Query Driven Algorithm Selection in Early Stage Retrieval

Joel Mackenzie
RMIT University
Melbourne, Australia
joel.mackenzie@rmit.edu.au

J. Shane Culpepper
RMIT University
Melbourne, Australia
shane.culpepper@rmit.edu.au

Roi Blanco*
Amazon
Barcelona, Spain
roiblan@amazon.com

Matt Crane
University of Waterloo
Waterloo, Canada
matt.crane@uwaterloo.ca

Charles L. A. Clarke
University of Waterloo
Waterloo, Canada
claclark@gmail.com

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

ABSTRACT

Large scale retrieval systems often employ cascaded ranking architectures, in which an initial set of candidate documents is iteratively refined and re-ranked by increasingly sophisticated and expensive ranking models. In this paper, we propose a unified framework for predicting a range of performance-sensitive parameters based on minimizing end-to-end effectiveness loss. The framework does not require relevance judgments for training, is amenable to predicting a wide range of parameters, allows for fine tuned efficiency-effectiveness trade-offs, and can be easily deployed in large scale search systems with minimal overhead. As a proof of concept, we show that the framework can accurately predict a number of performance parameters on a query-by-query basis, allowing efficient and effective retrieval, while simultaneously minimizing the tail latency of an early-stage candidate generation system. On the 50 million document ClueWeb09B collection, and across 25,000 queries, our hybrid system can achieve superior early-stage efficiency to fixed parameter systems without loss of effectiveness, and allows more finely-grained efficiency-effectiveness trade-offs across the multiple stages of the retrieval system.

ACM Reference Format:
Joel Mackenzie, J. Shane Culpepper, Roi Blanco, Matt Crane, Charles L. A. Clarke, and Jimmy Lin. 2018. Query Driven Algorithm Selection in Early Stage Retrieval. In *WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining , February 5–9, 2018, Marina Del Rey, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3159652.3159676

1 INTRODUCTION

The competing goals of maximizing both efficiency and effectiveness in large scale retrieval systems continue to challenge builders of search systems as the emphasis in modern architectures evolves towards multi-stage retrieval [40]. Many old efficiency problems become new again in the increasingly complex cascade of document re-ranking algorithms being developed. For example, research groups can focus on early stage retrieval efficiency [4, 16, 54], balancing feature costs [12, 53, 56], or improving the performance of the learning-to-rank algorithms [5, 26, 34, 35, 37].

While great strides have been made in all of these areas, gaps remain in our understanding of the delicate balance between efficiency and effectiveness in each "stage" of the re-ranking cascade. One of the most significant limitations preventing further progress is training data availability. While query sets to measure efficiency in various collections are plentiful, the costs of gathering relevance judgments in order to measure effectiveness limit the number of topics available for more detailed trade-off analyses.

In this work, we explore how to apply a *reference list* framework [13, 45, 46, 55] to alleviate this problem. We leverage a new labelling framework to build machine-learned models capable of predicting candidate set sizes, algorithm aggressiveness parameters, and query latency to balance efficiency and effectiveness on a query-by-query basis. In particular, we focus on using this unified framework to reduce the early-stage *tail latency* [18, 24, 25, 27], which are queries with a 95th percentile (or greater) response time in the candidate generation stage. We explore three important research questions:

Research Question 1 (RQ1): *What is the best way to use reference lists to accurately perform dynamic parameter predictions in early stage retrieval on a per-query basis?*

Research Question 2 (RQ2): *What is the relationship between tail latency and index traversal algorithms, and can our new prediction framework be used to reliably provide worst case guarantees on first-stage query efficiency?*

Research Question 3 (RQ3): *What combination of predictors will lead to efficient first-stage retrieval, minimizing the number of candidate documents returned in the first stage (and thus making later stages more efficient), while also minimizing the effectiveness loss in final stage re-ranking?*

In answering these questions, our research contributions include:

(1) A unified framework that can be used to predict a wide variety of performance-sensitive parameters in multi-stage retrieval systems, which can be trained without requiring relevance judgments.
(2) A pragmatic, yet highly tunable and easy to implement approach for parameterizing search systems on a per-query basis.

*This work was conducted while this author was at RMIT University.

WSDM 2018, February 5–9, 2018, Marina Del Rey, CA, USA

ACM ISBN 978-1-4503-5581-0/18/02…$15.00
https://doi.org/10.1145/3159652.3159676

(3) A pathway to more fine-tuned per-query optimization techniques, and the tools necessary to implement and test systems leveraging these ideas.

We achieve these goals using three ideas. First, we exploit the idea of query efficiency prediction and static pre-retrieval features to build a unified prediction framework. Next, we explore the relationship between the number of documents returned in a top-k candidate set and the efficiency of the index traversal algorithm. Finally, the efficiency predictors are integrated with an effectiveness loss minimization prediction. Together, this series of "Stage-0" pre-retrieval predictions produces a pipeline that allows fine-grained efficiency-effectiveness trade-offs in an existing multi-stage retrieval system.

2 BACKGROUND AND RELATED WORK

Efficient Query Processing. Efficient query processing can be attained through a range of index organizations and traversal strategies based on the *inverted index* data structure [59]. Document-at-a-time (DaaT) query processing relies on postings lists being sorted in ascending order of the document identifiers. At query time, a pointer is set at the beginning of each postings list. Once the current document has been evaluated, the pointers are forwarded to the next document in the lists. An efficient method for *disjunctive* DaaT processing is the *Weak-AND* (Wand) algorithm [8]. Wand is now a well understood algorithm used for a wide variety of different search tasks [4, 14, 21, 38, 41, 47].

Ding and Suel [20] (and at a similar time, Chakrabarti et al. [10]) explored an improved version of Wand named *Block-Max WAND* (Bmw). The key observation in Bmw is that since many index compression algorithms are block-based [30, 58], skipping can be achieved at the block level, thus saving an entire block decompression. Further enhancements to Bmw have been made in the literature, usually by using additional auxiliary structures that improve the efficiency of query processing at the cost of additional space consumption [19, 42, 43].

Another entirely different method for top-k query processing is the score-at-a-time (SaaT) approach. Anh et al. [2] made the observation that the term weight for any given document could be pre-computed and stored, rather than the term frequencies. Since the term weights are typically floating point numbers, they are quantized into integer values to facilitate compression [2], the range of which impacts both effectiveness and efficiency [15]. For SaaT processing, each postings list is sorted by *decreasing* impact score, which allows the most high scoring documents for each term to be processed first, and can allow for early termination without sacrificing effectiveness. Recently, Lin and Trotman [32] introduced Jass, a modern SaaT algorithm which can be used for *anytime* retrieval, making it suitable for use in time-constrained environments and for controlling tail latency.

Tail Latency. The tail latency of a system corresponds to the response times occurring above some high percentile, such as the 95th, 99th or even the 99.99th percentile [27, 57]. As collections grow larger, systems must scale accordingly. As systems become more complex, the probability of increasing the tail latency also increases [18], particularly for distributed architectures where end-to-end latency is often bound by the slowest component. Reducing the tail latency can be addressed through either hardware or software optimizations, or both. For example, replicating and partitioning collections [18, 22, 28] allows effective load balancing which can reduce the system tail latency.

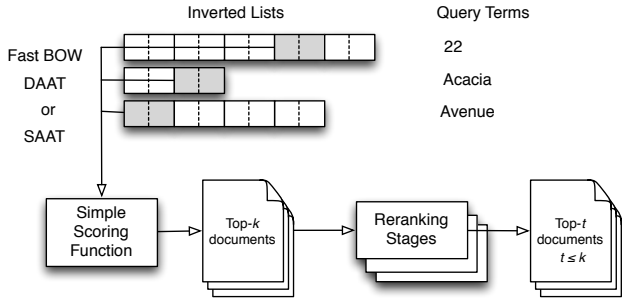

Figure 1: Architecture of a typical multi-stage retrieval system. Queries are first processed using an efficient bag-of-words retrieval algorithm. The initial candidate set of k documents then undergoes a series of re-ranking stages where the candidate pool is shrunk, and more expensive learning-to-rank algorithms are used to produce a final set of top-t documents to return to the user, where often $t \ll k$.

Previous work has attempted to reduce the tail response times in a range of different contexts. Jeon et al. [25] focus on 99th percentile tail latency at the level of a single *Index Server Node* (ISN) by predicting long running queries, and running them in parallel. Queries that are *not* predicted as long running are processed sequentially, which avoids the overhead cost of parallelization. Another recent work targets reducing the extreme tail latency (at the 99.99th percentile) [24, 27]. This target is achieved through *Dynamic, Delayed, Selective* (DDS) prediction. DDS prediction works as follows. First, a new query is processed for a short time, such as 20ms, and dynamic features are collected from this initial processing. Then, new dynamic features (and, some additional static features) are used to predict whether the query is a long running query. If so, then the query will be accelerated using parallelization. The prediction error is also estimated, and is used to improve coverage of mispredicted true long running queries.

Multi-Stage Search Architectures. Multi-stage retrieval has become the dominant model in modern web search systems [3, 4, 9, 36, 37, 40]. In this approach, a set of candidate documents is generated that is likely to be relevant to a query, and then in one or more stages, the document sample is iteratively reduced and reordered using a series of increasingly expensive machine learning techniques. Since re-ordering can be computationally expensive and is sensitive to the number of documents that must be reordered, minimizing the size of the candidate set is an important problem [9, 12, 36, 48].

Figure 1 shows a typical multi-stage retrieval architecture. A fast bag-of-words retrieval algorithm produces a top-k candidate set. This initial set of documents is then re-ranked one or more times using a learning-to-rank algorithm to produce a final output set of t documents, where $t \leq k$, and can be $t \ll k$ in some configurations.

Efficiency remains an important problem in multi-stage retrieval, with papers focused on cascaded ranking [12, 40, 56], early exit optimizations [9, 17], and efficient candidate generation [4, 54]. Recently, Wang et al. [54] proposed a fast candidate generation framework which opts to build a two-layer index. The bottom layer is the standard inverted index, and the top layer is a single or dual-term auxiliary structure which stores a subset of the bottom layer documents, sorted by impact score. At query time, a prefix of the top layer is accessed, which is then refined by accessing the lower layer of the index, which was shown to be efficient in practice. Such an approach could easily be used within our framework as it essentially attempts to improve the efficiency of the candidate generation phase of multi-stage retrieval, but we leave this as future work.

Reference List Evaluation in Multi-Stage Retrieval. One obvious question arises when trying to measure trade-offs in multi-stage retrieval systems – how can we quantify the impact on effectiveness when modifying different components of the search system? One approach is to simply make changes to the system, and re-compute a standard information retrieval metric such as average precision (AP), expected reciprocal rank (ERR), normalized discounted cumulative gain (NDCG), or rank biased precision (RBP) on the last stage result [11, 39]. However, this is unwieldy in practice, as it can be very difficult to identify exactly what changes are resulting in effectiveness differences.

A more compelling approach is to compute intermediate results at different stages of re-ranking, and measure the differences between the two. For example, in a simple two-stage system, we could generate the top-k list for both stages and somehow measure the similarity or difference between the two runs. We refer to this as a *reference list* comparison. There is now a large body of work on using reference lists for evaluation [13, 45, 46, 55]. In this work, we use *Maximized Effectiveness Difference* (MED) where the exact gain function used to compute the difference can depend on any utility-based evaluation metric, such as ERR, DCG, or RBP [46]. MED has the additional advantage that if partial judgments are available for any of the queries, the information can be used directly for the final comparison.

3 METHODOLOGY

Problem Definition. First, we define the problem we aim to solve. Given a query q, a series of re-ranking stages R, and a target evaluation metric M for the final stage, how can we predict both k and the processing algorithm A for the initial (bag-of-words) stage such that k, processing time t, and effectiveness loss L are minimized without requiring relevance judgments?

Labelling for Prediction without Relevance Annotations. We now turn to answering RQ1, by proposing a reference list model for learning efficiency parameters on a query-by-query basis. Since we wish to learn how to predict k (and other performance parameters) without relevance judgments, we employ a labelling algorithm based on the MED reference list approach. Given a *gold-standard* ranking for a query (as defined by an effective ranking model), and a corresponding bag-of-words candidate list, we iteratively measure the MED on an increasing prefix of the candidate list with respect to the gold-standard ranking. The goal is to find the shortest prefix of the candidate list that would not result in significant effectiveness loss after re-ranking. Algorithm 1 shows the psuedocode for this labelling algorithm. Once the optimal k is found, it can be used as a label for training. We note that our psuedocode is defined using a naïve (linear) approach for simplicity. In practice, a binary search can be conducted to yield the optimal k more efficiently. For our MED calculation, we use only MED_{RBP} with a small target threshold of $\epsilon = 0.001$ as we wish to aggressively minimize effectiveness loss. Clarke et al. [13] showed that other common utility-based metrics could also easily be used such as MED_{ERR} and MED_{DCG}, but we do not explore that option in this work.

Algorithm 1: Finds the smallest value of k such that effectiveness loss is minimized between a bag-of-words candidate list and a final stage re-ranked list for a given query.

Input : A bag-of-words run $\mathcal{D}^a$, a corresponding re-ranked list $\mathcal{D}^b$ (the gold-standard), and a desired MED threshold, ϵ.
Output : The smallest value of k such that $\text{MED}(\text{Top}k\text{Prefix}(\mathcal{D}^a, k), \mathcal{D}^b) < \epsilon$.

```
currentMED ← 1.0
k ← 0
candidateSet ← ∅
while currentMED > ε do
    k ← k + 1
    candidateSet.Append(DocAtRankk(D^a, k))
    currentMED ← MED(candidateSet, D^b)
end
return k
```

Generating Gold Standard Rankings. In order to accurately label parameter values using a reference list approach, we need a ground truth which represents an idealized last stage run over a large corpus of queries. This idealized last stage represents the trusted reference list for which all comparisons can be made. In order to build a competitive "last stage" reference list, we train a Risk-Sensitive LambdaMart model using the JForests library [23, 52], and 687 queries from the 2009 Million Query Track (MQ2009) query set, which have shallow relevance judgments. A set of 400 commonly used LtR features were used in the model, and significant effectiveness improvements were observed when testing the model using the 2009 ClueWeb09 Adhoc query set.

Following Clarke et al. [13], we also tested our entire prediction framework using the `uogTRMQdph40` run as a reference list, as it was one of the top scoring systems that returned results for all of the Million Query Track topics. We found that all of the results produced are comparable independent of the reference list used. The key point is that *any* reference list can be used as long as it is consistently "better" than the first stage method being evaluated. For example, the wins/ties/losses for our URisk model compared to the Okapi BM25 baseline with NDCG@10 is 33 (0.1793)/6/10

(0.0786), where the numbers in parentheses correspond to the average differences in score. Building the best reference list is an interesting problem in its own right, but beyond the scope of this work. In the interest of succinctness, we only report results from our own end-to-end system here.

Parameter Prediction using Regression. Recently, Culpepper et al. [16] described an effective approach of dynamically predicting k while minimizing effectiveness loss. The key idea was to use the reference list methodology described above to build ground truth labels to train a classifier. However, their approach has a few drawbacks. First, the cascade classifier they described is interesting but unconventional in that it requires multiple predictions to be made, depending on the final k. Fewer predictions are required for small k, but up to 8 independent predictions are required for large k. Secondly, the problem they describe is really a regression problem in practice. Using regression allows an exact k to be predicted instead of an approximate cutoff, which translates into fewer documents being re-ranked in later stages of the retrieval system.

Commonly, regression methods estimate the conditional expectation of a target dependent variable y given the independent variables (or features) $\mathbf{x}$. This implies that the method approximates the average value of the dependent variable when the independent variables are fixed. Given training data of the form $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)$ methods based on least squares try to optimize the loss function $L(\mathbf{x}, y) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\mathbf{x_i} - y_i)^2$, which results in a good estimator for the mean $E[y|\mathbf{x}]$.

So, the obvious way to reproduce their work is to use a similar feature set, and compute the exact k needed for each query that achieves a very small expected MED loss, say, $\epsilon < 0.001$, and use a random forest to produce the predictions. When we build this training set, one immediate problem becomes apparent – the ground truth labels do not follow a standard distribution, but an out-of-the-box regression algorithm *does*. Figure 2 shows three different distributions – the true distribution of k in the ground truth set (Oracle), the random forest prediction ($\text{RF}_{0.001}$), and a quantile regression prediction (QR_{τ}), which is described now.

A pitfall of standard regression methods is that they may become unstable under heavy-tailed distributions due to the dominant effects of outliers, or more precisely, when samples from the tail of the distribution have a strong influence on the mean. How to cope with this problem has been studied in the context of *robust estimation*. These estimators embody a family of methods designed to be more resilient to the data generation process by not following the underlying assumptions behind the regressor; in the context of least squares, this would be errors being uncorrelated and having the same variance.

One simple way of dealing with the outlier problem is *quantile regression*, which estimates either the conditional median or other quantiles of the response variable. If y has a cumulative distribution of $F_y(z) = p(y \leq z)$ then the τ-th quantile of y is given by $Q_y(\tau) = F_y^{-1} = \inf\{z : F_y(z) \geq \tau\}$. To learn a regressor that minimizes a τ value, we define the loss function $\xi_\tau(y) = y(\tau - \mathcal{I}\{y < 0\})$ where $\mathcal{I}\{\cdot\}$ is the indicator function. Therefore, τ-th quantile regression estimates the conditional τ-th quantile $F^{-1}_{y|\mathbf{x}}(\tau)$, or we want

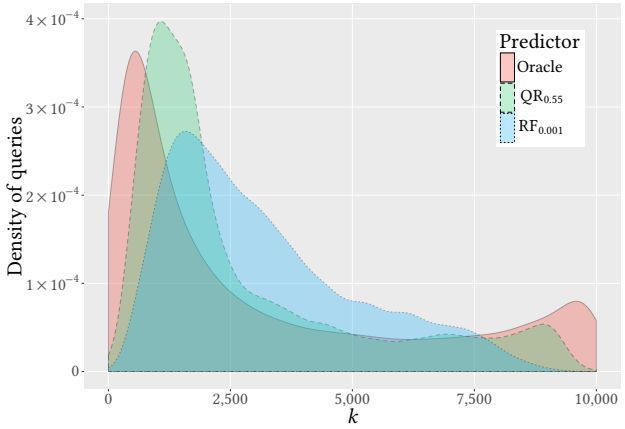

Figure 2: A comparison of the distributions of actual k versus predicted k when using a Random Forest regression and a Quantile Regression in first stage retrieval for the 26,959 queries from the MQ2009 TREC Task. Note that the Random Forest uses a training value of $\epsilon = 0.001$, whereas the best-fit distribution for the Quantile Regression was $\tau = 0.55$ for k.

an estimate $\hat{f}_\tau$ such that $p(y < \hat{f}_\tau(\mathbf{x})) = \tau$:

$$\hat{f}_\tau = \underset{f \in \mathcal{F}_\tau}{\text{argmin}} \sum_{i=1}^{n} \xi_\tau(y_i - f(\mathbf{x_i})) = \quad (1)$$

$$\underset{f \in \mathcal{F}_\tau}{\text{argmin}} \left[(1-\tau) \sum_{y_i < f(\mathbf{x_i})} |y_i - f(\mathbf{x_i})| + \tau \sum_{y_i \geq f(\mathbf{x_i})} |y_i - f(\mathbf{x_i})| \right], \quad (2)$$

where $\mathcal{F}_\tau$ is a predetermined class of functions.

A robust regression method is *random forests* (RF), which build several decision trees using attribute bagging. In a nutshell, the algorithm samples with replacement the training data B times and trains several decision trees f_b using only each portion of the data. The final prediction for an incoming new query is averaged from all the regressors $\hat{f} = \frac{1}{B}\sum_{i=1}^{B} f_B(\mathbf{x})$. Subsampling has the practical effect of decreasing the variance of the model, without increasing its complexity, given that even if the predictions of a single tree are highly sensitive to noise, the average of many trees is not (as long as the trees are not correlated). Bootstrapping achieves this effect by training each tree with a different randomized subsample.

When the individual trees f_b are learned, the building procedure creates tree nodes that branch data down the tree; in order to reduce the model variance, only a few features are candidates for splitting at each round. This mitigates the effect that happens when, if just a few features are very strong predictors for y, these features will be selected in many of the B trees, which become correlated.

We deploy the quantile regression within the same tree framework using gradient boosting regression trees (GBRT). In this case, each tree re-fits the training data using the residuals (gradients) of the training data with respect to the ξ_τ loss function, and a per-tree weight is calculated using line search. The final decision is a linear combination of the weighted prediction of the tree ensemble.

Parameter Prediction Features. For predicting the performance parameters, we used a similar set of features as Culpepper et al. [16].

These features are based on aggregating statistics for each postings list (such as maximum scores, harmonic/arithmetic mean/median scores, and so on) from a range of similarity functions, along with query specific features such as query length, max score of query terms, and many more. In addition to the TF·IDF, BM25 and query likelihood used by Culpepper et al. [16], we also build features using Bose-Einstein, DPH, and DFR similarity functions [1]. We also added the geometric mean as an aggregation statistic for each of these similarity functions. We used a total of 147 features for predicting parameters, and refer the reader to the work of Culpepper et al. [16] for a more detailed description of these features.

Experimental Setup. All experiments were executed on an idle 24-core Intel Xeon E5-2690 with 512 GB of RAM hosting RedHat RHEL v7.2. ATIRE [50] was used to parse and index the ClueWeb09B collection, which was stopped using the default Indri stoplist, and stemmed using an *s*-stemmer. Timings were conducted using publicly available implementations of Bmw[1] and Jass,[2] which use QMX compression [49, 51] and the BM25 scoring model. Each query is processed 5 times, and the average of the 5 runs is reported. Boxplots are standard Tukey plots, and diamonds in each box denote the mean value. For the prediction tasks, we use the 2009 Million Query Track queries. Single term queries were filtered from all test, train, and validation sets, as they can be answered trivially by taking the first k documents from the relevant postings list of the impact-ordered ISN. In addition, we filtered out queries which contained out-of-vocabulary terms, and the queries which were used to train the URisk gold-standard system, resulting in a set of 26,959 unique queries. For all predictions, queries were randomly assigned to 10 folds, and standard 10 fold cross validation was performed to produce the query predictions.

4 PRELIMINARY EXPERIMENTS

The improved approach to predicting k in first stage retrieval (Figure 2) is a promising first step to achieving efficient results without sacrificing effectiveness. However, assuming that the performance of Wand-based algorithms in the first stage is a function of k may not be correct in practice [14].

Tail-Latency in DaaT Algorithms. Crane et al. [14] showed that when using Wand and Bmw, outlier response times can occur at any k cutoff, making performance guarantees hard to enforce in production systems. The alternative to using Wand or Bmw in the first stage retrieval is to use a SaaT algorithm such as Jass. Unfortunately, this is not an entirely satisfactory answer either as most of the performance gains in Jass come from using aggressive early termination, which can hurt effectiveness when the number of documents that must be passed to the next stage must also be minimized. So, rank safety is yet another confounding factor. DaaT and SaaT processing algorithms can sacrifice effectiveness for efficiency by relaxing the rank-safety constraint. For example, Jass allows a parameter ρ to be set which bounds the maximum number of postings to score per query, and variants of Wand can use a parameter F which induces more aggressive skipping during

[1]http://github.com/JMMackenzie/Quant-BM-WAND
[2]http://github.com/lintool/JASS/

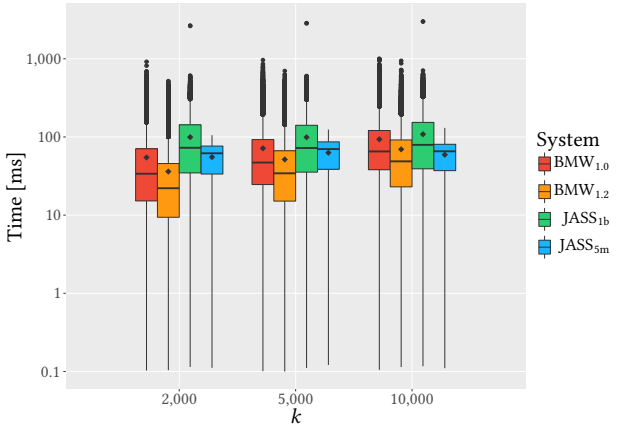

Figure 3: Efficiency comparison of the 26,959 queries from the MQ2009 TREC Task using both aggressive and rank-safe versions of Bmw and Jass. Subscripts denote the aggression parameters (F for Bmw and ρ for Jass).

postings list traversal. So, there is a trade-off between retrieval depth k and rank safety in a pure efficiency sense. This relationship was previously explored by Tonellotto et al. [48], who also used a query difficulty prediction framework to solve the problem. We build on this idea in this work, but also account for the fact that using only Wand-based algorithms can still result in high percentile latencies. We can see that boosting F alone does indeed make Bmw faster in Figure 3, but the high tail latency remains.

Our next task is to explore the likelihood of long-running queries when using the MQ2009 topic set. Crane et al. [14] performed a comparative analysis with the UQV [6] query set and the ClueWeb12B document collection with fixed values of k. We reproduce their work here across our own query set and fixed k values. Figure 3 shows the breakdown of all 26,959 queries across a number of fixed values of k, selected as appropriate sizes for an LtR system [36]. Similar to Crane et al., we observe that the exhaustive Bmw algorithm is superior to the exhaustive Jass algorithm, but the aggressive Jass traversal (with the recommended 10% heuristic) has a much lower tail latency. On the other hand, the aggressive Bmw traversal does improve the mean and median times, but does not adequately reduce the high percentile latency. Note that we selected the value for the heuristic, $F = 1.2$, based on other work that shows that more aggressive approaches result in reduced effectiveness [13]. It is also noteworthy that the exhaustive Bmw traversal has a faster median time than the aggressive Jass traversal when $k \leq 5{,}000$.

To further explore the relationship between the tail latency and the index traversal algorithm (RQ2), we do a simple overlap analysis on the slowest running 5% of queries for each algorithm. Table 1 shows the percentage of the tail latencies that overlap between each system, where $k = 2{,}000$. Exact Jass, exact Bmw and aggressive Bmw tend to share similar queries in the slowest running 5%. However, we note that the aggressive Jass traversal tends to share only a small percentage of the tail latencies that occur in the other systems.

In light of this new evidence, a pragmatic hypothesis emerges: Can we somehow combine the best properties of Jass and Bmw to create a hybrid approach that captures the best of both worlds?

	BMW$_{1.1}$	BMW$_{1.2}$	JASS$_{1b}$	JASS$_{5m}$
BMW$_{1.0}$	86.0	61.7	56.2	16.7
BMW$_{1.1}$	-	67.4	53.1	18.3
BMW$_{1.2}$	-	-	42.3	24.2
JASS$_{1b}$	-	-	-	8.0

Table 1: The percentage overlap of queries that fall in $95 \leq x \leq 100$ percentile efficiency band for k = 2,000. Clearly, making BMW more aggressive may improve timings, but outliers are still present. On the other hand, it is less common for JASS and BMW to have overlapping tail queries, especially when a non-exhaustive ρ value is used.

Algorithm 2: Candidate generation pipeline based on predicting k

Input : A query q, a regressor $\mathcal{R}_k$ that predicts the required k for q, a regressor $\mathcal{R}_\rho$ that predicts the required ρ for JASS up to a maximum ρ value $\rho_{\max}$, and a k-threshold T_k
Output : A set of candidate documents, C

```
C ← ∅
P_k ← R_k(q)
if P_k > T_k then
    P_ρ ← R_ρ(q)
    C ← ISN_JASS(q, P_k, P_ρ)
else
    C ← ISN_BMW(q, P_k)
end
return C
```

5 HYBRID ARCHITECTURE

The first major difference in our approach with respect to 'standard' systems is that we opt to build a hybrid architecture. Work on distributed IR has shown that an effective approach to scaling is to replicate popular indexes [18, 22, 28, 29]. Here, we assume that we can build ISNs that are optimized for different types of queries. In other words, when we build replicas, we may opt to build a document-ordered index (appropriate for DAAT traversal), or an impact-ordered index (appropriate for SAAT traversal). This idea is key to our novel framework: Selecting algorithm $a \in A$ actually refers to selecting an ISN to process the query which is configured to run algorithm a, and ISN selection is already a common problem in distributed search architectures [7, 27]. In practice, our "Stage-0" predictions would be performed by the resource selection process in a large scale distributed IR system.

Based on several observations about the relative performance of JASS and BMW, we are now in a position to describe a few different hybrid approaches to query processing, and to answer RQ2. Our goal is to limit the disadvantages of each traversal algorithm, and exploit the desirable properties. Several different variations were used in our preliminary experiments, and the best is shown here. The first step in the pipeline is to predict the k cutoff. If k is greater than the threshold T_k, then proceed to the JASS pipeline as outlined in Algorithm 2, If JASS is used, a prediction for ρ is made, but capped at $\rho_{\max}$, which allows us to achieve the desired performance guarantees. In our experiments, $\rho_{\max}$ = 5 million postings as this requires less than 200 ms on our current hardware configuration, and does not result in a significant loss of effectiveness for early precision metrics across ClueWeb09B [32]. The remaining queries are processed using BMW with rank-safety. We also experimented with a pipeline which predicted the run-time of the incoming query (as well as k), and used JASS if the predicted run-time was greater than 200ms, but its performance is comparable to the simpler model, so we do not report it in the interest of space.

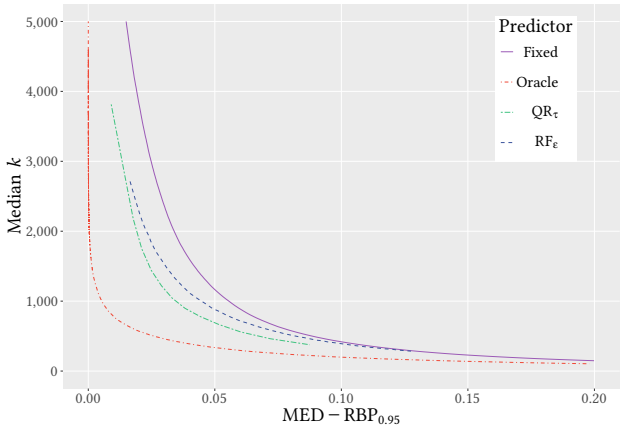

Figure 4: MED$_{\text{RBP}}$ versus median k for all ϵ thresholds between 0.001 and 0.200 when using a Random Forest regression, and for all τ values between 0.10 and 0.75 with ϵ = 0.001 for Quantile Regression, in first stage retrieval for the 26,959 queries from the MQ2009 TREC Task. Note that the Quantile Regression clearly improves the median k (compared with Random Forests) without negatively affecting the mean k.

6 EXPERIMENTS

We now look at the various predictions that are necessary to achieve our performance goals. Our performance requirement for effectiveness is a MED score that is low enough to result in no measurable effectiveness difference for the target metric. Our performance requirement for efficiency targets reducing high percentile tail latency in the candidate generation stage, while also attempting to reduce the size of both k and ρ in a query dependent way.

Predicting k. First, we validate that our new approach to k prediction using quantile regression is effective. Using our newly devised regression technique, we can compare the efficiency and effectiveness trade-offs between the size of the candidate retrieval set k, and the expected effectiveness loss MED$_{\text{RBP}}$. Figure 4 shows the predictive power of a random forest (RF$_\varepsilon$) and quantile regression (QR$_\tau$) when compared to the oracle results for the MED$_{\text{RBP}}$ target $\epsilon = [0.001, 0.20]$ and to using a fixed cutoff for all queries. Since the distribution of the true k values is skewed for the queries as shown in Figure 2, presenting the results using the median more accurately captures the trade-offs.

Predicting ρ. Based on the lessons learned when attempting to build a robust prediction framework for k, we now turn our attention to the aggressiveness parameter ρ in JASS. Previous work has

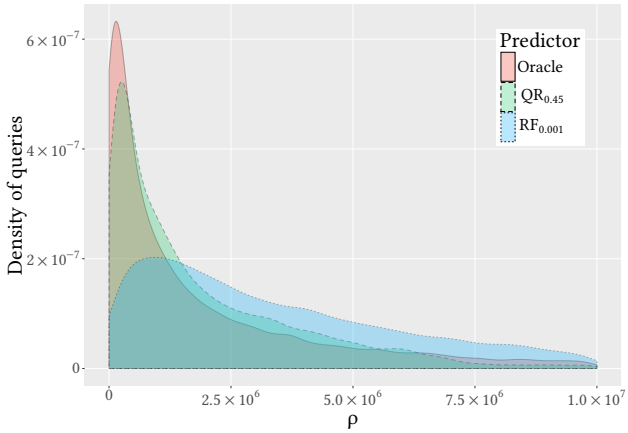

Figure 5: A comparison of the distributions the actual ρ vs the predicted ρ when using a Random Forest regression and a Quantile Regression in first stage retrieval for the 26,959 queries from the MQ2009 TREC Task. The best-fit distribution for the Quantile Regression was $\tau = 0.45$ for ρ.

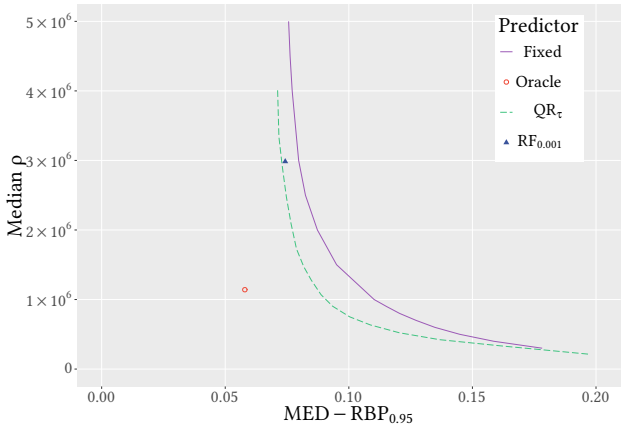

Figure 6: MED_{RBP} versus median ρ. The RF model was trained to target a MED_{RBP} of 0.001, and the QR model plots the various quantile points from $\tau = 0.15$ to $\tau = 0.75$. Quantile Regression and Random Forests behave similarly with respect to the median ρ, but QR is still preferred as the final predicted ρ distribution fits better with the idealized results as shown in Figure 5.

shown that using an exhaustive ρ results in effective top-k retrieval, however, using a heuristic ρ can give similar effectiveness, yet much more efficient retrieval [31, 32]. The recommended heuristic value of ρ is 10% of the size of the collection [32], which is around 5 million for the ClueWeb09B collection. Figure 5 shows the distribution of ρ values required to when targeting a $\text{MED}_{\text{RBP}} < 0.001$, which aggressively targets no measurable difference in the results lists between exhaustive and aggressive Jass traversals. Clearly, the majority of the distribution lies well to the lower side of the 10% heuristic value. This motivates us to predict ρ on a query-by-query basis. Again, we deploy both a Random Forest and a Gradient Quantile Regression method as the distribution of ρ is skewed, and build a suitable `find_rho` algorithm, similar to `find_k` (Algorithm 1), to label the prediction training data.

Figure 6 shows the median predicted ρ values compared with the fixed and oracle. Both the QR and RF regression methods manage to improve on the fixed ρ median. Note that when measuring the MED_{RBP} for this experiment (and subsequently, training the value of ρ), the k utilized was the optimal value of k from the previous experiment. The reason for using this k is that we must fix k, otherwise our effectiveness scores may change as a result of k, not just ρ. Indeed, this setting of k also allows us to find the true optimal MED_{RBP} for Jass, denoted by the oracle point in Figure 6.

Putting It All Together. Here, we show that by combining all of our predictions into hybrid first-stage retrieval systems, outlined in Algorithm 2, we can achieve effectiveness equal to a fixed parameter system, while controlling various early and late-stage efficiency parameters, thus answering RQ3.

Figure 7 shows the performance for 4 different MED_{RBP} cut-offs: 0.04, 0.06, 0.08, and 0.10. We present Jass_{1b}, Jass_{5m} and $\text{Bmw}_{1.0}$, which refer to using a fixed k and ρ for all queries. For these baselines, k was selected such that the mean MED value was equivalent to the target epsilon. We also report the results of the hybrid system based on Algorithm 2 (Hybrid-k-τ), which uses quantile regression

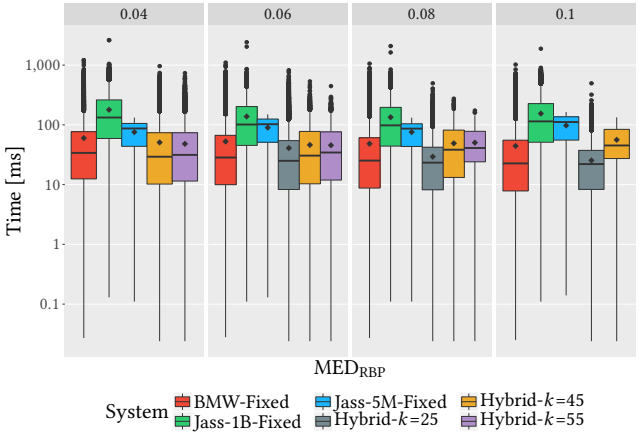

Figure 7: The response time for each system for different bands of MED_{RBP} in the candidate generation stage. Different configurations of the hybrid approach can bridge the gap between the Bmw and Jass systems across different MED targets, allowing for more finely-grained trade-offs.

for per-query predictions of both k and ρ. Note that the labels denote the τ value which was used to predict k using the QR predictor. For the ρ prediction, we only show results using the QR predictor with $\tau = 45$, although we can create more finely grained trade-offs by employing other values of τ. Additionally, Table 2 shows the median k, as well as the time characteristics for the systems presented in Figure 7.

Our results show that our hybrid system both outperform the equivalent fixed Bmw or Jass approaches for the given MED targets. For example, with a target of $\text{MED}_{\text{RBP}} = 0.02$, our hybrid systems can achieve a mean and median query response time around 10 ms, and 5 ms below the best fixed system, respectively, while also requiring a lower median k, which results in additional downstream

System	Median k	Mean time	Median time	95th perc. latency
MED-RBP$_{0.95}$ = 0.04				
Bmw	1598	60.2	34.0	205.6
Jass$_{1b}$	1598	178.7	132.7	473.0
Jass$_{5m}$	2970	75.6	87.1	**114.1**
Hybrid-k = 25	-	-	-	-
Hybrid-k = 45	**1148**	50.8	**29.3**	149.9
Hybrid-k = 55	1660	**48.1**	31.4	126.9
MED-RBP$_{0.95}$ = 0.06				
Bmw	889	52.7	28.4	183.4
Jass$_{1b}$	889	131.2	101.4	368.7
Jass$_{5m}$	1913	89.9	102.5	139.7
Hybrid-k = 25	**631**	**40.9**	**25.0**	132.8
Hybrid-k = 45	1148	46.2	30.6	121.9
Hybrid-k = 55	1659	45.6	34.5	**110.0**
MED-RBP$_{0.95}$ = 0.08				
Bmw	**578**	48.3	25.2	171.2
Jass$_{1b}$	**578**	134.1	98.1	357.4
Jass$_{5m}$	1324	75.6	86.6	119.8
Hybrid-k = 25	631	**29.3**	**23.3**	**73.4**
Hybrid-k = 45	1148	49.1	38.4	120.9
Hybrid-k = 55	1659	50.3	41.0	109.3
MED-RBP$_{0.95}$ = 0.10				
Bmw	**419**	44.3	22.6	158.2
Jass$_{1b}$	**419**	154.4	114.4	409.4
Jass$_{5m}$	959	97.7	111.5	152.2
Hybrid-k = 25	631	**25.4**	**22.1**	**59.8**
Hybrid-k = 45	1148	56.2	45.2	120.7
Hybrid-k = 55	-	-	-	-

Table 2: Summary statistics for k and time. Each sub-table corresponds to a section of Figure 7, and the best values are bold. Our Hybrid approaches generally outperform the fixed parameter systems with respect to all time dimensions, while also sometimes improving the median k value.

efficiency. Although the fixed Jass$_{5m}$ system outperforms our hybrids in reducing the tail latency, it must retrieve a larger number of documents to achieve the same effectiveness target, which has negative implications on the efficiency of the following stages. As we relax the MED target, the hybrid systems tend to require a slightly larger median k than the fixed parameter systems, but tend to perform much more efficiently in the early-stage efficiency dimensions. For example, the Hybrid-k=25 system has a mean, median and 95th percentile latency that is 18.9, 0.5, and 98.4 ms faster than the best fixed system, at the cost of requiring 212 more documents per query (based on the median) when targeting a MED$_{\text{RBP}}$ of 0.10. Note that we do not consider the time required to make our predictions, but this cost is an order of magnitude less than the run times being achieved. Recent work using similar models show a prediction overhead of < 0.75 ms per prediction [25], and this approach can be directly applied here.

Validating Robustness. As a final test of robustness, we run both our hybrid and fixed systems across the 50 (unseen) TREC 2009 Web Track queries. These queries were held out from the train and test procedures reported in earlier sections, and were used to validate the URisk model (our baseline here). Since these queries only have judgments to depth 12, we report NDCG@10, ERR@10 and RBP$_{0.80}$ [33]. For the hybrid systems, we used the same prediction configuration that was used in the tasks from Figure 7 and Table 2.

System	MED-RBP$_{0.95}$	NDCG@10	ERR@10	RBP p = 0.80
BM25	-	0.2055	0.0957	0.3070 (0.1734)
URisk-ideal	-	0.3102	0.1354	0.4250 (0.2195)
Fixed (Bmw/Jass$_{1b}$)	0.04	0.3027	0.1305	0.4176 (0.2240)
Fixed (Jass$_{5m}$)	0.04	0.3028	0.1291	0.4005 (0.2503)
Hybrid	0.04	0.3107	0.1357	0.4221 (0.2220)
Fixed (Bmw/Jass$_{1b}$)	0.06	0.3052	0.1320	0.4215 (0.2214)
Fixed (Jass$_{5m}$)	0.06	0.3057	0.1299	0.4028 (0.2531)
Hybrid	0.06	0.3139	0.1361	0.4265 (0.2320)
Fixed (Bmw/Jass$_{1b}$)	0.08	0.3082	0.1336	0.4226 (0.2161)
Fixed (Jass$_{5m}$)	0.08	0.3047	0.1309	0.4066 (0.2484)
Hybrid	0.08	0.3137	0.1353	0.4247 (0.2312)
Fixed (Bmw/Jass$_{1b}$)	0.10	0.2945	0.1270	0.4110 (0.2204)
Fixed (Jass$_{5m}$)	0.10	0.3090	0.1321	0.4114 (0.2449)
Hybrid	0.10	0.3137	0.1352	0.4247 (0.2312)

Table 3: Effectiveness measurements taken across the held-out query set. No statistical significance was measured between the hybrid systems with respect to the ideal system (URisk-ideal), using the two one-sided test with $p < 0.05$. We used the Hybrid-k = 45 system for this comparison, although the results are not significantly different to the other Hybrid parameterizations.

Table 3 shows the effectiveness measurements. Remarkably, our hybrid systems have no loss in effectiveness when computing an complete end-to-end run. We confirm this observation using a two one-sided test [44] of equivalence (Tost). For each Tost, we set the acceptable range of inequality to ±1%. We found that the ideal system is not statistically significantly different than our hybrid systems, with $p < 0.001$. We also observe that our system outperforms the fixed-parameter equivalents across these 50 validation queries, although no statistical significance was detected.

7 CONCLUSION

We presented and validated a unified framework to predict a wide range of performance-sensitive parameters for early-stage candidate retrieval systems using MED [46] and reference lists as guides for training. Preliminary experiments show that the DaaT Bmw approach is efficient but suffers from a comparatively large tail latency, while the SaaT Jass algorithm does not. A hybrid system based on this predictive framework was shown to minimize effectiveness loss while also minimizing query-latency across the candidate generation stage of the pipeline, providing improved trade-offs with respect to a standard, fixed-parameter system. Future work will involve exploring the design implications of the hybrid ISN approach, to quantify the trade-offs involved with the number of replicas required across the various hybrid parameterizations.

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP170102231), the Natural Sciences and Engineering Research Council of Canada, an Australian Government Research Training Program Scholarship, and a grant from the Mozilla Foundation. We thank Luke Gallagher for providing support with the Learning-to-Rank framework.

REFERENCES

[1] G. Amati and C. J. Van Rijsbergen. 2002. Probabilistic Models of Information Retrieval Based on Measuring the Divergence from Randomness. 20, 4 (2002), 357–389.

[2] V. N. Anh, O. de Kretser, and A. Moffat. 2001. Vector-Space Ranking with Effective Early Termination. In *Proc. SIGIR*. 35–42.

[3] N. Asadi and J. Lin. 2013. Document Vector Representations for Feature Extraction in Multi-Stage Document Ranking. *Inf. Retr.* 16, 6 (2013), 747–768.

[4] N. Asadi and J. Lin. 2013. Effectiveness/Efficiency Tradeoffs for Candidate Generation in Multi-Stage Retrieval Architectures. In *Proc. SIGIR*. 997–1000.

[5] N. Asadi, J. Lin, and A. P. De Vries. 2014. Runtime Optimizations for Tree-Based Machine Learning Models. *Trans. on Know. and Data Eng.* 26, 9 (2014), 2281–2292.

[6] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. 2016. UQV100: A Test Collection with Query Variability. In *Proc. SIGIR*. 725–728.

[7] D. Broccolo, C. Macdonald, O. Salvatore, I. Ounis, R. Perego, F. Silvestri, and N. Tonellotto. 2013. Load-sensitive Selective Pruning for Distributed Search. In *Proc. CIKM*. 379–388.

[8] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Y. Zien. 2003. Efficient Query Evaluation using a Two-Level Retrieval Process. In *Proc. CIKM*. 426–434.

[9] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. 2010. Early Exit Optimizations for Additive Machine Learned Ranking Systems.. In *Proc. WSDM*. 411–420.

[10] K. Chakrabarti, S. Chaudhuri, and V. Ganti. 2011. Interval-based Pruning for Top-k Processing over Compressed Lists. In *Proc. ICDE*. 709–720.

[11] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. 2009. Expected Reciprocal Rank for Graded Relevance. In *Proc. CIKM*. 621–630.

[12] R-C. Cheng, L. Gallagher, R. Blanco, and J. S. Culpepper. 2017. Efficient Cost-Aware Cascade Ranking in Multi-Stage Retrieval. In *Proc. SIGIR*. 445–454.

[13] C. L. A. Clarke, J. S. Culpepper, and A. Moffat. 2016. Assessing Efficiency–Effectiveness Tradeoffs in Multi-Stage Retrieval Systems without using Relevance Judgments. *Inf. Retr.* 19, 4 (2016), 351–377.

[14] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. 2017. A Comparison of Document-at-a-Time and Score-at-a-Time Query Evaluation. In *Proc. WSDM*. 201–210.

[15] M. Crane, A. Trotman, and R. O'Keefe. 2013. Maintaining Discriminatory Power in Quantized Indexes. In *Proc. CIKM*. 1221–1224.

[16] J. S. Culpepper, C. L. A. Clarke, and J. Lin. 2016. Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems. In *Proc. ADCS*. 17–24.

[17] V. Dang, M. Bendersky, and W. B. Croft. 2013. Two-Stage Learning to Rank for Information Retrieval. In *Proc. ECIR*. 423–434.

[18] J. Dean and L. A. Barroso. 2013. The Tail at Scale. *Comm. ACM* 56, 2 (2013), 74–80.

[19] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. 2013. Optimizing Top-k Document Retrieval Strategies for Block-Max Indexes. In *Proc. WSDM*. 113–122.

[20] S. Ding and T. Suel. 2011. Faster Top-k Document Retrieval Using Block-Max Indexes. In *Proc. SIGIR*. 993–1002.

[21] M. Fontoura, V. Josifovski, J. Liu, S. Venkatesan, X. Zhu, and J. Zien. 2011. Evaluation Strategies for Top-k Queries over Memory-resident Inverted Indexes. *Proc. VLDB* 4, 12 (2011), 1213–1224.

[22] G. Francès, X. Bai, B. B. Cambazoglu, and R. Baeza-Yates. 2014. Improving the Efficiency of Multi-site Web Search Engines. In *Proc. WSDM*. 3–12.

[23] Y. Ganjisaffar, R. Caruana, and C. Lopes. 2011. Bagging Gradient-Boosted Trees for High Precision, Low Variance Ranking Models. In *Proc. SIGIR*. 85–94.

[24] S-W. Hwang, K. Saehoon, Y. He, S. Elnikety, and S. Choi. 2016. Prediction and Predictability for Search Query Acceleration. *ACM Trans. Web* 10, 3 (Aug. 2016), 19.1–19.28.

[25] M. Jeon, S. Kim, S-W. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner. 2014. Predictive Parallelization: Taming Tail Latencies in Web Search. In *Proc. SIGIR*. 253–262.

[26] X. Jin, T. Yang, and X. Tang. 2016. A Comparison of Cache Blocking Methods for Fast Execution of Ensemble-based Score Computation. In *Proc. SIGIR*. 629–638.

[27] S. Kim, Y. He, S-W. Hwang, S. Elnikety, and S. Choi. 2015. Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search. In *Proc. WSDM*. 7–16.

[28] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2016. Efficient Distributed Selective Search. *Inf. Retr.* (2016), 1–32.

[29] Y. Kim, J. Callan, J. S. Culpepper, and A. Moffat. 2016. Load-Balancing in Distributed Selective Search. In *Proc. SIGIR*. 905–908.

[30] D. Lemire and L. Boytsov. 2015. Decoding Billions of Integers Per Second through Vectorization. *Soft. Prac. & Exp.* 45, 1 (2015), 1–29.

[31] J. Lin, M. Crane, A. Trotman, J. Callan, I. Chattopadhyaya, J. Foley, G. Ingersoll, C. Macdonald, and S. Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. In *Proc. ECIR*. 408–420.

[32] J. Lin and A. Trotman. 2015. Anytime Ranking for Impact-Ordered Indexes. In *Proc. ICTIR*. 301–304.

[33] X. Lu, A. Moffat, and J. S. Culpepper. 2016. The Effect of Pooling and Evaluation Depth on IR Metrics. *Inf. Retr.* 19, 4 (2016), 416–445.

[34] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. 2015. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. In *Proc. SIGIR*. 73–82.

[35] C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. 2016. Exploiting CPU SIMD Extensions to Speed-up Document Scoring with Tree Ensembles. In *Proc. SIGIR*. 833–836.

[36] C. Macdonald, R. L. T. Santos, and I. Ounis. 2013. The Whens and Hows of Learning to Rank for Web Search. *Inf. Retr.* 16, 5 (2013), 584–628.

[37] C. Macdonald, R. L. T. Santos, I. Ounis, and B. He. 2013. About Learning Models with Multiple Query-Dependent Features. *ACM Trans. Information Systems* 31, 3 (2013), 11.

[38] J. Mackenzie, F. M. Choudhury, and J. S. Culpepper. 2015. Efficient Location-aware Web Search. In *Proc. ADCS*. 4.1–4.8.

[39] A. Moffat and J. Zobel. 2008. Rank-Biased Precision for Measurement of Retrieval Effectiveness. *ACM Trans. Information Systems* 27, 1 (2008), 2.1–2.27.

[40] J. Pedersen. 2010. Query Understanding at Bing. *Invited talk, SIGIR* (2010).

[41] M. Petri, J. S. Culpepper, and A. Moffat. 2013. Exploring the Magic of WAND. In *Proc. ADCS*. 58–65.

[42] M. Petri, A. Moffat, and J. S. Culpepper. 2014. Score-safe Term Dependency Processing with Hybrid Indexes. In *Proc. SIGIR*. 899–902.

[43] C. Rossi, E. S. de Moura, A. L. Carvalho, and A. S. da Silva. 2013. Fast Document-at-a-time Query Processing Using Two-tier Indexes. In *Proc. SIGIR*. 183–192.

[44] D. J. Schuirmann. 1987. A Comparison of the Two One-Sided Tests Procedure and the Power Approach for Assessing the Equivalence of Average Bioavailability. *J. Pharmacokinetics and Biopharmaceutics* 15, 6 (1987), 657–680.

[45] A. Shtok, O. Kurland, and D. Carmel. 2016. Query Performance Prediction Using Reference Lists. *ACM Trans. Information Systems* 34, 4 (2016), 19.1–19.34.

[46] L. Tan and C. L. A. Clarke. 2015. A Family of Rank Similarity Measures Based on Maximized Effectiveness Difference. *Trans. on Know. and Data Eng.* 27, 11 (2015), 2865–2877.

[47] N. Tonellotto, C. Macdonald, and I. Ounis. 2011. Effect of Different Docid Orderings on Dynamic Pruning Retrieval Strategies. In *Proc. SIGIR*. 1179–1180.

[48] N. Tonellotto, C. Macdonald, and I. Ounis. 2013. Efficient and Effective Retrieval using Selective Pruning. In *Proc. WSDM*. 63–72.

[49] A. Trotman. 2014. Compression, SIMD, and Postings Lists. In *Proc. ADCS*. 50.50–50.57.

[50] A. Trotman, X-F. Jia, and M. Crane. 2012. Towards an Efficient and Effective Search Engine. In *Wkshp. Open Source IR*. 40–47.

[51] A. Trotman and J. Lin. 2016. In Vacuo and In Situ Evaluation of SIMD Codecs. In *Proc. ADCS*. 1–8.

[52] L. Wang, P. N. Bennett, and K. Collins-Thompson. 2012. Robust Ranking Models via Risk-sensitive Optimization. In *Proc. SIGIR*. 761–770.

[53] L. Wang, J. Lin, and D. Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *Proc. SIGIR*. 105–114.

[54] Q. Wang, C. Dimpoloulos, and T. Suel. 2016. Fast First-Phase Candidate Generation for Cascading Rankers. In *Proc. SIGIR*. 295–304.

[55] W. Webber, A. Moffat, and J. Zobel. 2010. A Similarity Measure for Indefinite Rankings. *ACM Trans. Information Systems* 28, 4 (2010), 20.1–20.38.

[56] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. 2014. Classifier Cascades and Trees for Minimizing Feature Evaluation Cost. *J. of Machine Learning Research* 15 (2014), 2113–2144.

[57] J-M. Yun, Y. He, S. Elnikety, and S. Ren. 2015. Optimal Aggregation Policy for Reducing Tail Latency of Web Search. In *Proc. SIGIR*. 63–72.

[58] J. Zhang, X. Long, and T. Suel. 2008. Performance of Compressed Inverted List Caching in Search Engines. In *Proc. WWW*. 387–396.

[59] J. Zobel and A. Moffat. 2006. Inverted Files for Text Search Engines. *ACM Comp. Surv.* 38, 2 (2006), 6.1–6.56.