Approximate Bag-of-Words Top-k Corpus Graphs

Lachlan Dunn[1], Luke Gallagher[2], and Joel Mackenzie[1]

[1] The University of Queensland, Brisbane, Australia
[2] The University of Melbourne, Melbourne, Australia

Abstract. A recent line of work has investigated the use of *corpus graphs* to improve the latency-vs-effectiveness envelope of information retrieval systems. The key idea is to build a document-to-document similarity graph offline, allowing additional relevance signals to be exploited during query processing. However, these graphs are inherently expensive to build, requiring a quadratic "all-pairs similarity" computation. In this work, we examine the problem of building corpus graphs using bag-of-words models, and explore heuristics to build high quality graphs at a fraction of the total cost of exhaustive algorithms. We demonstrate that simple mechanisms such as document titles, expanded surrogate queries, and high impact terms can yield effective graphs at a fraction of the cost of their exhaustive counterparts.

Keywords: Corpus graph construction · Top-k query processing.

1 Introduction and Background

Document retrieval systems are typically composed of multiple processing phases — cheap candidate generation, followed by subsequent rounds of more expensive re-ranking — such that utility can be maximized subject to strict resource budgets [4]. Retrieval systems that use a bag-of-words retrieval phase and then a neural re-ranking phase (such as a cross-encoder or bi-encoder) [11, 13, 22, 24] are subject to the vocabulary mismatch problem [21, 25]. That is, documents that are returned by the candidate selection mechanism must contain a subset of the user's query terms, thus ignoring semantic similarity and potentially limiting the effect of downstream re-ranking models.

Recently, MacAvaney et al. [15] introduced the notion of the *corpus graph*, a d-regular graph where document neighbors are reflective of document-to-document similarity; in essence, each document maintains an adjacency list of d neighbors that are deemed to be *most similar* to the document itself, explicitly clustering documents within the corpus. To build a corpus graph, an offline query-by-document method [30] is computed via *all-pairs similarity* [3], with the top-d neighbors from each document retained.

During the online processing phase, candidate documents can be expanded with their d nearest neighbors to populate the candidate pool with additional and potentially relevant documents that were not included in the initial first-stage of retrieval. Prior work has documented the online performance of corpus

graphs with both exhaustive and approximate similarity scoring mechanisms on both dense (TCT-HNP [14]) and sparse (BM25 [27]) paradigms that yield a competitive trade-off between query latency and effectiveness on MSMARCO-v1 [10, 15]. Subsequent work on dense corpus graphs has applied them to improve the performance of lexical retrievers [9], and to document re-ranking in the context of graph neural networks [7].

For a corpus containing $|D|$ documents, graph construction is asymptotically quadratic, $\mathcal{O}(|D|^2)$, due to the requirement of scoring each document against all other documents in the corpus. This motivated us to explore whether simple and cheap heuristics can be used to build corpus graphs that remain effective. We provide a comprehensive view on approximate corpus graphs based on simple bag-of-words retrieval models, and examine their influence toward trade-off decisions central to the ongoing online efficiency-effectiveness debate [10]. Our main contribution is on accelerating sparse top-k similarity scoring as an approximate method for graph construction. We show that rudimentary query-by-document representations can be readily processed by existing dynamic pruning methods in lieu of the full query document. In particular, we demonstrate reductions in sparse corpus graph construction costs of up to 138× over an exhaustive baseline with negligible quality degradation on MSMARCO-v1. We also affirm that typical online algorithms that make use of corpus graphs are robust to noise [10].

2 Cost Effective Corpus Graph Construction

Construction of corpus graphs can be framed as a *query-by-document* problem which is, in turn, a classic retrieval problem with long queries. However, it is well known that efficient *dynamic pruning* algorithms — those used for inverted index-based top-k retrieval — tend to slow down as query length increases [20, 28]. Hence, we primarily focus on *query reduction* heuristics [2], with the aim of reducing the total number of terms used during processing:

- Title+URL: One approach we employ is to obtain a succinct representation of the full document term vector by the concatenation of the document title and URL tokens, if available.[3]
- TF-IDF: Another simple heuristic from classic term ranking is to use TF-IDF weights (aboutness and eliteness) [26] by ranking the terms in each indexed document by their TF-IDF scores. We retain the top-n terms to represent each document, using both $n = 5$ and $n = 10$.
- DTQ: Our final query reduction heuristic relies on the assumption that the index being used to build the corpus graph has been augmented with queries (known as *document expansion* [23, 25]). In this case, it seems obvious to use these expansion queries as surrogates for the document in the query-by-document process. In our work, we assume a DocT5Query index is available, and use the concatenation of the first q expanded queries as the document representation, with $q = 1$ and $q = 5$.

[3] The first sentence is used in the absence of a title or URL — 43 times in total.

Instead of limiting the length of the input document used in the query-by-document process, we also designed an algorithm to reduce the number of similarity scores that would be computed. The idea is to issue a full document as a query to retrieve the k most similar documents, R; then, for a k' element prefix of R, each document is scored against *only* the k documents in R rather than against the whole of D. This process is repeated until each document has been processed. The intuition is that the documents in R should be similar to each other, allowing useful neighbors to still be found. We denote this algorithm as LimitPairs and set $k = 1{,}000$ and $k' = 64$, respectively.

3 Experimental Setup

Before outlining our experiments, we briefly describe the experimental settings used. Our experimental resources are available to facilitate reproducibility.[4]

3.1 Document Collections and Queries

Our experiments are conducted on the `MSMARCO-v1` [1] passage collection, consisting of 8.8M passages. We used the 2019 and 2020 TREC Deep Learning track queries [5].

3.2 System Configuration

The PISA search system [19] was used for both graph building (offline) and first stage retrieval (online) experiments. Top-k retrieval was performed using BM25 [27] with $k_1 = 0.82$, $b = 0.68$ and the dynamic pruning algorithm MaxScore [29], which has been shown to outperform alternatives for long queries or when k is large [20]. Two indexes were configured. The first index is built over the unmodified (or original) passage collection, referred to as Original. The second index used the passage collection augmented with queries via document expansion, referred to as DocT5Query. Indexes were compressed with SIMD-BP128 [12] following reordering with recursive graph bisection [6, 18] following best practice [17]. Second-stage re-ranking was conducted using the *lexically accelerated dense retrieval* (LADR) software made available by Kulkarni et al. [10], which is integrated into PyTerrier [16].[5]

3.3 Performance Measurement

Effectiveness evaluation was conducted using `trec_eval`. We primarily focus on recall, as we believe that LADR is more suited for deployment as an early-stage component of a multi-stage retrieval system. Following Kulkarni et al. [10], all end-to-end latency is accounted for — including the first stage inverted-index traversal — with the exception of encoding queries into their dense representations for re-scoring.

[4] github.com/lgrz/approx-bow-corpusgraph
[5] github.com/georgetown-ir-lab/ladr

3.4 Hardware

All experiments were conducted in-memory on a Ubuntu server with two Intel Xeon Gold 6144 CPUs and 512 GiB of RAM. Graph construction experiments made use of 28 cores; all online experiments used a single CPU core.

3.5 Online Algorithms

While there are a number of available algorithms that use corpus graphs including GAR, LADR (adaptive and proactive), and LexBoost, we opt to use the *proactive* version of LADR as a representative algorithm due to its simplicity. In essence, the proactive LADR algorithm takes an initial first-stage set of *seed documents*, denoted R_0, and re-scores each document in R_0 — as well as their corpus graph neighbors — using a dense retriever. BM25 is used to generate seed sets of varying sizes ($|R_0| = \{5, 10, 20, 50, 100, 200, 500, 1000\}$) to obtain different efficiency-vs-effectiveness points over each graph produced. Dense re-ranking uses 768 dimensional `TAS-B` passage embeddings [8].[6]

4 Experiments and Analysis

The results from our experiments in approximate bag-of-words graph construction are discussed. First we consider the offline component for approximate corpus graph construction followed by our evaluation of the trade-offs in efficiency and effectiveness during document retrieval. Then we conclude with a discussion on the similarity between the different approximate graphs and how this may relate to the proactive LADR retrieval mechanism.

4.1 Offline Graph Construction

Our first experiment measures the total latency required to process the input document collection in order to build a corpus graph. We experimented with both the Original index as well as a DocT5Query expanded index, measuring two graph settings: A small graph with $d = 16$; and a large graph with $d = 128$. Table 1 shows the results. It is evident that our heuristics can significantly reduce the total resource expenditure of the graph building process. In particular, the TF-IDF method can reduce the time required to construct the large graph from 18 hours to a mere 8 *minutes* on the expanded index. Other heuristics also perform well, with the lowest observed speedup around 6× using the LimitPairs algorithm.

4.2 End-to-End Retrieval

While the heuristics show rather impressive improvements over the default corpus graph construction cost, it is not yet clear that these graphs are effective;

[6] huggingface.co/sebastian-hofstaetter/distilbert-dot-tas_b-b256-msmarco

Table 1: Time in minutes to construct two corpus graphs denoted by d on `MSMARCO-v1` for the two indexes Original, DocT5Query. The relative improvement with respect to the Exhaustive query-by-document baseline is also shown.

	Name	Description	Avg. $\lvert Q\rvert$	$d = 16$		$d = 128$	
				Time	Imp.	Time	Imp.
Original	Exhaustive	Full doc text	30.1	448	–	564	–
	Title+URL	Title and URL only	7.7	28	16×	37	15×
	TF-IDF-5	Top 5 weighted terms	5.0	5	90×	7	81×
	TF-IDF-10	Top 10 weighted terms	10.0	24	19×	31	18×
	LimitPairs	Limit doc pairs scored	30.1	75	6×	75	8×
DocT5Query	Exhaustive	Full doc text	51.1	849	–	1099	–
	Title+URL	Title and URL only	7.7	27	31×	38	29×
	TF-IDF-5	Top 5 weighted terms	5.0	5	170×	8	138×
	TF-IDF-10	Top 10 weighted terms	10.0	24	37×	32	34×
	DTQ-1	1 surrogate query	4.6	22	39×	40	27×
	DTQ-5	5 surrogate queries	11.5	40	21×	67	16×

a trivial generation algorithm could simply return d random edges for each document, yielding a very cheap — but probably useless — document graph.

To determine whether the graphs produced by our heuristics are useful, we run a suite of end-to-end experiments. Figure 1 shows the results for both small and large graphs on TREC DL19 and DL20 tasks, using the DocT5Query index (with similar trends observed on the original index). Surprisingly, we observe that irrespective of how cheap our approximate graphs are to construct, LADR can perform as well as — and in some cases better than — the baseline Exhaustive graph. For example, the best performing graph overall appears to be constructed from using a single DTQ query, with an offline cost that is 27–39× lower than the Exhaustive baseline. While the DTQ methods dominate the Pareto frontier, the TF-IDF alternative (with $n = 5$ shown) also performs consistently well across the board; these methods are the cheapest to compute offline.

To further contextualize these results, we include two randomized graphs which replace 25% or 75% of the edges in the exhaustive graph with random edges, denoted Rand. This allows us to observe how (proactive) LADR behaves as graphs include non-relevant elements. Interestingly, LADR performs reasonably well with some noise, especially for the smaller graphs, but performance begins to drop steeply when the majority of the graph is composed of noise.

4.3 Early Precision Metrics

Although we primarily focused on recall, we also measured the effect of using different graphs approximations on early precision metrics such as NDCG@10. We found that early precision metrics are less sensitive to the input graph, with comparable performance observed (all within 0.02 points of Exhaustive)

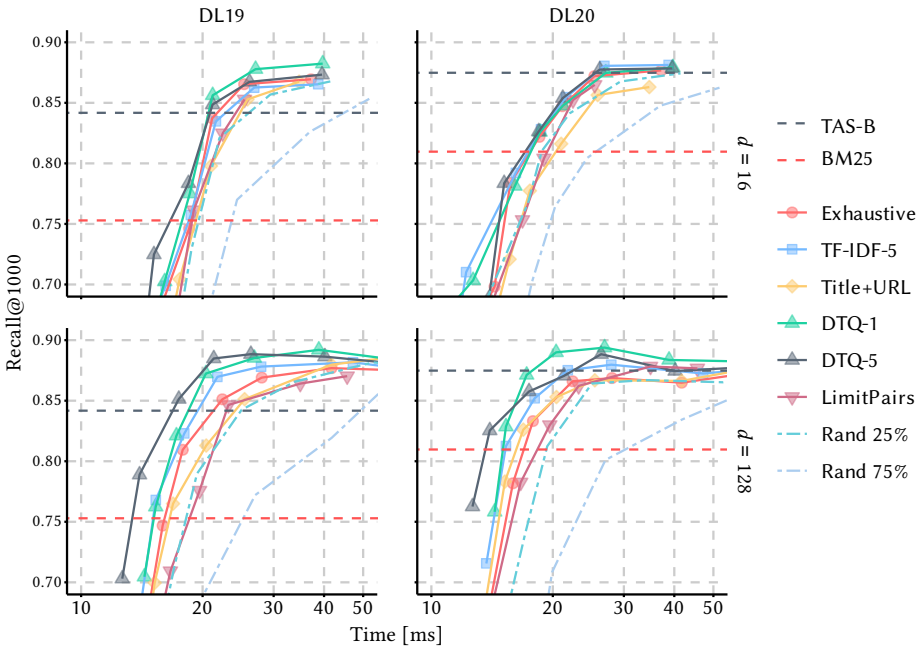

Fig. 1: Comparison of performance using proactive LADR and two corpus graph sizes ($d = 16$, top; and $d = 128$, bottom), on the DocT5Query index, for TREC DL19 (left) and DL20 topics (right). Non-graph baselines BM25 and TAS-B shown as horizontal lines on the y-axis. Note the logarithmic scale on the x-axis.

across all methods with the exception of the Rand 75% graph. This observation demonstrates that LADR is resilient to differences in the graph for ranking tasks; so long as the initial retrieval run yields good documents, LADR is a risk-averse strategy for incorporating other similar documents into the ranked results list.

4.4 Graph Similarity

Our final experiment explores the similarity between the approximate graphs and the Exhaustive target by computing the overlap between the adjacency list of each heuristic graph and the Exhaustive graph. Figure 2 shows the outcome on the larger $d = 128$ graphs. Evidently, the overlap is quite small, with most adjacency lists containing fewer than 25% of vertices in common with the Exhaustive graph. Given the strong performance observed in previous experiments, this leads us to a few potential hypotheses. Firstly, while the Exhaustive bag-of-words graph leans on the cluster hypothesis [10], it may not be representative of the ideal graph (of which user relevance plays a key role). This means that there may be significant room for improvement if corpus graphs of a higher quality were to be systematically constructed, an avenue we plan

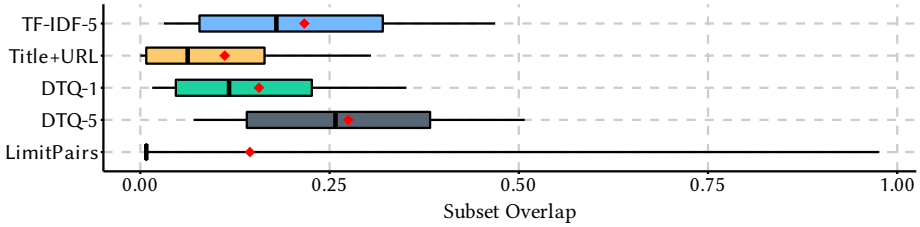

Fig. 2: The distribution of overlapping vertices comparing each heuristic to the Exhaustive graph with $d = 128$ on the DocT5Query index. The box covers the 25th to the 75th percentiles, with the median and mean shown as a line and a diamond, respectively. Whiskers extend to the 10th and 90th percentiles.

to explore in future work. Secondly, we hypothesize that the proactive LADR algorithm is robust to perturbations in the corpus graph, and is demonstrated by the strong effectiveness performance observed on the Rand 25% graphs (see Figure 1). Intuitively this makes sense, as the proactive LADR algorithm re-scores *all neighbors of all seed documents* — a good document will be re-ranked so long as it is a neighbor of at least one seed document.

5 Conclusion and Future Work

In this work, we applied a number of simple, readily available heuristics to reduce the cost of building corpus graphs with bag-of-words retrieval methods. These included heuristics that are intrinsic to any corpus (like TF-IDF, or limiting the number of pairs scored) as well as more specialized heuristics (using document titles, or expanded queries). Our experiments demonstrate that useful document graphs can be constructed at a fraction of the cost of the exhaustive baseline. We also demonstrated that the LADR mechanism [10] is surprisingly resilient to noise in the corpus graph, making it an attractive choice for practitioners using approximate methods for corpus graph construction.

There are a number of directions for future work. Firstly, it remains unclear how corpus graph construction scales to larger corpora and documents of greater length. Secondly, widening the analysis presented here to related algorithms that rely on corpus graphs (such as GAR [15] or LexBoost [9]) would paint a more comprehensive picture of the potential risks of using approximate corpus graphs. Finally, we only explored lexical corpus graphs in this work; a deeper exploration of approximate construction methods for dense corpus graphs is warranted, and some preliminary work in this direction has already been conducted [10].

Acknowledgments. This work was in part funded by the Australian Research Council (Project DP200103136) and a Google research scholar grant. We thank Maik Fröbe and the anonymous referees for their valuable feedback.

Disclosure of Interests. The authors have no competing interests of any sort.

Bibliography

[1] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., Wang, T.: MS MARCO: A human generated machine reading comprehension dataset. arXiv:1611.09268 (2016)

[2] Balasubramanian, N., Kumaran, G., Carvalho, V.R.: Exploring reductions for long web queries. In: Proc. SIGIR, pp. 571–578 (2010)

[3] Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. In: Proc. WWW, p. 131–140 (2007)

[4] Bruch, S., Lucchese, C., Nardini, F.M.: Efficient and effective tree-based and neural learning to rank. Found. Trends Inf. Ret. **17**(1), 1–123 (2023)

[5] Craswell, N., Mitra, B., Yilmaz, E., Campos, D., Voorhees, E.M., Soboroff, I.: TREC deep learning track: Reusable test collections in the large data regime. In: Proc. SIGIR, pp. 2369–2375 (2021)

[6] Dhulipala, L., Kabiljo, I., Karrer, B., Ottaviano, G., Pupyrev, S., Shalita, A.: Compressing graphs and indexes with recursive graph bisection. In: Proc. KDD, pp. 1535–1544 (2016)

[7] Francesco, A.G.D., Giannetti, C., Tonellotto, N., Silvestri, F.: Graph neural re-ranking via corpus graph. arXiv:2406.11720 (2024)

[8] Hofstätter, S., Lin, S.C., Yang, J.H., Lin, J., Hanbury, A.: Efficiently teaching an effective dense retriever with balanced topic aware sampling. In: Proc. SIGIR, pp. 113–122 (2021)

[9] Kulkarni, H., Goharian, N., Frieder, O., MacAvaney, S.: LexBoost: Improving lexical document retrieval with nearest neighbors. In: Proc. Symp. Doc. Eng. (2024), to appear

[10] Kulkarni, H., MacAvaney, S., Goharian, N., Frieder, O.: Lexically-accelerated dense retrieval. In: Proc. SIGIR, pp. 152–162 (2023)

[11] Kuzi, S., Zhang, M., Li, C., Bendersky, M., Najork, M.: Leveraging semantic and lexical matching to improve the recall of document retrieval systems: A hybrid approach. arXiv:2010.01195 (2020)

[12] Lemire, D., Boytsov, L.: Decoding billions of integers per second through vectorization. Soft. Prac. & Exp. **41**(1), 1–29 (2015)

[13] Leonhardt, J., Müller, H., Rudra, K., Khosla, M., Anand, A., Anand, A.: Efficient neural ranking using forward indexes and lightweight encoders. ACM Trans. Inf. Syst. **42**(5) (2024)

[14] Lin, S.C., Yang, J.H., Lin, J.: In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval. In: Proc. RepL4NLP-2021 at ACL-IJCNLP 2021, pp. 163–173 (2021)

[15] MacAvaney, S., Tonellotto, N., Macdonald, C.: Adaptive re-ranking with a corpus graph. In: Proc. CIKM, pp. 1491–1500 (2022)

[16] Macdonald, C., Tonellotto, N.: Declarative experimentation in information retrieval using PyTerrier. In: Proc. ICTIR, pp. 161–168 (2020)

[17] Mackenzie, J., Moffat, A.: Examining the additivity of top-k query processing innovations. In: Proc. CIKM, pp. 1085–1094 (2020)
[18] Mackenzie, J., Petri, M., Moffat, A.: Tradeoff options for bipartite graph partitioning **35**(08), 8644–8657 (2023)
[19] Mallia, A., Siedlaczek, M., Mackenzie, J., Suel, T.: PISA: Performant indexes and search for academia. In: Proc. SIGIR OSIRRC Workshop, pp. 50–56 (2019)
[20] Mallia, A., Siedlaczek, M., Suel, T.: An experimental study of index compression and DAAT query processing methods. In: Proc. ECIR, pp. 353–368 (2019)
[21] Metzler, D., Strohman, T., Turtle, H., Croft, W.B.: Indri at TREC 2004: Terabyte track. In: Proc. TREC (2004)
[22] Nogueira, R., Jiang, Z., Pradeep, R., Lin, J.: Document ranking with a pretrained sequence-to-sequence model. In: Proc. EMNLP, pp. 708–718 (2020)
[23] Nogueira, R., Lin, J.: From doc2query to docTTTTTquery. Tech. rep. (2019), URL https://cs.uwaterloo.ca/~jimmylin/publications/Nogueira_Lin_2019_docTTTTTquery-latest.pdf
[24] Nogueira, R., Yang, W., Cho, K., Lin, J.: Multi-stage document ranking with BERT. arXiv:1910.14424 (2019)
[25] Nogueira, R., Yang, W., Lin, J., Cho, K.: Document expansion by query prediction. arXiv:1904.08375 (2019)
[26] Robertson, S.: Understanding inverse document frequency: On theoretical arguments for idf. J. Doc. **60**, 503–520 (2004)
[27] Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M.: Okapi at TREC-3. In: Proc. TREC, pp. 109–126 (1994)
[28] Tonellotto, N., Macdonald, C., Ounis, I.: Efficient query processing for scalable web search. Found. Trends Inf. Ret. **12**, 319–500 (2018)
[29] Turtle, H., Flood, J.: Query evaluation: Strategies and optimizations. Inf. Proc. & Man. **31**(6), 831–850 (1995)
[30] Yang, Y., Bansal, N., Dakka, W., Ipeirotis, P., Koudas, N., Papadias, D.: Query by document. In: Proc. WSDM, pp. 34–43 (2009)