 and 8× on TREC 2006) than the comparable conditions in Lucene (a more modest improvement of around 3×). This is due to a more optimized implementation of exhaustive OR in Lucene, which, for example, implements block processing [5]. Interestingly, Ding and Suel report faster query evaluation in absolute terms, even on hardware that is much older: among the differences include C++ vs. Java, as well as the simplicity of a research prototype vs. the realities of a fully-featured search library. Beyond implementation differences, Lucene must additionally compute the upper bound scores per block from the stored (tf, dl) pairs on the fly.

[4] `http://anserini.io/`

Table 2. Per-query latency (ms) for different queries, collections, and retrieval depths.

Collection	ClueWeb09b			ClueWeb12-B13		
Retrieval depth k	10	100	1000	10	100	1000
TREC 2005 queries						
Lucene: exhaustive Or	331	371	521	321	371	669
Lucene: Bmw	96	137	370	94	148	489
Speedup	3.4×	2.7×	1.4×	3.4×	2.5×	1.4×
TREC 2006 queries						
Lucene: exhaustive Or	424	464	659	404	442	693
Lucene: Bmw	123	191	431	127	186	526
Speedup	3.4×	2.4×	1.5×	3.2×	2.4×	1.3×

Table 3. Indexing time in seconds.

Collection	Lucene 7.6	Lucene 8.0	
Gov2	2528	2719	+7.6%
ClueWeb09b	6333	6817	+7.6%
ClueWeb12-B13	7514	7943	+5.7%

We also report performance evaluations on two other standard test collections frequently used in academic information retrieval: ClueWeb09b and ClueWeb12-B13, with the same sets of queries. These results are shown in Table 2, where we report figures for different values of retrieval depth k, also averaged over three trials. These numbers are consistent with Figure 7 in Ding and Suel's paper: performance of exhaustive Or drops modestly as depth k increases, but Bmw performance degrades much more quickly. This is exactly as expected.

Finally, we quantify the modest increase in indexing time due to the need to maintain (tf, dl) pairs in the inverted indexes, shown in Table 3 (averaged over three trials, using 44 threads in all cases). These experiments used Anserini's default regression settings on the respective collections, which builds full positional indexes and also stores the raw documents.

5 Discussion

The story of block-max Wand in Lucene provides a case study of how an innovation that originated in academia made its way into the world's most widely-used search library and achieved significant impact in the "real world" through hundreds of production deployments worldwide (if we consider the broader Lucene ecosystem, which includes systems such as Elasticsearch and Solr). As there are very few such successful case studies (the other prominent one being the incorporation of BM25 in Lucene), it is difficult to generalize these narratives into "lessons learned". However, here we attempt to offer a few observations about how academic research might achieve greater real-world impact.

In short, block-max WAND is in Lucene because the developers learned about Ding and Suel and decided to reimplement it. This is somewhat stating the obvious, but this fateful decision highlights the idiosyncratic nature of technology adoption. We could easily imagine alternative scenarios where the Lucene developers had not come across the paper and developed a comparable solution in isolation, or they might have come across the paper and instead elected to take a different approach. In either case, the Lucene solution would likely differ from block-max WAND in some respects. This would be akin to convergent evolution in evolutionary biology, whereby different organisms independently evolve similar traits because they occupy similar environments. In such an "alternate reality", this paper would be comparing and contrasting different solutions to handling score outliers, not describing a reproducibility effort. Could a lesson here be that researchers need to better "evangelize" their innovations?

Eight years passed from the publication of the original paper (2011) until the release of Lucene that included block-max WAND (2019). The entire course of innovation was actually much longer if we trace the origins back to MaxScore (1995) and WAND (2003). One obvious question is: Why did it take so long?

There are many explanations, the most salient of which is the difference between a research prototype and a fully-featured search library that was already widely deployed. This decomposes into two related issues, the technical and the social. From a technical perspective, supporting BMW required invasive changes to Lucene's index format and a host of related changes in scoring functions—for example, scores could no longer be negative, and implementations could no longer access arbitrary fields (which was an API change). These had to be staged incrementally. Concomitant with technical changes and backwards-compatibility constraints were a host of "social" changes, which required changing users' expectations about the behavior of the software. In short, BMW was *not* simply a drop-in replacement. For example, as discussed in Section 3, the hit count was no longer accurate, which required workarounds for applications that depended on the value. Because such major changes can be somewhat painful, they need to be justified by the potential benefits. This means that only dramatic improvements really have any hope of adoption: multiple-fold, not marginal, performance gains. An interesting side effect is that entire generations of techniques might be skipped, in the case of Lucene, directly from exhaustive OR to BMW, leapfrogging intermediate innovations such as MAXSCORE and WAND.

6 Conclusions

Aiming to achieve real-world impact with academic research is a worthwhile goal, and we believe that the case study in this paper represents an endorsement of efforts to better align research prototypes with production systems, as exemplified by Lucene-based projects like Anserini. If academic researchers are able to look ahead "down the road" to see how their innovations might benefit end applications, the path from the "ivory tower" to the "real world" might become more smoothly paved.

7 Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. We'd like to thank Craig Macdonald and Antonio Mallia for helpful discussions on the intricacies of computing flexible per-block score bounds, and Torsten Suel for providing us with the original queries used in their evaluations.

References

1. Azzopardi, L., Crane, M., Fang, H., Ingersoll, G., Lin, J., Moshfeghi, Y., Scells, H., Yang, P., Zuccon, G.: The Lucene for Information Access and Retrieval Research (LIARR) Workshop at SIGIR 2017. In: Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017). pp. 1429–1430. Tokyo, Japan (2017)
2. Azzopardi, L., Moshfeghi, Y., Halvey, M., Alkhawaldeh, R.S., Balog, K., Di Buccio, E., Ceccarelli, D., Fernández-Luna, J.M., Hull, C., Mannix, J., Palchowdhury, S.: Lucene4IR: Developing information retrieval evaluation resources using Lucene. SIGIR Forum **50**(2), 58–75 (2017)
3. Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A., Zien, J.: Efficient query evaluation using a two-level retrieval process. In: Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM 2003). pp. 426–434. New Orleans, Louisiana (2003)
4. Buckley, C.: Implementation of the SMART information retrieval system. Department of Computer Science TR 85-686, Cornell University (1985)
5. Cutting, D.R., Pedersen, J.O.: Space optimizations for total ranking. In: Computer-Assisted Information Searching on Internet (RIAO '97). pp. 401–412. Paris, France (1997)
6. Ding, S., Suel, T.: Faster top-k document retrieval using block-max indexes. In: Proceedings of the 34rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011). pp. 993–1002. Beijing, China (2011)
7. Lin, J., Crane, M., Trotman, A., Callan, J., Chattopadhyaya, I., Foley, J., Ingersoll, G., Macdonald, C., Vigna, S.: Toward reproducible baselines: The open-source IR reproducibility challenge. In: Proceedings of the 38th European Conference on Information Retrieval (ECIR 2016). pp. 408–420. Padua, Italy (2016)
8. Macdonald, C., McCreadie, R., Santos, R.L., Ounis, I.: From puppy to maturity: Experiences in developing Terrier. In: Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval. pp. 60–63. Portland, Oregon (2012)
9. Macdonald, C., Ounis, I., Tonellotto, N.: Upper-bound approximations for dynamic pruning. ACM Transactions on Information Systems **29**(4), 17:1–17:28 (2011)
10. Macdonald, C., Tonellotto, N.: Upper bound approximation for BlockMaxWand. In: Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR '17). pp. 273–276 (2017)
11. Mallia, A., Siedlaczek, M., Mackenzie, J., Suel, T.: PISA: Performant Indexes and Search for Academia. In: Proceedings of the Open-Source IR Replicability Challenge (OSIRRC 2019): CEUR Workshop Proceedings Vol-2409. pp. 50–56. Paris, France (2019)

12. Metzler, D., Croft, W.B.: Combining the language model and inference network approaches to retrieval. Information Processing and Management **40**(5), 735–750 (2004)
13. Metzler, D., Strohman, T., Turtle, H., Croft, W.B.: Indri at TREC 2004: Terabyte track. In: Proceedings of the Thirteenth Text REtrieval Conference (TREC 2004). Gaithersburg, Maryland (2004)
14. Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Lioma, C.: Terrier: A high performance and scalable information retrieval platform. In: Proceedings of the SIGIR 2006 Workshop on Open Source Information Retrieval. pp. 18–25 (2006)
15. Pohl, S.: Efficient scoring in Lucene. In: Berlin Buzzwords 2012 (2012)
16. Trotman, A., Clarke, C.L., Ounis, I., Culpepper, S., Cartright, M.A., Geva, S.: Open source information retrieval: A report on the SIGIR 2012 workshop. SIGIR Forum **46**(2), 95–101 (2012)
17. Turtle, H., Flood, J.: Query evaluation: Strategies and optimizations. Information Processing and Management **31**(6), 831–850 (1995)
18. Turtle, H., Hegde, Y., Rowe, S.A.: Yet another comparison of Lucene and Indri performance. In: Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval. pp. 64–67. Portland, Oregon (2012)
19. Yang, P., Fang, H., Lin, J.: Anserini: Enabling the use of Lucene for information retrieval research. In: Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017). pp. 1253–1256. Tokyo, Japan (2017)
20. Yang, P., Fang, H., Lin, J.: Anserini: Reproducible ranking baselines using Lucene. Journal of Data and Information Quality **10**(4), Article 16 (2018)
21. Yee, W.G., Beigbeder, M., Buntine, W.: SIGIR06 workshop report: Open source information retrieval systems (OSIR06). SIGIR Forum **40**(2), 61–65 (2006)