Profiling and Visualizing Dynamic Pruning Algorithms

Zhixuan Li
zhixuan.li1@uq.net.au
The University of Queensland
Brisbane, Australia

Joel Mackenzie
joel.mackenzie@uq.edu.au
The University of Queensland
Brisbane, Australia

ABSTRACT

Efficiently retrieving the top-k documents for a given query is a fundamental operation in many search applications. Dynamic pruning algorithms accelerate top-k retrieval over inverted indexes by skipping documents that are not able to enter the current set of results. However, the performance of these algorithms depends on a number of variables such as the ranking function, the order of documents within the index, and the number of documents to be retrieved. In this paper, we propose a diagnostic framework, Dyno, for profiling and visualizing the performance of dynamic pruning algorithms. Our framework captures *processing traces* during retrieval, allowing the operations of the index traversal algorithm to be visualized. These visualizations support both query-level and system-to-system comparisons, enabling performance characteristics to be readily understood for different systems. Dyno benefits both academics and practitioners by furthering our understanding of the behavior of dynamic pruning algorithms, allowing better design choices to be made during experimentation and deployment.

CCS CONCEPTS

• **Information systems → Information retrieval query processing**; **Retrieval efficiency**; • **Human-centered computing → Visualization systems and tools**.

KEYWORDS

Query processing, visualization, profiling, explainability

ACM Reference Format:
Zhixuan Li and Joel Mackenzie. 2023. Profiling and Visualizing Dynamic Pruning Algorithms. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3539618.3591806

1 INTRODUCTION

Inverted indexes facilitate scalable query processing over massive text collections, and can be used to directly answer queries, or to generate input to more expensive and complex re-rankers [49]. Although different index arrangements are possible, the most common one is the *document-ordered* index, where a *postings list* records pairs of document identifiers and payloads for each term [52]. These postings lists are sorted increasing on the document identifier, and payloads can be term frequencies, or pre-computed impacts.

A fundamental operation on inverted indexes is top-k retrieval, where the highest scoring k documents are returned according to a chosen ranking function. Since postings lists can, in the worst case, store an entry for every document in the corpus, naïve index traversal algorithms that examine every posting are prohibitively expensive. To accelerate the retrieval process, *dynamic pruning* algorithms such as WAND [6] or MaxScore [48] can be employed. These algorithms typically rely on the highest impact observed across each postings list, known as a score upper-bound, to be stored during indexing. At query time, a min-heap maintains the highest scoring k documents observed. Each candidate document score is then estimated via the list-wise upper-bounds before committing to the computation of the true score (which could result in expensive decompression and scoring operations). If this estimation is lower than the lowest scoring element in the min-heap (known as the *heap threshold*, θ), then the document can be safely bypassed; it has no prospect of entering the top-k results. Further optimizations can be made by storing a per-block upper-bound, allowing for better score estimations at the cost of a larger index [7, 15, 32].

While dynamic pruning algorithms are widely accepted as "must-have" optimizations for fast top-k retrieval [16], there are a number of subtleties that can impact their performance. Petri et al. [38], and later Khattab et al. [19], showed that the choice of ranker alone could drastically alter the performance characteristics of different traversal algorithms. This was also observed for neural *learned sparse* ranking models [27, 28, 31, 36]. Mallia et al. [34] explored how different compression codecs [40] and retrieval settings affect performance. They found that the fastest algorithm depended on both the value of k and the length of the given query. Similarly, Mackenzie and Moffat [25] found that algorithm-independent optimizations such as estimating the initial heap threshold [18, 35, 39, 50], reordering the index [4, 13, 26, 43, 44], pre-computing and quantizing the impact scores [2, 8], and applying stopword lists to queries can have *additive* efficiency benefits. We refer the interested reader to the survey of Tonellotto et al. [47] for a more detailed discussion on efficient query processing.

Given the complex trade-off space of available algorithms and optimizations, choosing the right configuration, as well as understanding and diagnosing performance shortcomings, can be difficult in practice. In this work, we propose a diagnostic framework called Dyno[1] for profiling, visualizing, and comparing the performance of dynamic pruning algorithms. Our contributions are as follows:

(1) We propose a web-based visualization framework for rapidly comparing and contrasting the performance of top-k retrieval algorithms that facilitates analysis on a *whole-of-experiment* and on a *query-by-query* basis;

[1]Named after a *dynamometer*, a tool used to measure force, torque, or power.

(2) We describe the profiling mechanism used to instrument the PISA search toolkit [33] to collect the appropriate data; and
(3) We run a proof-of-concept analysis campaign on the commonly used `MSMARCO-v1` passage collection to demonstrate the usage and utility of our framework.

2 DYNO: SYSTEM DESCRIPTION

This section describes the technical details of Dyno, including how algorithmic profiling is implemented within the PISA system [33] to generate traces that characterize the index traversal, and how Dyno processes and visualizes those traces.

2.1 Overview

Dyno is built as a *dash* app with *plotly* supporting the data visualization.[2] It is implemented entirely within Python, and can be executed either locally or deployed as a web app. Dyno ingests simple space-separated data files and processes them to generate a *global* dashboard view (allowing comparisons across an entire log of queries) and *per-query traces* (for diagnosing algorithmic performance).

2.2 Generating Profiling Data

In this work, we instrument the highly efficient PISA system [33] to generate the appropriate input data, noting that any other IR system supporting dynamic pruning (such as Anserini [51] or Terrier [23]) could be instrumented in the same way. To generate the required data, a new `profile_viz` tool was built within PISA. This tool, like the other standard PISA tools, accepts a series of configuration arguments such as the index, the query file, and so on. The tool then executes two individual runs of a given configuration; the first run generates latency data (taking the mean per-query latency over three individual runs), with no profiling or instrumentation included to avoid biasing the measurement; and secondly, a run which instruments the processing, incrementing counters for each operation of interest (such as the number of postings scored). This tool then generates the appropriate output for post-processing, including the query information, global statistics, and per-query traces:

- **Query Information:** For each query, the query tokens, postings list lengths, and list-wise upper-bounds are recorded.
- **Global Statistics:** For each query, the query latency, number of documents scored, number of postings scored, and number of decompression operations are recorded. Since PISA compresses document identifiers separately to payloads, a counter is maintained for each.
- **Per-query Traces:** For each document scored, the current state of processing is captured. This includes the current document identifier, the heap threshold, and the score achieved by the current document before it was either added to, or rejected from, the heap.

Both the *query information* and *global statistics* are collected on a "one per query" basis, and are thus typically small. On the other hand, the *traces* collect a data point for each document scored, and can hence result in tens or hundreds of thousands of points per query. Furthermore, a trace must be generated for each combination of settings. To reduce storage costs, traces are stored as gzip compressed files, one for each unique query, and are decoded on-the-fly when required by Dyno.

2.3 Visualization

Once profiling is complete and the output files are placed in the corresponding data directory, Dyno can be invoked to start the visualization dashboard. There are three main areas of interest within the Dyno dashboard.

Firstly, the *global comparison* page facilitates the comparison of algorithms across the whole log of queries, allowing the experimenter to gain an overall understanding of how algorithms compare as settings are varied. Figure 1 demonstrates this page. Note that there are four key metrics of interest: the overall latency, in microseconds; the number of documents that were considered viable candidates during retrieval; the number of postings scored during retrieval; and the number of compressed document and frequency blocks decoded. These metrics, together, provide a detailed view of algorithm performance, and can direct further "drill downs" on the data for query-level tracing.

Secondly, the *query tracing* page provides a detailed snapshot of the work done during index traversal for a specific algorithm configuration and query. Figure 2 shows the query tracing interface. Of particular interest is the statistics panel and the query trace visualization, which provide information for explaining the performance of the current configuration and query. This visualization is inspired by the static figures from Petri et al. [38].

Thirdly, the *per-query statistics* page tabulates all available metrics and configurations with interactive filtering and sorting capability.

3 DEMONSTRATION

In this section, we demonstrate how Dyno can be deployed to facilitate comparisons between systems, and for diagnosing performance issues. Our demo compares a number of top-k retrieval algorithms under varying settings on the 2019 TREC Deep Learning track data.

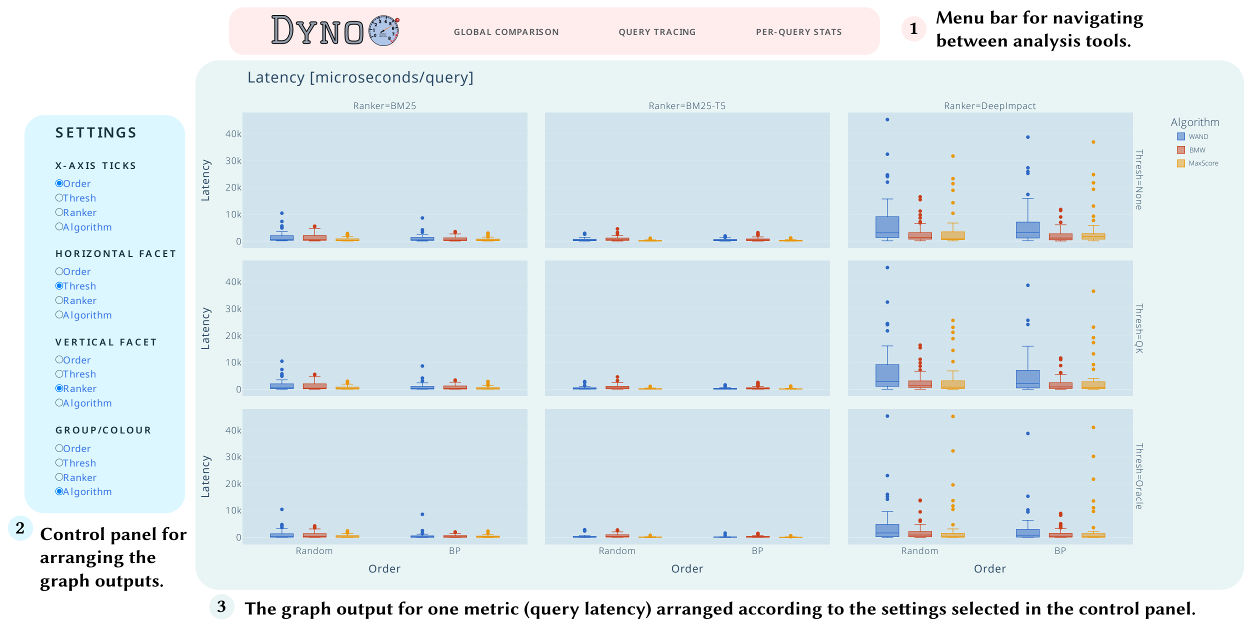
3.1 Data and Queries

We employ the `MSMARCO-v1` passage collection, consisting of around 8.8 million English passages taken from Bing web pages [3]. The 2019 TREC Deep Learning track provided a total of 200 queries, 43 of which were judged for the passage ranking task [10]; we use the latter as our input log.

3.2 Algorithms and Parameters

To generate a set of representative systems to compare, we apply a combination of choices. In particular, we use three algorithms, three rankers, two index orderings, and three heap threshold estimators, resulting in 36 unique systems. We fix $k = 10$ and pre-quantize all scores during indexing (such that ranking becomes a simple sum over impacts) to simplify the experimentation and to avoid a combinatorial explosion of configurations. We also fixed the index compression codec to SIMD-BP128 [21].

[2]See: https://dash.plotly.com/, accessed January 17, 2023.



1 Menu bar for navigating between analysis tools.

SETTINGS

X-AXIS TICKS
[✓ Order] Thresh Ranker Algorithm

HORIZONTAL FACET
Order [✓ Thresh] Ranker Algorithm

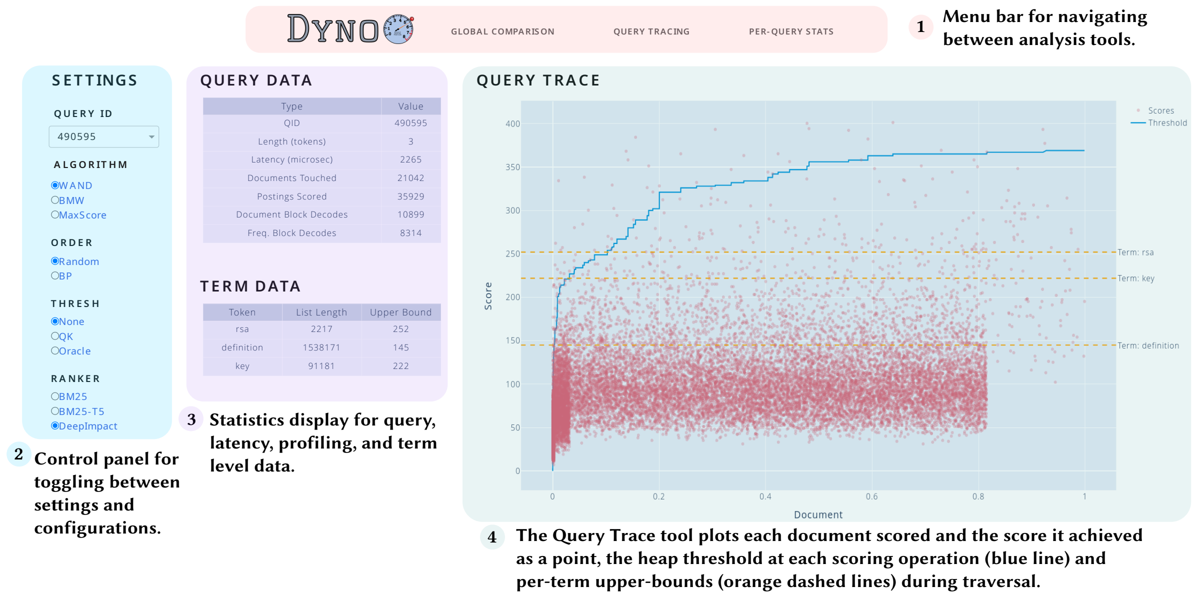
VERTICAL FACET
Order Thresh [✓ Ranker] Algorithm

GROUP/COLOUR
Order Thresh Ranker [✓ Algorithm]

2 Control panel for arranging the graph outputs.

3 The graph output for one metric (query latency) arranged according to the settings selected in the control panel.

Figure 1: The global comparison page showing the latency profiles of all possible system configurations. Each component of the page is labeled with a short explanation. Note that the global comparison page also plots a number of other metrics such as *postings scored*.



1 Menu bar for navigating between analysis tools.

SETTINGS

QUERY ID
490595

ALGORITHM
[✓ WAND] BMW MaxScore

ORDER
[✓ Random] BP

THRESH
[✓ None] QK Oracle

RANKER
BM25 BM25-T5 [✓ DeepImpact]

2 Control panel for toggling between settings and configurations.

QUERY DATA

Type	Value
QID	490595
Length (tokens)	3
Latency (microsec)	2265
Documents Touched	21042
Postings Scored	35929
Document Block Decodes	10899
Freq. Block Decodes	8314

TERM DATA

Token	List Length	Upper Bound
rsa	2217	252
definition	1538171	145
key	91181	222

3 Statistics display for query, latency, profiling, and term level data.

4 The Query Trace tool plots each document scored and the score it achieved as a point, the heap threshold at each scoring operation (blue line) and per-term upper-bounds (orange dashed lines) during traversal.

Figure 2: The Dyno query tracing display showing a visualized trace of the query *rsa key definition* using WAND processing, a randomly ordered index, no threshold estimation, and DeepImpact scoring. Again, each component of the page is labeled with a short explanation.

Algorithms. We make use of three popular rank-safe dynamic pruning algorithms. MaxScore is a family of dynamic pruning algorithms proposed by Turtle and Flood [48] that make use of list-wise upper-bound estimations and partial scoring to accelerate query processing; we use the *document-at-a-time* version. We also consider the WAND approach of Broder et al. [6] as well as the *block-max* version (BMW) proposed by Ding and Suel [15]. WAND, similar to MaxScore, uses list-wise bounds for pruning, whereas BMW also relies on block-wise upper-bounds for more accurate score estimations at the cost of a larger index.

Rankers. Since dynamic pruning algorithms are sensitive to term score distributions [28, 38], we apply three rankers. BM25 represents the classical BM25 scoring function [41] over the original corpus with PISA's BM25 function configured with $k_1 = 0.82$ and $b = 0.68$. BM25-T5[3] applies the same formulation and parameters as above, but over a corpus augmented with queries generated via the DocT5Query model [22, 37]. DeepImpact represents the neural term impact estimation model from Mallia et al. [36] which estimates per-term impacts over a DocT5Query expanded corpus.

Index Orderings. The order in which document identifiers are assigned can have a large impact on both the compressibility and, in turn, the efficiency of inverted indexes [4, 26, 43–45]. For simplicity, two index orderings were used. A random ordering simply permutes the document identifiers of all passages, and indexes them in the new order. BP ordering represents the current state-of-the-art approach to document reordering which directly optimizes the estimated cost of storage across all postings lists [13, 29].

Threshold Estimation. As described in Section 1, dynamic pruning algorithms use cheap score estimations to decide whether a candidate document is worth a "closer look" — if the estimated score exceeds θ, the current threshold of the min-heap, then the document will be scored. The idea of threshold estimation is to get the value of θ as close to its terminal value as possible (but not over it), allowing more documents to be pruned during the traversal [35, 39]. Here, we apply three possibilities. First, we do not apply prediction, and allow the value of θ to grow organically. Second, we use the maximum of the k th highest per-term scores, denoted Q_k, which can be computed offline for a fixed set of k values [12, 18]. Third, we use an *oracle* which initializes θ to its terminal value, representing the "best case" scenario for a threshold estimator. We note that while "unsafe" optimizations or estimators are also possible [9, 46], we do not consider them here.

3.3 Global Comparisons

The first application of Dyno compares systems using a series of configurable box-and-whisker plots, one for each metric of interest. The plots can be arranged such that the x-axis, horizontal and vertical facets, and the color of the boxes facilitates comparisons between different dimensions of interest. Figure 1 demonstrates the global comparison interface, plotting query latency according to the settings shown in the left panel. In this particular instance, it is quite clear that the ranker (vertical facet) has the highest net effect on latency; BP indexes are typically faster than their random counterparts; and both MaxScore and BMW are more competitive than WAND on this particular collection. Transforming the plot allows different comparisons to be made effectively. For example, to compare the effect of the different threshold estimators, they could be moved to the group/color option, with the algorithms then occupying the horizontal facet.

3.4 Query-Level Analysis

To demonstrate query-level analysis, Figure 2 shows the processing trace for query 490595 *"rsa key definition"* under a given configuration. From the statistics panel, it is clear that this particular configuration yields a latency of around 2,200 microseconds, with around 36,000 postings scored. The visual trace demonstrates that, after an initial "dense" period of scoring (over the first 5% of the document space), the heap threshold exceeds the first single-term bound (corresponding to the term *"definition"* with an upper-bound of 145) which allows the algorithm to better discern between candidates worth scoring, and those that can be safely ignored. A similar effect can be seen at around 82%, where the combination of the terms *"definition"* and *"key"* (with a combined bound of $145 + 222 = 367$) is no longer sufficient to cause a document to be fully scored; after this point, a document must contain the term *"rsa"* to be considered a viable candidate, and hence a much lower density of documents are scored after this point.

Toggling the configuration then leads to a number of insights. For example, moving from WAND processing to MaxScore leads to a 4.1× speedup under this particular configuration, demonstrating the sensitivity of the choice of algorithm. Even more interesting, however, is the fact that MaxScore processes 81,000 postings — more than two times the amount that WAND processes — yet is over four times faster. This can be explained by examining the other performance metrics; WAND actually requires more than 4× and 7× the amount of document and frequency blocks to be decoded, respectively. This indicates that the volume of postings scored may not be a good proxy for latency [24].

4 CONCLUSION

In this paper we presented Dyno, a framework for profiling and visualizing inverted index-based dynamic pruning algorithms. Dyno is designed to simplify performance analysis and system design for both researchers and practitioners. While we believe that Dyno is an interesting proof of concept, it has a number of limitations. For example, Dyno is best viewed on a large, high resolution monitor, and while it is unlikely to be used on mobile devices, more flexible support for different window sizes would improve the user experience. Furthermore, Dyno currently ingests data collected during dedicated profiling runs from the PISA system; it would be useful to extend it to a live monitoring scenario, where interactive queries are being served with real-time profiling. However, it is unclear whether the computational overhead of profiling is prohibitively expensive for live deployments.

In future work, we plan to extend the data model to more well-supported standard such as JSON to facilitate interoperability with other IR toolkits. It would be interesting to apply our framework to the analysis of more advanced processing regimes, such as those which employ distributed or cluster-based querying [1, 17, 20, 30], multiple index tiers [11], or more accurate upper-bound filtering and estimation processes [5, 14, 19, 42]. Finally, it would also be valuable to include information about search effectiveness, as this would facilitate more robust analysis of unsafe algorithms which can trade effectiveness for efficiency improvements.

Demo. The live system and video are available at https://dyno.uqcloud.net/ and https://youtu.be/mQyFOtGvsyw.

Acknowledgment. This work was supported by a University of Queensland New Staff Research Grant. We thank Maxime Cordeil, Shengyao Zhuang, Guido Zuccon, and the anonymous reviewers for their helpful suggestions.

[3]While this is not exactly a different ranker, the expansions result in modified term frequencies, and hence alters the behavior of retrieval algorithms.

REFERENCES

[1] I. S. Altingovde, E. Demir, F. Can, and O. Ulusoy. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Trans. Inf. Sys.*, 26(3), 2008.

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

[3] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. MS MARCO: A human generated MAchine Reading COmprehension dataset. *arXiv:1611.09268v3*, 2018.

[4] D. Blandford and G. Blelloch. Index compression through document reordering. In *Proc. DCC*, pages 342–352, 2002.

[5] E. Bortnikov, D. Carmel, and G. Golan-Gueta. Top-k query processing with conditional skips. In *Proc. WWW*, pages 653–661, 2017.

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

[7] K. Chakrabarti, S. Chaudhuri, and V. Ganti. Interval-based pruning for top-k processing over compressed lists. In *Proc. ICDE*, pages 709–720, 2011.

[8] M. Crane, A. Trotman, and R. O'Keefe. Maintaining discriminatory power in quantized indexes. In *Proc. CIKM*, pages 1221–1224, 2013.

[9] 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. WSDM*, pages 201–210, 2017.

[10] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, and E. M. Voorhees. Overview of the TREC 2019 deep learning track. In *Proc. TREC*, 2021.

[11] C. M. Daoud, E. S. de Moura, D. Fernandes, A. S. da Silva, C. Rossi, and A. Carvalho. Waves: A fast multi-tier top-k query processing algorithm. *Inf. Retr.*, 20(3):292–316, 2017.

[12] L. L. S. de Carvalho, E. S. de Moura, C. M. Daoud, and A. S. da Silva. Heuristics to improve the BMW method and its variants. *Journal of Information & Data Management*, 6(3):178–191, 2015.

[13] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *Proc. KDD*, pages 1535–1544, 2016.

[14] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. A candidate filtering mechanism for fast top-k query processing on modern CPUs. In *Proc. SIGIR*, pages 723–732, 2013.

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

[16] A. Grand, R. Muir, J. Ferenczi, and J. Lin. From MaxScore to Block-Max Wand: The story of how Lucene significantly improved query evaluation performance. In *Proc. ECIR*, pages 20–27, 2020.

[17] F. Hafizoglu, E. C. Kucukoglu, and I. S. Altingovde. On the efficiency of selective search. In *Proc. ECIR*, pages 705–712, 2017.

[18] A. Kane and F. W. Tompa. Split-lists and initial thresholds for WAND-based search. In *Proc. SIGIR*, pages 877–880, 2018.

[19] O. Khattab, M. Hammoud, and T. Elsayed. Finding the best of both worlds: Faster and more robust top-k document retrieval. In *Proc. SIGIR*, pages 1031–1040, 2020.

[20] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. Does selective search benefit from WAND optimization? In *Proc. ECIR*, pages 145–158, 2016.

[21] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.*, 45(1):1–29, 2015.

[22] X. Ma, R. Pradeep, R. Nogueira, and J. Lin. Document expansions and learned sparse lexical representations for MSMARCO V1 and V2. In *Proc. SIGIR*, 2022.

[23] C. Macdonald, R. McCreadie, R. L. T. Santos, and I. Ounis. From puppy to maturity: Experiences in developing Terrier. In *Proc. OSIR at SIGIR 2012*, 2012.

[24] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proc. SIGIR*, pages 621–630, 2012.

[25] J. Mackenzie and A. Moffat. Examining the additivity of top-k query processing innovations. In *Proc. CIKM*, pages 1085–1094, 2020.

[26] J. Mackenzie, A. Mallia, M. Petri, J. S. Culpepper, and T. Suel. Compressing inverted indexes with recursive graph bisection: A reproducibility study. In *Proc. ECIR*, pages 339–352, 2019.

[27] J. Mackenzie, Z. Dai, L. Gallagher, and J. Callan. Efficiency implications of term weighting for passage retrieval. In *Proc. SIGIR*, pages 1821–1824, 2020.

[28] J. Mackenzie, A. Mallia, A. Moffat, and M. Petri. Accelerating learned sparse indexes via term impact decomposition. In *Findings of the ACL: EMNLP 2022*, pages 2830–2842, 2022.

[29] J. Mackenzie, M. Petri, and A. Moffat. Tradeoff options for bipartite graph partitioning. *IEEE Trans. Know. & Data Eng.*, 2022. To appear.

[30] J. Mackenzie, M. Petri, and A. Moffat. Anytime ranking on document-ordered indexes. *ACM Trans. Inf. Sys.*, 40(1):13.1–13.32, 2022.

[31] J. Mackenzie, A. Trotman, and J. Lin. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Trans. Inf. Sys.*, 41(4), 2023.

[32] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster BlockMax WAND with variable-sized blocks. In *Proc. SIGIR*, pages 625–634, 2017.

[33] A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. PISA: Performant indexes and search for academia. In *Proc. OSIRRC at SIGIR 2019*, pages 50–56, 2019.

[34] A. Mallia, M. Siedlaczek, and T. Suel. An experimental study of index compression and DAAT query processing methods. In *Proc. ECIR*, pages 353–368, 2019.

[35] A. Mallia, M. Siedlaczek, M. Sun, and T. Suel. A comparison of top-k threshold estimation techniques for disjunctive query processing. In *Proc. CIKM*, pages 2141–2144, 2020.

[36] A. Mallia, O. Khattab, N. Tonellotto, and T. Suel. Learning passage impacts for inverted indexes. In *Proc. SIGIR*, pages 1723–1727, 2021.

[37] R. Nogueira and J. Lin. From doc2query to docTTTTTquery, 2019. Unpublished technical report.

[38] M. Petri, J. S. Culpepper, and A. Moffat. Exploring the magic of WAND. In *Proc. Aust. Doc. Comp. Symp.*, pages 58–65, 2013.

[39] M. Petri, A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. Accelerated query processing via similarity score prediction. In *Proc. SIGIR*, pages 485–494, 2019.

[40] G. E. Pibiri and R. Venturini. Techniques for inverted index compression. *ACM Comp. Surv.*, 53(6):125.1–125.36, 2021.

[41] S. E. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trnd. Inf. Retr.*, 3:333–389, 2009.

[42] D. Shan, S. Ding, J. He, H. Yan, and X. Li. Optimized top-k processing with global page scores on block-max indexes. In *Proc. WSDM*, pages 423–432, 2012.

[43] W.-Y. Shieh, T.-F. Chen, J. J.-J. Shann, and C.-P. Chung. Inverted file compression through document identifier reassignment. *Inf. Proc. & Man.*, 39(1):117–131, 2003.

[44] F. Silvestri. Sorting out the document identifier assignment problem. In *Proc. ECIR*, pages 101–112, 2007.

[45] N. Tonellotto, C. Macdonald, and I. Ounis. Effect of different docid orderings on dynamic pruning retrieval strategies. In *Proc. SIGIR*, pages 1179–1180, 2011.

[46] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. In *Proc. WSDM*, pages 63–72, 2013.

[47] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient query processing for scalable web search. *Found. Trnd. Inf. Retr.*, 12(4-5):319–500, 2018.

[48] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Inf. Proc. & Man.*, 31(6):831–850, 1995.

[49] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. SIGIR*, pages 105–114, 2011.

[50] E. Yafay and I. S. Altingovde. Caching scores for faster query processing with dynamic pruning in search engines. In *Proc. CIKM*, pages 2457–2460, 2019.

[51] P. Yang, H. Fang, and J. Lin. Anserini: Reproducible ranking baselines using lucene. *J. Data Inf. Qual.*, 10(4):16.1–17.20, 2018.

[52] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comp. Surv.*, 38(2):6.1–6.56, 2006.