Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures

Nima Asadi[1,2], Jimmy Lin[3,2,1]
[1]Dept. of Computer Science, [2]Institute for Advanced Computer Studies, [3]The iSchool
University of Maryland, College Park
nima@cs.umd.edu, jimmylin@umd.edu

ABSTRACT

This paper examines a multi-stage retrieval architecture consisting of a candidate generation stage, a feature extraction stage, and a reranking stage using machine-learned models. Given a fixed set of features and a learning-to-rank model, we explore effectiveness/efficiency tradeoffs with three candidate generation approaches: postings intersection with SvS, conjunctive query evaluation with WAND, and disjunctive query evaluation with WAND. We find no significant differences in end-to-end effectiveness as measured by NDCG between conjunctive and disjunctive WAND, but conjunctive query evaluation is substantially faster. Postings intersection with SvS, while fast, yields substantially lower end-to-end effectiveness, suggesting that document and term frequencies remain important in the initial ranking stage. These findings show that conjunctive WAND is the best overall candidate generation strategy of those we examined.

Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms: Algorithms, Experimentation

Keywords: query evaluation; postings intersection

1. INTRODUCTION

One possible architecture for web retrieval breaks document ranking into three stages: candidate generation, feature extraction, and document reranking. There are two main reasons for this multi-stage design. First, there is a general consensus that learning to rank provides the best solution to document ranking [13, 12]. As it is difficult to apply machine-learned models over the *entire* collection, in practice a candidate list of potentially-relevant documents is first generated. Thus, learning to rank is actually a *reranking* problem (hence the first and third stages). Second, separating candidate generation from feature extraction has the advantage of providing better control over cost/quality tradeoffs. For example, term proximity features are significantly more costly to compute than unigram features; therefore, by decoupling the first two stages in the architecture, systems can exploit "cheap" features to generate candidates quickly and only compute "expensive" term proximity features when necessary—thus decreasing overall query evaluation latency.

Given a fixed set of features and a learning-to-rank model, we explore effectiveness/efficiency tradeoffs in the candidate generation stage, comparing postings intersection with SvS, conjunctive query evaluation with WAND, and disjunctive query evaluation with WAND. Previous work has suggested that conjunctive query evaluation yields early precision results that are at least as good as disjunctive query evaluation, but is much faster. We experimentally confirm this observation, and additionally show that postings intersection (with results sorted by spam scores) yields substantially worse output within the same framework.

The contribution of this work is an empirical evaluation of common candidate generation algorithms in a multi-stage retrieval architecture. By fixing the feature generation and reranking stages, we are able to isolate the end-to-end effectiveness and efficiency implications of different algorithms.

2. BACKGROUND AND RELATED WORK

We begin with a more precise specification of our three-stage architecture, illustrated in Figure 1. The input to the candidate generation stage is a query Q and the output is a list of k document ids $\{d_1, d_2, ...d_k\}$. In principle, this can be considered a sorted list, but that detail is unimportant here. These document ids serve as input to the feature extraction stage, which returns a list of k feature vectors $\{\mathbf{f}_1, \mathbf{f}_2, ...\mathbf{f}_k\}$, each corresponding to a candidate document. These serve as input to the third document reranking stage, which typically applies a machine-learned model to produce a final ranking.

This multi-stage retrieval architecture has recently been explored by many researchers. Some have examined the entire pipeline; for example, Macdonald et al. [14] assessed the impact of variables such as the number of candidate documents and the objective metric to use when training the learning-to-rank model (however, they did not explore effectiveness/efficiency tradeoffs with candidate generation). Others have specifically looked at candidate generation, e.g., postings intersection on multi-core architectures [17], dynamic pruning [18], and approximate techniques [3]. There has been some work focused on the feature extraction stage, exploring how to best represent positional information; two studies have independently come to the conclusion that it is advantageous to store document positions in a document vector representation distinct from the inverted index [1, 2]. Finally, most work on learning to rank [13, 12] assumes such

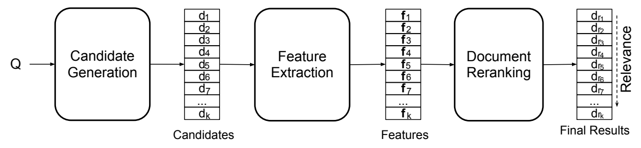

Q
Candidate Generation
d_1 d_2 d_3 d_4 d_5 d_6 d_7 ... d_k
Candidates
Feature Extraction
$\mathbf{f}_1$ $\mathbf{f}_2$ $\mathbf{f}_3$ $\mathbf{f}_4$ $\mathbf{f}_5$ $\mathbf{f}_6$ $\mathbf{f}_7$ ... $\mathbf{f}_k$
Features
Document Reranking
d_{r_1} d_{r_2} d_{r_3} d_{r_4} d_{r_5} d_{r_6} d_{r_7} ... d_{r_k}
Relevance
Final Results

Figure 1: Illustration of a multi-stage retrieval architecture with distinct candidate generation, feature extraction, and document reranking stages.

an architecture, because the input to the machine-learned model is a set of feature vectors. Occasionally, this fact is made explicit: for example, Cambazoglu et al. [6] experimented with reranking 200 candidate documents to produce the final ranked list of 20 results.

There are two general approaches to candidate generation: conjunctive and disjunctive query processing. In the first, only documents that contain *all* query terms are considered, whereas in the second, any document with at least one query term is potentially retrievable. One popular algorithm is WAND [4], which can operate in either conjunctive or disjunctive mode with respect to a particular scoring model (e.g., BM25). WAND uses a pivot-based pointer movement strategy to avoid needlessly evaluating documents that cannot be in the final top k ranking. An alternative approach to candidate generation involves simple postings intersection, without reference to a particular scoring model. Culpepper and Moffat [8] demonstrated SvS to be the best algorithm for accomplishing this. Note that since SvS works by iteratively intersecting the current result set with the next shortest postings list, it must exhaustively compute the intersection set. In this paper we explore both WAND and SvS for candidate generation.

It is a well-known fact that disjunctive query processing is much slower than conjunctive processing. However, it is unclear what impact output quality at the candidate generation stage has on end-to-end effectiveness. Although Broder et al. [4] found conjunctive processing to yield higher early precision, the results were not in the context of learning-to-rank experiments. We hypothesize that end-to-end effectiveness is relatively insensitive to candidate generation quality, due to an emphasis on early precision in most web search tasks today—we simply need to make sure there are enough relevant documents for the machine-learned model to identify. Thus, there is an interesting possibility that conjunctive query processing might lead to both good *and* fast results. Our paper explores this hypothesis.

3. EXPERIMENTAL SETUP

3.1 Candidate Generation Algorithms

We examined four approaches to candidate generation, outlined below. For each, we varied the number of candidates generated k, where $k \in \{100, 250, 500, 1000\}$.

SvS$_{\text{Spam}}$. Since SvS must compute the entire intersection set, to obtain k documents, we sort the intersection results by a static prior and return the top k.[1]

SvS$_{\text{BM25}}$. In this approach, we first compute the full intersection set, and then in a second pass we compute BM25 scores for every document in the intersection set. After the second pass, the top k documents are returned.

WAND$_{\text{Con}}$. We evaluate the query in conjunctive mode and return the top k documents based on BM25.

WAND$_{\text{Dis}}$. We evaluate the query in disjunctive mode and return the top k documents based on BM25.

All algorithms were implemented in C and an equal amount of time was spent optimizing each to ensure a fair comparison. All conditions used the same (non-positional) inverted index (even though SvS$_{\text{Spam}}$ does not need access to *tf*'s) and for SvS$_{\text{BM25}}$ we modified the accumulators to hold the *tf*'s extracted from the postings for the second-pass scoring. We assumed that all index structures are retained in memory, so query evaluation never involves hitting disk. Experiments were performed on a server running Red Hat Linux, with dual Intel Xeon "Westmere" quad-core processors (E5620 2.4GHz) and 128GB RAM.

3.2 Test Collections and Metrics

We performed experiments on the ClueWeb09 collection, a best-first web crawl from early 2009. Our experiments used only the first English segment, which has 50 million documents (247GB compressed). The Waterloo spam scores [7] were used as the static priors for SvS reranking.

For evaluation, we used three different sets of queries: first, the TREC 2005 terabyte track "efficiency" queries (50,000 queries total).[2] Since there are no relevance judgments for these queries, they were used solely for efficiency experiments. Second, a set of 100,000 queries sampled randomly from the AOL query log [16]. Our sample retains the query length distribution of the original dataset. Similar to the TREC 2005 terabyte track queries, we used these queries only to evaluate efficiency.

Finally, we used the TREC web track topics from 2009–2011, 150 in total. These topics comprise a complete test collection in that we have relevance judgments, but there are too few queries for meaningful efficiency experiments. We performed five fold cross validation, using three folds for training, one for validation, and one for testing.

We collected two figures of merit: the first was end-to-end effectiveness in terms of NDCG. For efficiency, we measured query latency of the candidate generation stage, defined as the elapsed time between the moment a query is presented to the system and the time when top k candidates are retrieved and forwarded to the feature extraction stage. To capture variance, we repeated runs five times.

3.3 Features and Ranking Model

We used a standard suite of features very similar to those described in previous work [15, 18]. They consist of basic information retrieval scores (e.g., language modeling and BM25 scores), term proximity features (exact phrases, ordered windows, unordered windows), query-independent features (e.g., PageRank, content quality score [7], etc.). Relevant features were computed across multiple fields: the entire document, anchor text, as well as title fields. In total, there are 91 features.[3]

[1]One caveat worth mentioning: the common trick to renumber docids based on the static prior doesn't help much here since we need to compute the complete intersection set regardless; it would save the final sort by score but time spent on that is negligible.

[2]`http://www-nlpir.nist.gov/projects/terabyte/`

[3]The complete test collection, including feature descriptions, can be found at `https://github.com/lintool/ClueWeb09-TREC-LTR`.

		NDCG@1				NDCG@5				NDCG@20				NDCG			
	Model	100	250	500	1000	100	250	500	1000	100	250	500	1000	100	250	500	1000
Spam	BM25	0.06	0.09	0.09	0.11	0.05	0.06	0.08	0.10	0.04	0.05	0.06	0.08	0.027	0.044	0.060	0.077
	Linear	0.06	0.08	0.10	0.14	0.05	0.06	0.08	0.10	0.03	0.05	0.07	0.08	0.027	0.045	0.060	0.079
	λ-MART	0.05	0.09	0.10	0.15	0.05	0.07	0.09	0.11	0.03	0.05	0.06	0.08	0.027	0.045	0.060	0.079
Con.	BM25	0.25	0.25	0.25	0.25	0.22	0.22	0.22	0.22	0.21	0.21	0.21	0.21	0.230	0.280	0.304	0.327
	Linear	0.26	0.25	0.25	0.24	0.25	0.24	0.25	0.23	0.25*	0.24*	0.23*	0.23*	0.248*	0.297*	0.317*	0.340
	λ-MART	0.26	0.25	0.24	0.25	0.28*	0.27*	0.25*	0.25*	0.26*	0.25*	0.24*	0.23*	0.250*	0.300*	0.318*	0.340*
Dis.	BM25	0.25	0.25	0.25	0.25	0.22	0.22	0.22	0.22	0.21	0.21	0.21	0.21	0.230	0.281	0.308	0.331
	Linear	0.26	0.25	0.25	0.23	0.25	0.24	0.24	0.23	0.25*	0.24*	0.23*	0.23*	0.248*	0.298*	0.320*	0.344*
	λ-MART	0.26	0.25	0.25	0.24	0.28*	0.27*	0.26*	0.25*	0.26*	0.25*	0.24*	0.23*	0.250*	0.301*	0.321*	0.344*

Table 1: NDCG at different cutoffs using different candidate generation algorithms: SvS_{Spam} (Spam), SvS_{BM25} and WAND_{Con} (Con.), and WAND_{Dis} (Dis.). Vertical blocks show each metric, and columns show the number of candidate documents retrieved. * indicates statistical significance vs. BM25 using the t-test (p<0.05).

For the third stage reranking model, we used two different learning-to-rank techniques. In the first, we trained a linear model using the greedy feature selection approach [15]. The model is iteratively constructed by adding features, one at a time, according to a greedy selection criterion. During each iteration, the feature that provides the biggest gain in effectiveness (as measured by NDCG [11]) after being added to the existing model is selected. This yields a sequence of one-dimensional optimizations that can easily be solved using line search techniques. The algorithm stops when the difference in NDCG between successive iterations drops below a given threshold (10^{-4}). This training procedure is simple, fast, and fairly effective.

Separately, we learned a LambdaMART model [5] using the open-source jforests implementation[4] [10]. To tune parameters, we used grid search as suggested by the authors and selected the parameter setting that results in the largest gain in NDCG on the validation set: max number of leaves (9), feature and data sub-sampling (0.3), minimum observations per leaf (0.75), and the learning rate (0.05).

4. RESULTS

Table 1 summarizes NDCG at different rank cutoffs and no cutoff for all combinations of candidate generation and reranking models. The first horizontal block of the table labeled "Spam" refers to SvS_{Spam}, "Con." indicates SvS_{BM25} or WAND_{Con} (as both algorithms produce the same results), and "Dis." shows the use of WAND_{Dis}. Within each block of the table, "BM25" indicates reranking candidates with BM25, which serves as the baseline (note this only alters the postings intersection candidate results; in the other cases, this is equivalent to a reranker that does nothing); "Linear" is the linear model learned using Metzler's greedy-feature selection method; and "λ-MART" is LambdaMART. Each column shows the number of candidate documents retrieved.

Candidates generated using SvS_{Spam} yield very low end-to-end effectiveness. This shows that combining postings intersection and with a static prior does not provide a sufficiently discriminative signal to include relevant documents in the top k, where k is small relative to the size of the collection. The quality of the static prior is not to blame here, as the Waterloo spam scores have been demonstrated to be very helpful in reranking [7]. Of course, as we increase the size of k, the results of SvS_{Spam} will approach that of the other candidate generation algorithms, but at the cost of more time spent performing feature extraction. It is clear that both term frequencies and document frequencies cannot be ignored in candidate generation.

[4]`http://code.google.com/p/jforests/`

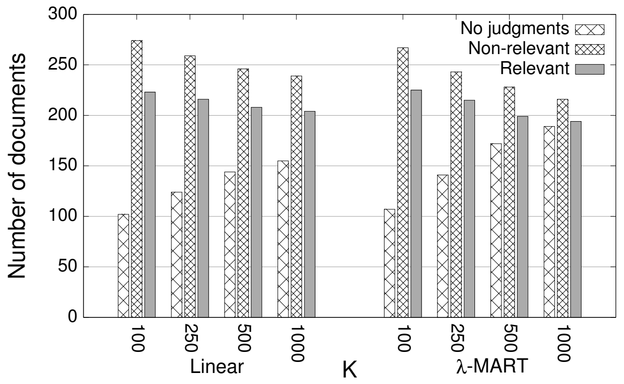
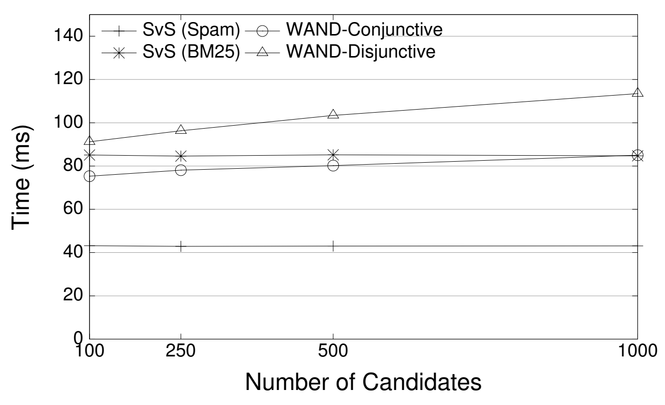


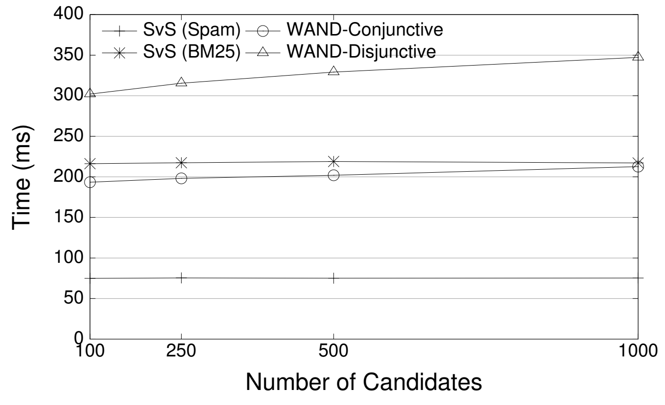
Figure 2: Number of rel, non-rel, and unjudged documents in the top 20 for different candidate sizes using WAND_{Dis} (averaged across folds).

On the other hand, WAND_{Con} (and SvS_{BM25}, which produces exactly the same results) yields NDCG scores that are statistically indistinguishable from those produced by WAND_{Dis} (with no cutoff, WAND_{Dis} yields slightly higher scores, but the differences are not statistically significant). In other words, with BM25 scoring, conjunctive candidate generation and disjunctive candidate generation are equally good from the end-to-end effectiveness perspective. This finding is consistent with the results of Broder et al. [4] (but in a learning-to-rank context).

Increasing k improves NDCG at all rank cutoffs for candidates obtained by SvS_{Spam}. We observe the same trend for NDCG without cutoff in all settings. This is expected since larger k translates into higher recall at the candidate generation stage, i.e., more relevant documents are available to the reranker. However, with conjunctive and disjunctive query processing we see a decline in NDCG at rank cutoffs 5 and 20 as we increase the number of candidate documents; this trend is consistent for both the linear model and LambdaMART. To examine this effect in a bit more detail, we computed the number of relevant, non-relevant, and unjudged documents in the top 20 obtained by reranking WAND_{Dis}. Figure 2 shows these statistics for different values of k. We see that as k increases, more unjudged documents find their way to the top 20, while both the number of relevant and non-relevant documents decreases. Thus, this trend appears to be an artifact of the test collection and not a property of the candidate generation algorithms.



(a) Terabyte



(b) AOL

Figure 3: Average candidate generation per-query latency across five trials.

Figure 3 shows per-query latency of the candidate generation stage for different values of k. On average, $\mathrm{SvS}_{\mathrm{Spam}}$ is the fastest since the ranking function is query independent: score computation boils down to table lookups of spam scores. We see that $\mathrm{SvS}_{\mathrm{BM25}}$ is on par with $\mathrm{WAND}_{\mathrm{Con}}$ for larger k values. $\mathrm{SvS}_{\mathrm{BM25}}$ retrieves the full intersection set and computes scores for every document in the set regardless of k; therefore, increasing k has no impact on latency. On the other hand, $\mathrm{WAND}_{\mathrm{Con}}$ slows with increasing k (but is faster with a small k). Conjunctive query evaluation with WAND can only terminate when a postings list is fully consumed. However, this termination is mostly independent of k; the only factor affected by k is the set of heap operations performed during query evaluation. Finally, $\mathrm{WAND}_{\mathrm{Dis}}$ is not only the slowest overall, but latency grows with larger values of k faster than with conjunctive evaluation.

5. CONCLUSIONS

Our experiments show that conjunctive WAND is the best candidate generation strategy of those examined: in terms of end-to-end effectiveness, it is statistically distinguishable from disjunctive query evaluation using WAND, but much faster in query evaluation. Note that the "block-max" optimization to WAND proposed by Ding and Suel [9] does not affect this conclusion—although the optimization increases disjunctive query evaluation speed, it remains slower than conjunctive processing. In addition, we also show that postings intersection with static priors yields very poor end-to-end effectiveness, at least with the size of the candidate sets we examined. This suggests that postings intersection, while an interesting algorithmic challenge since it represents the more general problem of intersecting two lists of sorted integers, is not particularly useful by itself if one's goal is to build effective and efficient search systems.

6. ACKNOWLEDGMENTS

This work has been supported by NSF under awards IIS-0916043, IIS-1144034, and IIS-1218043. Any opinions, findings, or conclusions are the authors' and do not necessarily reflect those of the sponsor. The first author's deepest gratitude goes to Katherine, for her invaluable encouragement and wholehearted support. The second author is grateful to Esther and Kiri for their loving support and dedicates this work to Joshua and Jacob.

7. REFERENCES

[1] D. Arroyuelo, S. González, M. Marin, M. Oyarzún, and T. Suel. To index or not to index: Time-space trade-offs in search engines with positional ranking functions. *SIGIR*, 2012.
[2] N. Asadi and J. Lin. Document vector representations for feature extraction in multi-stage document ranking. *IRJ, in press*, 2012.
[3] N. Asadi and J. Lin. Fast candidate generation for two-phase document ranking: Postings list intersection with Bloom filters. *CIKM*, 2012.
[4] A. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. *CIKM*, 2003.
[5] C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010.
[6] B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. *WSDM*, 2010.
[7] G. Cormack, M. Smucker, and C. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *arXiv:1004.5168v1*, 2010.
[8] J. Culpepper and A. Moffat. Efficient set intersection for inverted indexing. *TOIS*, 29(1), 2010.
[9] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. *SIGIR*, 2011.
[10] Y. Ganjisaffar, R. Caruana, and C. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. *SIGIR*, 2011.
[11] K. Järvelin and J. Kekäläinen. Cumulative gain-based evaluation of IR techniques. *TOIS*, 20(4):422–446, 2002.
[12] H. Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool, 2011.
[13] T.-Y. Liu. Learning to rank for information retrieval. *FnTIR*, 3(3):225–331, 2009.
[14] C. Macdonald, R. Santos, and I. Ounis. The whens and hows of learning to rank for web search. *IRJ, in press*, 2012.
[15] D. Metzler. Automatic feature selection in the Markov random field model for information retrieval. *CIKM*, 2007.
[16] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. *InfoScale*, 2006.
[17] S. Tatikonda, B. Cambazoglu, and F. Junqueira. Posting list intersection on multicore architectures. *SIGIR*, 2011.
[18] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. *WSDM*, 2013.