A Comparison of Top-k Threshold Estimation Techniques for Disjunctive Query Processing

Antonio Mallia
New York University
antonio.mallia@nyu.edu

Michał Siedlaczek
New York University
michal.siedlaczek@nyu.edu

Mengyang Sun[†]
Tsinghua University
sunmy19@mails.tsinghua.edu.cn

Torsten Suel
New York University
torsten.suel@nyu.edu

ABSTRACT

In the top-k threshold estimation problem, given a query q, the goal is to estimate the score of the result at rank k. A good estimate of this score can result in significant performance improvements for several query processing scenarios, including selective search, index tiering, and widely used disjunctive query processing algorithms such as MaxScore, WAND, and BMW. Several approaches have been proposed, including parametric approaches, methods using random sampling, and a recent approach based on machine learning. However, previous work fails to perform any experimental comparison between these approaches. In this paper, we address this issue by reimplementing four major approaches and comparing them in terms of estimation error, running time, likelihood of an overestimate, and end-to-end performance when applied to common classes of disjunctive top-k query processing algorithms.

ACM Reference Format:
Antonio Mallia, Michał Siedlaczek, Mengyang Sun, Torsten Suel. 2020. A Comparison of Top-k Threshold Estimation Techniques for Disjunctive Query Processing. In *Proceedings of the 29th ACM Int'l Conference on Information and Knowledge Management (CIKM'20), Oct. 19-23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3412080

1 INTRODUCTION

A lot of research has focused on how to efficiently execute queries in large search engines. This is often modeled as a top-k query processing problem: Given a document collection and associated index structures, a scoring function r, and a query q, the goal is to retrieve the k documents with the highest scores. Disjunctive top-k query processing, where all documents containing at least one query term are considered, is often used to obtain an initial set of candidate results that are then reranked using more complex machine-learned ranking functions. Many algorithms for disjunctive top-k query processing have been proposed, including MaxScore [25], WAND [4], and Block-Max methods such as [8, 9, 16, 17].

We consider a closely related problem, top-k threshold estimation, where we need to estimate the score of the k-th highest scoring document for a given query, preferably in a fraction of the time it would take to execute the query. Threshold estimation has been used to improve query processing efficiency in two main scenarios. First, threshold estimation can be used for resource selection in distributed search architectures such as selective search [12] and index tiering [3, 15, 18, 21]. In this case, by estimating the top-k thresholds for different index shards and tiers, we can estimate how many top-k results are contributed by each shard and tier. Thus, threshold estimation provides useful features to decide which parts of the index should be searched; see [2, 5] for examples in the context of selective search.

Second, it is well known [6, 7, 9, 11, 19] that good threshold estimates can boost the efficiency of top-k disjunctive query processing algorithms such as MaxScore [25], WAND [4], and Block-Max based methods [9, 16]. In this scenario, a strong initial estimate of the top-k threshold allows these algorithms to skip the evaluation of documents whose score cannot reach the threshold – though the precise way this is exploited depends on the algorithm.

Note that in this second scenario, it is not enough to get a good threshold estimate – we also need to limit the number of overestimates, since running the query algorithm with a too-high threshold estimate will result in less than k results above the threshold. To guarantee retrieval of the correct top-k results, we would then need to rerun the query at significant additional cost. This is in contrast to the first scenario of resource selection for unsafe methods such as selective search and index tiering, where overestimates are acceptable as long as the estimate is a useful feature for resource selection. On the other hand, we have more stringent efficiency requirements for threshold estimation in this case, since we need to obtain an estimate for each index shard.

In this paper, we perform the first experimental comparison of existing techniques for top-k threshold estimation. Our comparison includes the Taily approach [2], the Q_k method [6, 11, 26], the recently proposed BLR approach [19], and a method based on random sampling. We report the precision of the threshold estimates, the percentage of overestimates, the efficiency of the methods, and the resulting end-to-end performance improvements when used in the MaxScore, WAND, and BMW query processing algorithms.

2 ESTIMATION ALGORITHMS

We now briefly describe the main existing approaches.

Taily: This method was proposed in [2] in the context of shard selection in selective search. The basic idea is simple. It is observed that the top results for disjunctive queries are typically dominated by conjunctive results, and that the scores of the top conjunctive results for certain ranking functions can be approximated by a Gamma distribution, derived from the mean and variance of the

[†]Work done while the author was at New York University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412080

scores. These distributions and the term frequencies are then used to estimate the score threshold.

Taily assumes that the scoring function is the sum of term scores, which is true for cases such as BM25 and Query Likelihood Model (QL) [20]. The assumption of a Gamma distribution is justified for QL, but not really for BM25. Finally, as k increases, the difference between disjunctive and conjunctive score distributions becomes significant. On the plus side, estimation with Taily is very fast.

Simple Quantile-Based Prediction: Work in [6] and [11] proposes to precompute the k-highest impact score for each term and store it in the index lexicon. Clearly, the top-k threshold of a query can be lower-bounded by the maximum of the k-th highest impact scores of the query terms. Following [19], we refer to this method as Q_k. It is extremely fast, and never results in an overestimate.

Predictors based on Machine Learning: Recent work in [19] proposed methods based on machine learning. One, called BLR, is based on Bayesian Linear Regression, while two others, MLP-L2 and MLP-L4, use multi-layer perceptrons with two and four hidden layers, respectively. Both approaches enable a trade-off between the overestimation rate and the precision of the estimate.

Experiments in [19] indicate that BLR performs best, and thus we compare to BLR only. One disadvantage of both approaches is that they are much slower than Taily and Q_k, and they require very significant amounts of feature data to be added to the index.

Random Sampling: Random sampling is of course widely used in Computer Science, and can also be applied to threshold estimation. A fairly straightforward approach selects a sample of, say, around 1% of the document collection, by selecting each document independently with probability 0.01. Then the incoming query is run on the sample, and the top-k' threshold on the sample is used as an estimate of the top-k threshold on the full collection.

This leads to a trade-off between the precision of the estimate and the rate of overestimates, where a larger k' results in fewer overestimates and less precision. In particular, given k, k', and sample rate s, the probability of an overestimate is the probability that more than $k'-1$ of the top $(k-1)$ query results were selected into the sample, which is $\sum_{i=k'}^{k-1} \binom{k-1}{i} \cdot s^i(1-s)^{k-i-1}$.

Random sampling has been previously used for resources selection in distributed search, including [13, 22–24], though not in the exact formulation above. The speed and space consumption of sampling depends linearly on the sample size, but is typically much higher than for Taily and Q_k and more comparable to BLR.

Higher-Order Quantile Methods: A natural way to improve the quantile-based method is to store the k-th highest scores for certain sets of terms instead of just individual terms. One approach recently proposed in [26] caches the top-k thresholds of past queries, and then estimates the threshold of a future query by looking at past queries contained in the query. We propose and evaluate two related methods, called Q_k^2 and Q_k^3, that store the k-th highest scores for certain pairs and triplets of terms, respectively. At preprocessing time, suitable pairs and triplets are selected, and disjunctive queries are issued to compute their k-th highest scores. At estimation time, we use the maximum score over all query terms and all applicable query pairs and triplets.

We consider two approaches for selecting pairs and triplets. In the log-based approach, called Q_k^2-*log*, we select pairs that occur frequently in a training query log, while the other approach, Q_k^2-*lex*, uses term frequencies in the lexicon to choose pairs (and similarly for triplets). The resulting methods use more space than Q_k, but are still very fast, and much more accurate.

Sampling-Quantile Hybrid: It was suggested in [19] to combine BLR with Q_k by choosing the estimate provided by Q_k whenever it is higher than that from BLR. We adopt this idea by combining random sampling with Q_k^3-*log*, which was the most promising combination after an initial round of experiments.

Sampling with Dochits: This method attempts to decrease the sample size, and thus running time, by exploiting the significant skew in the rates at which documents are returned in the top k, similarly to [1, 10]. During preprocessing, a large query trace is executed on the collection, and we record the number of times each document is returned in the top k, called *dochits*. This distribution is highly skewed, with a fraction of the documents responsible for a majority of total dochits. This can be exploited as follows: We first take, say, a 1% sample, but only keep the documents in the sample with the highest dochits values, until, say, $t = 80\%$ of the total dochits in the 1% sample are covered. Through careful selection of parameters k', s, and t, a better trade-off between precision and running time can be achieved for the same overestimation rate.

3 EXPERIMENTAL RESULTS

Test Environment: All algorithms were reimplemented in C++17, and compiled with GCC 7.4.0 with the highest optimization settings. All runs are performed on a machine with 8 Intel Core i7-4770 Haswell cores clocked at 3.50GHz, with 32GiB RAM, running GNU/Linux 4.15. Our source code is publicly available[1] for readers interested in replicating the experiments.

Data Sets: We built inverted indexes for a commonly used document set, ClueWeb09B, consisting of roughly 50 million documents. The terms in the collection were lowercased and stemmed using the Porter2 stemmer; no stopwords were removed. Document IDs were assigned according to lexicographic order of their URLs. The inverted index was compressed using SIMD-BP128 [14]. Unless stated otherwise, we assume the BM25 scoring function.

To test estimation performance and resulting end-to-end query processing speed, we used TREC 2005 Terabyte Track Efficiency Task data and a combination of the the TREC 2007, 2008, and 2009 Million Query Track topics. (Due to space limitations, only results for the former are included here; results were very similar for the other set.) We selected 5000 random test queries, excluding single term queries, and left the rest as training queries for the BLR method. We also used the AOL query trace to select pairs and triplets in Q_k^2 and Q_k^3, and to estimate dochits for the method combining sampling and dochits, as these methods require much larger training sets.

Evaluation Metrics: We evaluate our implementations with respect to precision, rate of overestimates, running time, and end-to-end performance improvements when used in conjunction with the MaxScore, WAND, and BMW query processing algorithms.

There is a trade-off between precision and rate of overestimates for all methods, except the quantile-based ones, which have no overestimates. We decided to compare the various methods by limiting the rate of overestimates to some value, say 1% or 0.1%,

[1] https://github.com/pisa-engine/topk-threshold-estimation

	10				1000			
	Original BLR		Our BLR		Original BLR		Our BLR	
Quant.	O%	MUF	O%	MUF	O%	MUF	O%	MUF
0.50	59.86	0.89	59.76	0.89	57.32	0.88	57.46	0.88
0.30	28.80	0.88	28.88	0.88	32.10	0.86	32.38	0.86
0.10	8.44	0.79	8.54	0.79	8.36	0.80	8.54	0.80
0.05	5.74	0.74	5.86	0.74	4.32	0.75	4.40	0.75
0.01	2.90	0.64	2.86	0.64	1.26	0.65	1.40	0.65

Table 1: Comparison between our implementation of BLR and the one in [19], for $k = 10$ and $k = 1000$.

and then choosing parameters for each method to achieve this value. For sampling, this means setting k' appropriately, while for BLR we set parameters as discussed in [19]. For Taily, which was not designed to support a limit in the rate of overestimates, we choose a suitable $k' > k$ and then use the top-k' threshold estimate. We then compare precision and efficiency for each rate of overestimates.

There are a many measures for precision, such as mean square error (MSE) or mean absolute error (MAE). We adopted instead the mean under-prediction fraction (MUF) measure used in [19], which is the mean fraction of the predicted threshold and the actual threshold for those queries where no overestimate occurred. A good method should have MUF close to 1.0 with few overestimates.

Implementation Notes: Our **BLR** is a complete reimplementation in C++ of the Bayesian Linear Regression model used in [19]. Our implementation was validated against the original version and achieved almost identical results, as shown in Table 1. The original implementation was in Python and not optimized for efficiency as stated in [19]. **Taily** was reimplemented based on the description in [2]. To limit overestimate rates, we experimentally determined values $k' > k$ that achieve those rates. Our **Sample** method builds an inverted index on a random document sample, and then uses MaxScore to perform top-k' query processing on this index. To choose the pairs in Q_k^2-*log* and pairs and triplets in Q_k^3-*log*, we took all unique queries in the AOL query log and added any pairs and triplets that occur, resulting in about 13 million pairs and 60 million triplets. In the Q_k^2-*lex* methods, a pair was added if both inverted lists were longer than 2^{16}, leading to about 100 million pairs. For the hybrid, 10 million AOL queries were run to obtain dochits values for the documents in the collection.

	2	3	4	5	6+	avg
Q_k	0.79	0.69	0.62	0.59	0.55	0.70
Q_k^2-*lex*	0.96	0.89	0.83	0.79	0.72	0.89
Q_k^2-*log*	0.99	0.91	0.86	0.81	0.75	0.91
Q_k^3-*log*	0.99	0.96	0.93	0.90	0.83	0.95
Taily	0.48	0.22	0.04	0.02	0.00	0.26
BLR	0.56	0.64	0.68	0.71	0.71	0.63
Samp-0.2%	0.89	0.89	0.88	0.88	0.89	0.89
Samp-1%	0.94	0.94	0.94	0.94	0.94	0.94
Hybrid	0.99	0.97	0.94	0.93	0.91	0.97

Table 2: MUF for different methods and query lengths, with O% $\leq$ 1% and $k = 1{,}000$.

Comparing BLR and Sampling: In Figure 1, we compare sampling and BLR in terms of speed and MUF, for various rates of overestimates. In terms of speed, BLR is faster than a 0.5% or 1% sample, but slower than the smaller sample sizes. In terms of MUF,

O%	50	10	1	0.1
Q_k	-	-	-	0.79
Q_k^2-*log*	-	-	-	0.93
Q_k^3-*log*	-	-	-	0.96
Taily	0.55	0.51	0.38	0.32
Samp-0.2%	0.94	0.91	0.87	0.84
Samp-1%	0.97	0.96	0.94	0.92
Hybrid	0.98	0.98	0.97	0.97

Table 3: MUF using Query Likelihood as ranking function.

BLR is clearly worse than sampling at every rate of overestimates, even when compared to the smallest 0.1% sample. Thus, BLR does not seem to offer any improvements over straightforward sampling. We also show in purple data points for the method combining sampling and dochits, where we attempted to match the precision of a 1% sample with a smaller sample that allows faster estimates. Similar speed-ups can also be obtained for the other sample sizes.

Comparing MUF Across all Methods: In Table 2, we can see MUF results for all our methods, and for different query lengths, for a 1% rate of overestimates and $k = 1000$. We note that Taily is by far the worst method, with MUF always below 0.5 and falling with query length. Q_k and BLR are the next worst methods, where Q_k does better for short queries and BLR for longer ones. The Sample methods are not impacted by query length, as predicted by the theoretical analysis. A 1% sample does of course better than a smaller 0.2% sample, but also takes more time.

We also see that Q_k^2-*lex*, Q_k^2-*log* and Q_k^3-*log* all achieve significant boosts over Q_k, making the methods competitive with the Sample methods. Log-based methods appear to perform better than methods using collection statistics to select pairs, and the performance of Q_k^3-*log* is particularly impressive. As one would expect, these methods do best for shorter queries, while for longer queries Sample methods can do better. This motivated our hybrid method combining Q_k^3-*log* and a 0.2% sample, which is the overall best shown (slight improvements would be obtained by using a larger sample). Note that the relative performance of the methods was the same for other rates of overestimates.

Comparing MUF for QL Scoring: In Table 3, we show MUF when using a query likelihood scoring function instead of BM25, for various rates of overestimates. We do not show results for BLR, as this would require recomputing a new set of features. Since the quantile-based methods always have zero overestimates, we only compare them to other methods on a 0.1% rate of overestimates. Relative performance of the various methods is similar to Table 2. Note in particular that Taily is still the worst, and performs only slightly better on QL than on BM25.

Time and Space: Due to page constraints, we can only briefly discuss estimation times and space requirements. Taily and the quantile-based methods are extremely fast, taking at most a few microseconds per estimate. Speeds for BLR and Sample were shown in Figure 1, with running times in the tens to hundreds of microseconds. The time for the hybrid is dominated by the Sample method. Taily and Q_k use fairly little space, while the other methods use extra space on the order of a few percent of index size. BLR uses the most space, at least as described in [19], though this could probably be significantly reduced through appropriate engineering.

End-to-End Performance: We conclude with a comparison of the performance boosts achieved when using threshold estimation

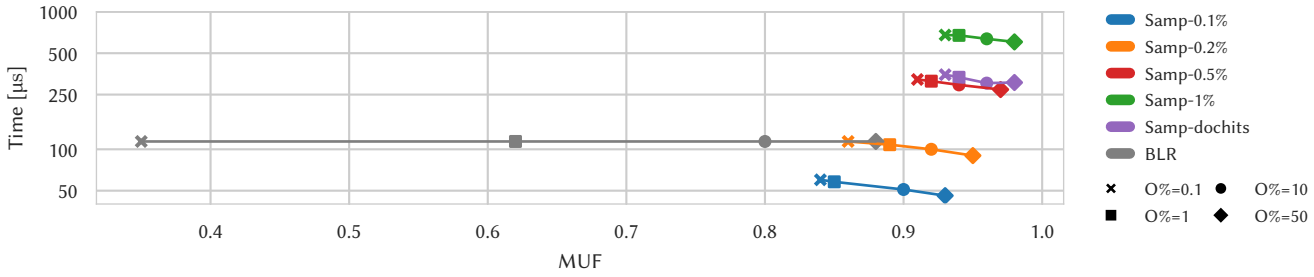

Figure 1: Estimation running time and MUF for sampling methods and BLR, for several rates of overestimates.

k			-	$\mathcal{O}$	Q_k	Q_k^2-*log*	Q_k^3-*log*	Taily	BLR	Samp-0.2%	Hybrid
	10	MaxScore	20.01	18.43	19.62	19.33	**19.13**	19.68	19.62	19.90	19.68
		WAND	24.82	23.05	24.34	24.21	**24.10**	24.21	24.46	24.67	24.21
		BMW	15.42	11.79	15.11	14.41	**13.65**	15.18	15.03	15.08	13.70
	100	MaxScore	24.74	22.00	23.88	23.35	**22.81**	24.48	24.26	23.95	22.99
		WAND	29.96	26.56	29.32	28.87	**28.50**	29.73	29.33	29.48	28.75
		BMW	24.63	18.92	24.22	22.16	21.13	24.64	23.98	23.45	**21.02**
	1000	MaxScore	35.73	28.03	33.60	31.47	30.10	35.40	34.00	32.36	**29.94**
		WAND	44.53	34.15	40.21	38.61	37.21	43.24	41.00	38.68	**37.06**
		BMW	43.45	30.53	40.29	36.66	34.58	42.66	41.08	37.70	**34.06**

Table 4: Query processing speed (in milliseconds) for ClueWeb09B and the TREC 2005 query log (O% $\leq$ 1%).

in highly tuned implementations of three well-known disjunctive top-k algorithms, MaxScore, WAND, and BMW, shown in Table 4. The first and second columns are the execution times with an initial threshold set to zero and to a perfect oracle value, respectively. We see that for $k = 1000$, the Hybrid method always does best, while for $k = 10$, Q_k^3-*log* does best. This is because for smaller k the potential gain of using threshold estimation is limited, and thus the cost of using sampling in the hybrid is higher than the gain.

4 DISCUSSION AND CONCLUDING REMARKS

In this paper, we have performed an experimental comparison of top-k threshold estimation methods. Our results show the best performance for quantile-based and sampling-based methods, and in particular for a hybrid of these. In particular, the Q_k^2-*log* and Q_k^3-*log* methods can give very good threshold estimates in a few microseconds with zero overestimates, while a hybrid combining Q_k^3-*log* and sampling gives even better results in scenarios such as top-1000 query processing where it is acceptable to spend up to a few hundred microseconds for a better estimate.

There are several open problems that can lead to further improvements. A better selection of pairs and triplets for the quantile-based methods should result in more precise estimates or smaller space requirements. Our use of dochits in the sampling approach seems somewhat clumsy, and we suspect there are more elegant biased sampling approaches that will perform better.

Acknowledgements. This research was supported by NSF Grant IIS-1718680 and a grant from Amazon.

REFERENCES

[1] I. S. Altingovde, R. Ozcan, and Ö. Ulusoy. 2012. Static Index Pruning in Web Search Engines: Combining Term and Document Popularities with Query Views. *ACM Trans. Inf. Syst.* 30, 1 (2012).
[2] R. Aly, D. Hiemstra, and T. Demeester. 2013. Taily: Shard Selection Using the Tail of Score Distributions. In *SIGIR*. 673–682.
[3] R. Baeza-Yates, V. Murdock, and C. Hauff. 2009. Efficiency Trade-Offs in Two-Tier Web Search Systems. In *SIGIR*. 163–170.
[4] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. 2003. Efficient Query Evaluation Using a Two-Level Retrieval Process. In *CIKM*.
[5] Z. Dai, Y. Kim, and J. Callan. 2017. Learning to rank resources. In *SIGIR*. 837–840.
[6] L. L. S. de Carvalho, E. S. de Moura, C. M. Daoud, and A. S. da Silva. 2015. Heuristics to Improve the BMW Method and Its Variants. *JIDM* 6, 3 (2015).
[7] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. 2013. A Candidate Filtering Mechanism for Fast Top-k Query Processing on Modern Cpus. In *SIGIR*. 723–732.
[8] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. 2013. Optimizing Top-k Document Retrieval Strategies for Block-Max Indexes. In *WSDM*.
[9] S. Ding and T. Suel. 2011. Faster Top-k Document Retrieval Using Block-Max Indexes. In *SIGIR*. 993–1002.
[10] S. Garcia. 2007. *Search engine optimisation using past queries.* Ph.D. Dissertation. RMIT University, Melbourne, Australia.
[11] A. Kane and F. Wm. Tompa. 2018. Split-Lists and Initial Thresholds for WAND-based Search. In *SIGIR*. 877–880.
[12] A. Kulkarni and J. Callan. 2015. Selective search: Efficient and effective search of large textual collections. *TOIS* 33, 4 (2015), 1–33.
[13] A. Kulkarni, A. S. Tigelaar, D. Hiemstra, and J. Callan. 2012. Shard Ranking and Cutoff Estimation for Topically Partitioned Collections. In *CIKM*. 555–564.
[14] D. Lemire and L. Boytsov. 2015. Decoding Billions of Integers Per Second Through Vectorization. *Software: Practice and Experience* 45, 1 (2015), 1–29.
[15] G. Leung, N. Quadrianto, A. J. Smola, and K. Tsioutsiouliklis. 2010. Optimal Web-Scale Tiering as a Flow Problem. In *NIPS*. 1333–1341.
[16] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. 2017. Faster BlockMax WAND with Variable-Sized Blocks. In *SIGIR*.
[17] A. Mallia and E. Porciani. 2019. Faster BlockMax WAND with longer skipping. In *European Conference on Information Retrieval*. 771–778.
[18] A. Ntoulas and J. Cho. 2007. Pruning Policies for Two-Tiered Inverted Index with Correctness Guarantee. In *SIGIR*. 191–198.
[19] M. Petri, A. Moffat, J. Mackenzie, J. S. Culpepper, and D. Beck. 2019. Accelerated Query Processing Via Similarity Score Prediction. In *SIGIR*. 485–494.
[20] Jay M Ponte and W Bruce Croft. 1998. A language modeling approach to information retrieval. In *SIGIR*. 275–281.
[21] K. M. Risvik, Y. Aasheim, and M. Lidal. 2003. Multi-Tier Architecture for Web Search Engines. In *Proc. of the First Conf. on Latin American Web Congress*. 132.
[22] L. Si and J. Callan. 2002. Using Sampled Data and Regression to Merge Search Engine Results. In *SIGIR*. 19–26.
[23] L. Si and J. P. Callan. 2003. Relevant document distribution estimation method for resource selection. In *SIGIR*. 298–305.
[24] P. Thomas and M. Shokouhi. 2009. SUSHI: Scoring Scaled Samples for Server Selection. In *SIGIR*. 419–426.
[25] H. Turtle and J. Flood. 1995. Query Evaluation: Strategies and Optimizations. *Information Processing & Management* 31, 6 (1995), 831–850.
[26] E. Yafay and I. S. Altingovde. 2019. Caching Scores for Faster Query Processing with Dynamic Pruning in Search Engines. In *CIKM*. 2457–2460.