Application of Dynamic Pruning to Field-Based Retrieval Models

ABSTRACT

We introduce a method for reformulating field-based retrieval models to significantly improve execution efficiency over standard model implementations. Field-based retrieval models have shown significant improvements over models without field awareness. Field-based models utilize the document structure to improve retrieval results, representing fields or tagged content as micro-documents within the larger document. However processing field information separately in these retrieval models often causes a significant increase in computational cost over their field-less counterparts. Additionally, field-based retrieval models in their original formulation do not easily lend themselves to dynamic pruning via algorithms such as Maxscore and WAND.

To mitigate these problems we introduce the idea of a δ-function — the reformulation of a given field-based model that allows iterative evaluation of a document. The δ-function form is completely score-safe for all documents, transitively making it rank-safe. We derive δ-functions for the PRMS, BM25F, and PL2F field-based retrieval models, showing how these formulations allow fine-grained dynamic pruning during query evaluation.

Our experiments show that using the δ-function formulation results in a significant increase in query efficiency over both unmodified and Maxscore-pruned test collections. Using a collection of documents from the OpenLibrary, we also show that the δ-function better scales with increasing field size than Maxscore.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process,retrieval models*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*performance evaluation*

General Terms

Algorithms, Experimentation, Performance

SIGIR '12 Portland, Oregon USA

Keywords

Dynamic Optimization, Pruning, Field Retrieval Models

1. INTRODUCTION

The vast majority of documents today have some form of structure. Large documents such as books can often have over a dozen fields of metadata or specific structure. Web documents have had their link content and title information heavily exploited to improve web retrieval. All emails are sent with sender, recipient, and subject information that can be used to better retrieve relevant emails. Even tweets and text messages come with metadata fields such as timestamps and sender/recipient information. As a simple example, imagine we would like to search for "a New York Times article that discusses the role of the United States in aiding Haiti after the 2010 earthquake". Based on this information need, we may form a query that looks like the following: `united states haiti earthquake 2010 aid`. However if we can use information in certain fields of the document, we could be more specific:

```
keywords: haiti earthquake aid
keyphrases: "united states"
publisher: "new york times"
timeframe: 2010
```

In addition to the typical keyword and phrase description, we also specify who we would like the publisher to be, and about what time we expect the article to appear. Field-based retrieval models often utilize the information in fields like "publisher" or "timeframe" by incorporating weighted scores of these fields into the final document score. Many of these field models are natural mathematical extensions of well-known retrieval models, but by using this additional information these models often outperform their field-less counterparts. While the mathematical formulation of such models is often straightforward, transferring these models to implementation can result in inefficient execution, and techniques for improving their efficiency, such as dynamic pruning, cannot easily be applied to these implementations without ad-hoc workarounds.

In this work, we present a model reformulation technique that produces a *δ-function*. The δ-function form allows for iterative computation of document scores; this in turn allows current dynamic pruning methods such as Maxscore and WAND to be used to their full potential. To demonstrate the use of δ-functions over models too complex for existing dynamic pruning techniques to operate effectively, we consider field-based extensions to 3 popular retrieval models today:

the probabilistic retrieval model for semi-structured data (PRMS), which is a field-based extension to the language model family; BM25F, which provides field-aware semantics for the Okapi BM25 model; and PL2F, which extends the Divergence from Randomness PL2 model to use fields by creating composite term scores based on fields. By deriving the δ-functions of these three models, we can use them to then apply standard dynamic optimization techniques to more aggressively prune during the scoring process. Our experiments show that this improved pruning can provide substantial gains over the standard processing model and standard Maxscore algorithm applied to the original model.

This paper proceeds as follows. In Section 2 we discuss related work, including more detail on prior dynamic pruning techniques and the three retrieval models under consideration. In Section 3 we introduce the general concept of a δ-function and perform a simple derivation of a δ-function over a query-likelihood model. In Section 4 we derive δ-functions for the three target models. Section 5 describes the setup and evaluation of all experiments conducted. We discuss results in Section 6, and conclude with a discussion in Section 7.

2. RELATED WORK

The work discussed in this paper draws from two separate research areas in IR: retrieval models that leverage semi-structured documents that contain fields, and query-time (also known as 'dynamic') optimizations, which lower the time it takes to return a scored set of documents by changing the score processing model. We review these two areas separately, then discuss how these two areas come together for the purposes of this paper.

2.1 Field-Based Retrieval Models

There are many ways to model structure in documents, and how to use them during retrieval. The Database (DB) community specializes in the case where documents are completely structured (i.e. every piece of data falls into a specific, well-defined field). Even considering semi-structured data, there has been research that bridges work between the DB and IR community, such as similarity joins [4]. Our focus here is on ranked retrieval models, so an in-depth review of DB-style retrieval models is beyond the scope of this paper.

Considering retrieval models that retrurn results as a ranking as in a typical IR system, numerous researchers have spent years working on constructing usable digital libraries [6, 7, 16], where the elements in the collection carry a considerable amount of structured metadata. The Initiative for the evaluation of XML has engaged researchers in search over XML documents for several years, spurring research over documents with a hierarchical structuring of fields [8]. For the scope of this work, we focus on three recent probabilistic ranking retrieval models that have shown promise as natural extensions of several well-known IR retrieval models. We describe these extensions below.

The PRMS model was first developed by Kim and Croft in order to improve search over desktop-type collections[9]. The PRMS model uses inferred field-mapping probabilities to weight the importance of each term/field pair in a given query. This is in contrast to the mixture of language-models (MLM) approach first proposed by Ogilvie and Callan [12], where the component model weights are externally parameterized. For the purposes of this work, we may view PRMS and MLM as the same formulation; therefore the δ-function derivation of PRMS applies to MLM without modication.

The BM25F model was first developed by Robertson et al., as a response to many models at the time linearly combining scores [13]. The authors showed that when a retrieval model uses a saturating term scoring function (such as BM25), scoring the fields independently results in the term appearing novel to each field. As the first occurrence of a term provides more confidence then subsequent occcurrences, scoring fields independently overweights the importance of a term. Instead, the authors combine at the term frequency level, then apply the scoring function over the combined frequencies. This to a much smoother rise in scoring as a function of term frequency across fields. We follow the BM25F formulation put forth by the same group of researchers for the TREC 2004 HARD Track [17].

PL2F was developed as a field-aware extension for the PL2 variant of the DFR framework. The term/field score contributions are linearly combined to make a "pseudo-term" score, and this composite score is then used in the standard PL2 score combination model. The PL2F model implemented for the experiments here are from the dsecriptions provided by Macdonald et al. for the TREC 2005 Enterprise Track [11].

2.2 Query-Time Optimization

There have been numerous contributions to query-time optimizations; we limit our review only to selected contributions that apply to the current experimental setting.

Turtle and Flood first describe the Maxscore algorithm after a review of the two major index organizations at the time: document-at-a-time (daat) and term-at-a-time (taat) evaluation [15]. The authors show that Maxscore can operate over both daat and taat, provided the index model supports skips over postings within a given term posting list, and the model can provide estimates of the upper bound that may be produced by a posting list. We call a function which provides such an estimate an *upper-bound estimator* (UBE).

Broder et al. described a two-pass algorithm over daat indexes they dubbed WAND, which reduces the number of postings to decode by first considering any given query to be a strictly conjunctive query, then relaxing that constraint as necessary to complete scoring [1]. Like Maxscore, WAND relies on upper-bound estimators to operate efficiently. The authors describe several UBEs for unigrams and bigrams, however they admit their estimators could be improved upon.

In recent work, Macdonald et al. define a new UBE they call max_{tf}. They derive max_{tf} for Language Models, BM25, and DLH13, by viewing the estimator as a constrained maximization problem (CMP) over the space of term-frequencies and document lengths [10]. We follow their approach here to verify that the estimators used here are reasonable and admissible heuristics for field-based models. We now introduce the terminology used in the remainder of the paper.

2.2.1 Terminology

Let D denote the set of documents and $|D|$ the number of the documents in the collection. C denotes the multiset of term occurrences in the whole collection. $|C|$ is then the number of tokens in the collection. For all formulae, we

assume $d \in D$, $t \in C$. A query Q is composed of one or more terms. We assume $|Q| = n : Q = q_1 ... q_n$. For a given collection D, we assume there is a set of m specified fields: $F = \{f_1 \dots f_m\}$. We assume that we can access individual posting lists for each term/field combination, and we use $c_{t_i f_j d}$ to denote the number of times term i occurs in in field j in document d, and $s_{t_i f_j d}$ is the score for term t_i in field f_j in d (e.g. the score of `york` in the `publisher` field in document 37).

For a given retrieval model M, our desire is to construct δ_M, which is the δ-function specific to M. For the scope of this paper, we can think of M as a function that produces a score when given Q and $d \in D$, which we denote as S_{Qd}. We denote a document score using S, and a partial (i.e. component) score for d using lower case s. In all cases, when it is umambiguous we drop the d subscript for brevity: $s_{t_i f_j d} \rightarrow s_{t_i f_j}$.

2.2.2 On Potentials

We assume all models are capable of generating an estimate (also called a *potential*) for $s_{t_i f_j}$, the score of a term t_i in field f_j. This potential, denoted $\phi_{t_i f_j}$, 1) must be document-independent, and 2) must never under-estimate the maximum score for the term-field pairing. Any estimator that meets the two above criteria we say is *admissible*[1]. The ideal admissible estimate is the supremum of the set of scores for the term, however if the scores are computed at retrieval time it is not practical to compute this value at index time, and we assume this to be the case. All of the UBEs discussed above meet the admissibility criteria. However the UBEs discussed so far are specific to term-level posting lists, whereas we require admissible estimators for term/field posting lists. When necessary, we derive a reasonable and admissible UBE for a term/field scoring function. We denote the composite potential ϕ_{t_i} as the potential function for the entirety of term t_i. To simplify reading, we may drop the subscript for the term or field (e.g. $\phi_{t_i} \rightarrow \phi_t$) if it is held invariant for the formula. Finally, we denote ϕ as the potential of a query Q, which is the best score possible for any document to receive for query Q. Although ϕ depends on both Q and M, we omit these symbols as they can always be assumed in the following formulae.

Our discussion will involve substitutions of the term/field potentials with actual scores. We denote the substitution of quantity $\phi_{t_i f_j}$ with $s_{t_i f_j}$ in ϕ as $\phi[\phi_{t_i f_j} / s_{t_i f_j}]$. For generality, when we want to indicate a quantity that has some number of substitutions (including zero substitutions), we modify the symbol with a caret (e.g. $\phi \rightarrow \hat{\phi}$). When appropriate to quanify the number of substitutions made so far, we superscript the number of substitutions so far (e.g. $\hat{\phi}^3$ indicates 3 substitutions from the original ϕ). We now describe the Maxscore and WAND optimization algorithms in detail, and discuss why their use is limited over field-based models.

2.2.3 Maxscore and WAND

For a query Q, Maxscore makes use of ϕ_t, the admissible UBE for each term. The algorithm also maintains an R value, which is the lowest score in the list of scored candidates so far. If the number of documents scored so far is less than k, the desired number of ranked documents to return, then $R = -\infty$. At the beginning of scoring a candidate document d, it is assumed to have the maximum possible score ϕ. As each term in evaluated for that document, its potential is replaced by its actual score: $\hat{\phi} \leftarrow \hat{\phi} + s_t - \phi_t$. Note that these successive replacements act as a monotonically decreasing estimate of the score for d. Therefore after this replacement, if $\hat{\phi} < R$, then we know d cannot make it into the final ranked list, and we stop scoring d. Therefore Maxscore looks to short-circuit score calculation at the term level.

Weak-AND, or WAND for short, was proposed by Broder et al. as a way to fully skip scoring documents using a two-level processing mechanism [1]. Like Maxscore, WAND depends on admissible UBEs of each term. When considering a candidate d, WAND first uses the UBEs to consider the query in a conjunctive manner, which determines whether or not d will be fully scored. The threshold θ is used to determine how conjuctive the query is considered to be. If $\theta = \phi$ a candidate must contain all query terms to qualify for scoring. As θ is lowered, fewer terms are required to be in d to qualify for scoring. Therefore θ provides a trade-off between speed and accuracy. As opposed to Maxscore, WAND looks to short-circuit scoring of entire documents using the θ threshold.

Both algorithms maintain a "quorum" of posting list pointers in order to cull the number of candidates. Based on R or θ, a minimum number of terms must be present in order for a candidate to be fully scored. For Maxscore, as R increases (due to more documents being scored), the quorum begins to shrink because fewer pointers must be checked to determine candidacy (imagine when missing only one term causes $\hat{\phi} < R$ to be true). For WAND, the authors describe the quorum implicitly as a "pivot term", which is the first term such that $\sum_{q \in Q} \phi_q < \theta$. In both cases, only candidates in the quorum posting lists are considered. All iterators not in the quorum do not actively provide candidates, therefore the total candidate pool shrinks as the quorum shrinks.

2.3 Limitations of Prior Work

The field-based models we consider project each query term into each field to produce a score (if the term does not appear in a field, its score contribution reduces to background). This means for n query terms and m fields, we open $n \times m$ index pointers to evaluate the given query. Both Maxscore and WAND have been shown to work quite effectively for unigram queries [14, 1, 10]. However both mechanisms rely on the UBEs of the posting lists to be directly combinable in order to either update a dwindling potential (as in Maxscore), or provide a cumulative estimate of a subset of terms (as in WAND). All of the field-based models have a sum or product across the query terms involved, but each term score is actually a function of a set of term/field scores. Therefore, in order to calculate a score for term i, we must generate $s_{t_i f_1}$ thru $s_{t_i f_m}$. Then after generating s_{t_i}, the algorithm decides whether to forgo scoring the rest of the document or continue.

The problem lies in the lack of granularity when looking for the pruning threshold. Suppose we are using Maxscore to prune as we score, and we have established at least k candidates. Therefore R is a bounded number. We move to

[1]This is in fact not an abuse of language. The complete heuristic is when we consider $h = s_{t_i f_j} - \phi_{t_i f_j}$ to update $\hat{\phi}$. Therefore if $\phi_{t_i f_j}$ does not underestimate, h does not overestimate, which is the technical definition.

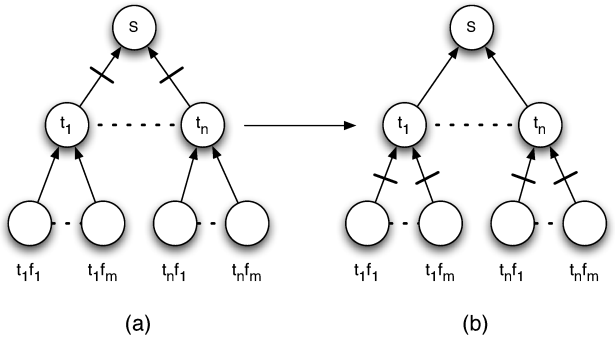

Figure 1: Two different locations for pruning decisions. The small lines indicate where the Maxscore algorithm checks to stop processing.

score document d. We first generate s_{q_1}, as described above. We make our update: $\hat{\phi} \leftarrow \hat{\phi} + s_{q_1} - \phi_{q_1}$. We check that $\hat{\phi} > R$ and find that the expression is false. The algorithm skips scoring the remaining terms, and moves to the next candidate. However if this decision could have been made after only determining $s_{q_1 f_1}$, we could have saved even more by not generating the rest of s_{q_1}. Under the standard formulation, this is not possible for these models. Fig. 1(a) shows the current scenario. Recent research by Cartright and Allan suggests that when the Maxscore algorithm only applies to the higher-level nodes of a layered query, the performance of the algorithm can be significantly improved if the query can be "flattened" to provide access to the lower level nodes [3], therefore we would like to score as in Fig. 1(b). Reformulating a given model into its δ-function form allows us to evaluate documents in such a manner. Hence the name of these functions — using each one causes a small change (a delta) to the current estimated score ($\hat{\phi}$). After applying each of the functions for a particular document, we arrive at S_d, the intended score for document d.

3. DERIVING δ-FUNCTIONS

We now describe the steps to derive a δ-function, and proceed to derive the δ-function for a relatively simple model. As stated above, we would like to process a document as in Fig. 1(b), therefore we need a single function for each term-field pair that can provide the full update to the running estimate. Using the substitution scheme defined above, we would like $\hat{\phi}^{nm} = S_d$, meaning after $n \times m$ substitutions, we arrive at S_d. We express ϕ as $\hat{\phi}^0$. Using these equalities, we can express the transformation of ϕ into S_d via a finite telescoping series:

$$\begin{aligned} S_d =& \left(\cdots \left(\hat{\phi}^0 + \left(\hat{\phi}^1 - \hat{\phi}^0 \right) \right) + \cdots + \left(\hat{\phi}^{nm} - \hat{\phi}^{(nm)-1} \right) \cdots \right) \\ =& \hat{\phi}^0 + \sum_{i=1}^{nm} \left(\hat{\phi}^i - \hat{\phi}^{i-1} \right) = \hat{\phi}^{nm} = S_d \end{aligned} \quad (1)$$

The summation form of the series provides a clean analytical representation of S_d using the potentials. We have $\hat{\phi}^0$ in hand (recall all documents start with this score), therefore we need to figure out $\left(\hat{\phi}^i - \hat{\phi}^{i-1} \right)$ for any $i > 0$. This quantity is the core of the δ-function. For a given model M, for a given t_u and f_v s.t. $1 \leq u \leq n$ and $1 \leq v \leq m$, we define

$$\delta_M^i = \left(\hat{\phi}_M^i - \hat{\phi}_M^{i-1} \right) = \hat{\phi}_M^{i-1}[\phi_{t_u f_v} / s_{t_u f_v}] - \hat{\phi}_M^{i-1} \quad (2)$$

which allows us to rewrite Eq. 1 as

$$S_d = \hat{\phi}^0 + \sum_{i=1}^{nm} \delta_M^i$$

The potential functions $\hat{\phi}_M$ are dependent on the model formulation, as they involve substituting in the term scoring function for that model. In general, the superscript over the δ-function is implied, and we drop it to refer to the generic form. Note that the only difference between $\hat{\phi}^i$ and $\hat{\phi}^{i-1}$ is a single substitution. Our goal, for any given M, is to determine the effect of that substitution on $\hat{\phi}$. Also notice that t_u and f_v are not defined in terms of the previous substitutions, but just as adjustments to the current estimate $\hat{\phi}^{i-1}$. This means that we may order the δ-functions in any way we like when scoring, and are not required to compose an entire term score before producing a decidable estimate.

3.1 A Quick Derivation

As a brief example, suppose we have a new field-based retrieval model SOF, which is just a pair of nested sums over the term fields, and that we are given an admissible estimator $\phi_{t_i f_j}$. We formulate SOF, and its corresponding potential, as follows:

$$S_{Qd} = \sum_{i=1}^{n} w_i \sum_{j=1}^{m} s_{q_i f_j d} \;\;,\;\; \phi_{SOF} = \sum_{i=1}^{n} w_i \sum_{j=1}^{m} \phi_{q_i f_j}$$

The exact formulation of $s_{q_i f_j d}$ is not important. We only need access to the quantity when scoring d. We set up δ_{SOF} as follows:

$$\delta_{SOF} = \hat{\phi}^i - \hat{\phi}^{i-1} = \hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}] - \hat{\phi}^{i-1}$$

Noting that $\hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}]$ reduces to simply subtracting $\phi_{t_u f_v}$ from $\hat{\phi}^{i-1}$ and then adding $s_{t_u f_v d}$ to it, we can write a simple closed form for δ_{SOF}:

$$\begin{aligned} &\hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}] - \hat{\phi}^{i-1} = \\ &\left(\sum_{i=1}^{n} w_i \sum_{j=1}^{m} \phi_{q_i f_j d} - w_u \phi_{t_u f_v} + w_u s_{t_u f_v d} \right) - \\ &\left(\sum_{i=1}^{n} w_i \sum_{j=1}^{m} \phi_{q_i f_j d} \right) = \\ &w_u (s_{t_u f_v d} - \phi_{t_u f_v}) = \delta_{SOF} \end{aligned}$$

We now have a concrete version of δ_{SOF} we can use to score. Brief inspection of δ_{SOF} provides some intuition of its function. The quantity $(s_{t_u f_v d} - \phi_{t_u f_v})$ is negative, since $\phi_{t_u f_v} > s_{t_u f_v d}$. As we intended, each application of the δ-function reduces the estimate a bit, moving it closer to S_{Qd}, until it has been applied for all term and field combinations.

4. REWRITING FIELD-BASED MODELS

We now derive δ-functions for three field-based ranking models. In each case, we determine an admissible UBE

for each term-field pair (e.g. $\phi_{t_i f_j}$), then we derive the δ-function for the given model using that UBE.

4.1 PRMS

The probabilistic retrieval model for semi-structured data (PRMS) by Kim and Croft [9] is an extension to the language model family of retrieval algorithms which incorporates field information into the estimation of relevance. Assuming m fields and n query terms, the score for a document d is the likelihood that d would generate the query Q:

$$P(Q|d) = \prod_{i=1}^{n} \sum_{j=1}^{m} P_\mu(f_j|q_i) P(q_i|f_j, d) \tag{3}$$

where P_μ is the "mapping probability" that field f_j is involved in the relevance estimation, given that q_i appeared. The probability is estimated as follows:

$$P_\mu(f_j|q_i) = \frac{P(q_i|f_j)P(f_j)}{\sum_{f_k \in F} P(q_i|f_k)}$$

We assume a uniform prior distribution for $P(f_j)$. In order to find δ_{PRMS}, we first make sure we have, or can find, an admissible estimator for the model. $P(q_i|f_j, d)$ in Eq. 3 is a language model estimate using either Dirichlet or Jelinek-Mercer smoothing. In the terminology introduced in Section 2.2.1, $P(q_i|f_j, d)$ is $s_{t_i f_j d}$. Macdonald et al. derived max_{tf} as an admissible UBE for this scoring function [10], and we use that estimator here. We can now define the potential of PRMS:

$$\phi_{PRMS} = \prod_{i=1}^{n} \sum_{j=1}^{m} P_\mu(f_j|q_i) \phi_{q_i f_j} \tag{4}$$

Which is the value we start scoring a document from. In the following, we use w_{ij} to refer to $P_\mu(f_j|q_i)$, and let $\hat{\phi}_{t_i} = \sum_{j=1}^{m} w_{ij} \phi_{t_i f_j}$ to simplify the derivation. We start with the generic form of δ_{PRMS}:

$$\delta_{PRMS} = \hat{\phi}^i - \hat{\phi}^{i-1} = \hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}] - \hat{\phi}^{i-1} \tag{5}$$

We need to concretely define $\hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}]$. Given the formulation in Eq. 4, when $i \neq u$, $\hat{\phi}_{t_i}$ is a constant over the course of the substitution, as we are only replacing the value for term i in field j. Therefore, the components of term $i+1$ or term $i-1$, for instance, remain unchanged. So we only need to consider the substitution's effects on $\hat{\phi}_{t_u}$. The composite term score is a sum of term/field estimates, therefore the substitution involves subtracting $w_{uv}\phi_{t_u f_v}$ and adding $w_{uv} s_{t_u f_v d}$:

$$\begin{aligned}
\hat{\phi}_{t_u}[\phi_{t_u f_v} / s_{t_u f_v d}] &= \left(\sum_{j=1}^{m} w_{uj} \phi_{t_u f_j} \right) [\phi_{t_u f_v} / s_{t_u f_v d}] \\
&= \left(\sum_{j=1}^{m} w_{uj} \phi_{t_u f_j} \right) - w_{uv} \phi_{t_u f_v} + w_{uv} s_{t_u f_v d} \\
&= \hat{\phi}_{t_u} + w_{uv}(s_{t_u f_v d} - \phi_{t_u f_v})
\end{aligned}$$

We now define $\Phi = \left(\prod_{i=1}^{n} \hat{\phi}_{t_i} \right)$, the product of all term estimates, and by extension $\Phi_{-u} = \left(\prod_{i=1, i \neq u}^{n} \hat{\phi}_{t_i} \right)$, which is the product of all term estimates, excluding $\hat{\phi}_{t_u}$. Note that $\Phi = \hat{\phi}_{PRMS}$ (the total estimate, possibly with substitutions). This allows us to rewrite Eq. 5 compactly:

$$\begin{aligned}
\delta_{PRMS} &= \hat{\phi}^i - \hat{\phi}^{i-1} = \hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}] - \hat{\phi}^{i-1} \\
&= \Phi_{-u} \left(\hat{\phi}_{t_u} + w_{uv}(s_{t_u f_v d} - \phi_{t_u f_v}) \right) - \Phi \\
&= \Phi_{-u} \left(\hat{\phi}_{t_u} + w_{uv}(s_{t_u f_v d} - \phi_{t_u f_v}) \right) - \Phi_{-u} \hat{\phi}_{t_u} \\
&= \Phi_{-u} \left(\hat{\phi}_{t_u} + w_{uv}(s_{t_u f_v d} - \phi_{t_u f_v}) - \hat{\phi}_{t_u} \right) \\
&= \Phi_{-u} w_{uv} \left(s_{t_u f_v d} - \phi_{t_u f_v} \right)
\end{aligned}$$

The result is once again intuitive. Mulitplied out, the weighted potential $\phi_{t_u f_v}$ is removed from the total estimate, while the actual weighted contribution $s_{t_u f_v d}$ is being added. The weight is the term/field weight w_{uv} multiplied by the remaining term estimates. This quantity is fairly compact, but in implementation it can be difficult to correctly maintain Φ_{-u}. We can remedy this by involving $\hat{\phi}_{t_u}$ again:

$$\delta_{PRMS} = \Phi_{-u} w_{uv} \left(s_{t_u f_v d} - \phi_{t_u f_v} \right) =$$

$$\left(\frac{\hat{\phi}_{t_u}}{\hat{\phi}_{t_u}} \right) \Phi_{-u} w_{uv} \left(s_{t_u f_v d} - \phi_{t_u f_v} \right) = \Phi \left(\frac{w_{uv}(s_{t_u f_v d} - \phi_{t_u f_v})}{\hat{\phi}_{t_u}} \right)$$

We can easily maintain cumulative scores for each term (i.e. $\hat{\phi}_{t_i}$), and we already have access to Φ, the total estimate. In this form, δ_{PRMS} reduces to multiplying the current estimate by a small factor. As each replacement is a small negative number, the total probability slightly drops as we iterate through the different term/field pairs.

4.2 BM25F

BM25F is an extension of the BM25 retrieval model. Instead of scoring fields independently, the term frequencies per field are first combined, then scored [13]. We begin by considering the set of formulae that constitute the BM25F scoring function:

$$s_{t_i f_j d} = \frac{c_{t_i f_j d}}{(1 + B_{f_j}(\frac{l_{f_j d}}{l_{f_j}} - 1))} \tag{6}$$

$$s_{t_i d} = \sum_{j=1}^{m} W_{f_j} s_{t_i f_j d} \tag{7}$$

$$\bar{s}_{t_i d} = \frac{s_{t_i d}}{K + s_{t_i d}} idf_{t_i} \tag{8}$$

$$BM25F(Q, d) = \sum_{t \in q \cap d} \bar{s}_{t_i d} \tag{9}$$

where l_{f_j} is the average length of field f_j across all documents, W_{f_j} is a field-specific tuning parameter, and $l_{f_j d}$ is the length of field f_j in document d. As before, we first find an admissible estimate for the term/field scoring function. In this model, the term/field scoring function corresponds to Eq. 6.

4.2.1 Finding $\phi_{t_i f_j}$ for BM25F

We would like to create a reasonable admissable estimate we can make for any term/field pair. We shorten t_i to t and f_j to f in the following. We use the same technique as Macdonald et al. [10], and view the problem as a relaxed

CMP. Let $x = c_{tfd}$, and let $y = l_{fd}$:

$$\begin{aligned} s_{tfd} &= \frac{c_{tfd}}{(1 + B_f(\frac{l_{fd}}{l_f} - 1))} &&= \frac{x}{(1 + B_f(\frac{y}{l_f} - 1))} \\ &= \frac{x}{(1 + B_f(\frac{y}{l_f} - 1))} &&= \frac{x}{1 + \frac{B_f y}{l_f} - B_f} \\ &= \frac{x}{\frac{B_f y}{l_f} + (1 - B_f)} &&= \frac{x}{\alpha y + \beta} \end{aligned}$$

where $\alpha = B_f / l_f$ and $\beta = (1 - B_f)$. We assume both x and y have lower and upper bounds (e.g. x_{max} is the maximum x for any document). We would like to maximize s_{ftd} subject to

- $x \leq y$
- $x_{min} \leq x \leq x_{max}$
- $y_{min} \leq y \leq y_{max}$
- $0 < x_{min} < x_{max}$
- $0 < y_{min} < x_{max} < y_{max}$

This function monotonically increases w.r.t. x and monotonically decreases w.r.t. y, very much like the classic functions studied before [10]. Given our constraints, this function is maximized when $x = y$, since a constraint is that $x \leq y$, and in the case of $x < y$, we can find a larger value by increasing x to y. We substitute x for y, and take the derivative w.r.t. to x to get

$$\frac{\partial s_{tfd}}{\partial x} = \frac{\beta}{(\alpha x + \beta)^2} \tag{10}$$

which is a positive-valued function $\forall x \geq 0$. Therefore we can follow the gradient produced in Eq. 10 to increase the value of x until we reach it's maximum allowable value as defined by the CMP. This value is $\text{argmax}_{d \in D}\, c_{tfd}$, which we label as $\bar{x}$. Based on this derivation, we now have an admissible estimator for Eq. 6. Therefore, for a given term and parameter B_f, we define our estimator as:

$$\phi_{tf} = \frac{\bar{x}}{(1 + B_f \left(\frac{\bar{x}}{l_f} - 1\right)}$$

We now proceed to derive δ_{BM25F}.

4.3 Deriving δ_{BM25F}

Similar to Eqs. 6-9, we define potentials for BM25F as follows:

$$\begin{aligned} \phi_{t_i} &= \sum_{j=1}^{m} W_{f_j} \phi_{t_i f_j} \\ \psi_{t_i} &= \frac{\phi_{t_i}}{K + \phi_{t_i}} idf_{t_i} \\ \psi &= \sum_{t \in q \cap d} \psi_{t_i} \end{aligned}$$

Modifiers described in Section 2.2.1 extend to the ψ quantities defined here. We start with the general version of δ_{BM25F}:

$$\delta_{BM25F} = \hat{\phi}^i - \hat{\phi}^{i-1} = \hat{\phi}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}] - \hat{\phi}^{i-1} \tag{11}$$

As before, when making a single substitution, all $\hat{\psi}_{t_i}$ where $i \neq u$ are held constant during the substitution, and can be ignored. This leaves us with:

$$\begin{aligned} \delta_{BM25F} &= \hat{\psi}_{t_u}^{i-1}[\phi_{t_u f_v} / s_{t_u f_v d}] - \hat{\psi}_{t_u}^{i-1} \\ &= \left(\frac{\phi_{t_i}[\phi_{t_u f_v} / s_{t_u f_v d}]}{K + \phi_{t_i}[\phi_{t_u f_v} / s_{t_u f_v d}]} idf_{t_u}\right) - \left(\frac{\phi_{t_i}}{K + \phi_{t_i}} idf_{t_i}\right) = \\ &\quad \left(\frac{\phi_{t_i}[\phi_{t_u f_v} / s_{t_u f_v d}]}{K + \phi_{t_i}[\phi_{t_u f_v} / s_{t_u f_v d}]} - \frac{\phi_{t_i}}{K + \phi_{t_i}}\right) idf_{t_u} \end{aligned} \tag{12}$$

If we consider the semantics of $\hat{\phi}_{t_u}[\phi_{t_u f_v} / s_{t_u f_v d}]$ w.r.t. BM25F, we find that the substitution has a similar effect as in the PRMS case:

$$\hat{\phi}_{t_u}[\phi_{t_u f_v} / s_{t_u f_v d}] = \hat{\phi}_{t_u} + W_{f_v}(s_{t_u f_v d} - \phi_{t_u f_v})$$

As both formulae use sums over the fields to generate the term values, this similarity is expected. Substituting into Eq. 12, we have:

$$\delta_{BM25F} = idf_{t_u} \left(\frac{\hat{\phi}_{t_u} + \xi_{uv}}{K + \hat{\phi}_{t_u} + \xi_{uv}} - \frac{\hat{\phi}_{t_u}}{K + \hat{\phi}_{t_u}}\right)$$

where $\xi_{uv} = W_{f_v}(s_{t_u f_v d} - \phi_{t_u f_v})$. As in the case of δ_{PRMS}, we must maintain $\hat{\phi}_{q_i}$ for all $q_i \in Q$. However using this value it is easy to compute the value of δ_{BM25F}.

4.4 Divergence From Randomness

The final model we consider is the PL2F variant of the Divergence From Randomness (DFR) framework. PL2F was first introduced by Macdonald et al. at TREC 2005 [11]. The PL2F model is formulated as follows:

$$S_{Q,d} = \sum_{q_i \in Q} \frac{|q_i|}{|\bar{q}|} \frac{1}{s_{q_i} + 1} \left(s_{q_i} \log \frac{s_{q_i}}{\lambda} + (\lambda - s_{q_i}) \log e + 0.5 \log(2\pi s_{q_i})\right) \tag{13}$$

where $\lambda = cf_{q_i} / |D|$, the collection frequency of q_i divided by the size of the collection. $|q_i|$ is the frequency of q_i in Q, and $|\bar{q}|$ is the maximum $|q_i|$ for any $q_i \in Q$. All log functions are in base 2 unless otherwise noted. For this model s_{t_i} is defined as

$$s_{t_i} = \sum_{j=1}^{m} w_{f_j} s_{t_i f_j d} \quad \text{, where} \tag{14}$$

$$s_{t_i f_j d} = c_{t_i f_j d} \log \left(1 + B_{f_j} \frac{l_{f_j}}{l_{f_j d}}\right) \tag{15}$$

where B_{f_j} is a field-specfic tunable parameter. We assume that $|q_i| = |\bar{q}| = 1$ for all terms in all queries. As before, we now derive an admissible UBE $\phi_{t_i f_j}$ before proceeding to derive δ_{PL2F}.

4.5 Finding ϕ_{tif_j} for PL2F

We need to generate an admissible UBE for the term/field scoring function of PL2F. Using the formulation above, that is Eq. 15. We begin there:

$$s_{t_i f_j d} = c_{t_i f_j d} \log \left(1 + B_{f_j} \frac{l_{f_j}}{l_{f_j d}}\right)$$

We make the same substitutions as before; let $x = c_{t_i f_j d}$, and $y = l_{f_j,d}$:

$$s_{t_i f_j d} = x \log\left(1 + \beta y^{-1}\right) \quad (16)$$

where $\beta = B_{f_j} l_{f_j}$. Inspection reveals that $s_{t_i f_j d}$ monotonically increases with x and monotonically decreases with y. As before, the maximal contour is the line where $x = y$: for any case where $x < y$, we can increase the value of x to y to obtain a larger value. We can now substitute x for y to simplify Eq. 16 and take its derivative:

$$f(x) = s_{t_i f_j d} = x \log\left(1 + \beta x^{-1}\right)$$
$$f'(x) = \frac{\partial s_{t_i f_j d}}{\partial x} = \log(1 + \beta x^{-1}) - \left(\frac{\beta x^{-1}}{(1 + \beta x^{-1}) \ln 2}\right)$$

We need to know the behavior of $f'(x), \forall x > 0$. Substituting $z = \beta x^{-1}$ and taking the derivative again:

$$\frac{\partial f'}{\partial z} = \frac{\frac{z}{(1+z)\ln 2}}{((1+z)\ln 2)^2}$$

We can see that for $\frac{\partial f'}{\partial z} > 0, \forall z > 0$. This means that for all $x > 0$, $f'(x)$ and therefore $f(x)$ are also positive. As $x \to \infty$, $f(x)$ grows unbounded. With respect to our constraints, this function maximizes at $\text{argmax}_{d \in D}\, c_{t_i f_j d}$, or $\bar{x}$, as before. We now have an admissible UBE for PL2F:

$$\phi_{t_i f_j} = \bar{x} \log\left(1 + B_{f_j} \frac{l_{f_j}}{\bar{x}}\right)$$

We can now proceed to deriving the δ-function.

4.6 Deriving δ_{PL2F}

We first determine the total potential ϕ for PL2F using the UBE. We define ϕ as:

$$\phi = \sum_{t_i \in Q} \frac{1}{\phi_{t_i} + 1}\left(s_{t_i} \log \frac{\phi_{t_i}}{\lambda} + (\lambda - \phi_{t_i}) \log e + 0.5 \log(2\pi\phi_{t_i})\right)$$
$$\phi_{t_i} = \sum_{j=1}^{m} w_{f_j} \phi_{t_i f_j}$$

We start with the generic formulation of δ_{PL2F}:

$$\delta_{PL2F} = \hat{\phi}^i - \hat{\phi}^{i-1} = \hat{\phi}^{i-1}[\phi_{t_u f_v}/s_{t_u f_v d}] - \hat{\phi}^{i-1} \quad (17)$$

The substitution takes place at field j for term i. Given the formulation in Eq. 13, terms are calculated independently; we can ignore the other term quantities as they are constant over the substitution. This makes determination of $\hat{\phi}^{i-1}[\phi_{t_u f_v}/s_{t_u f_v d}]$ simple:

$$\hat{\phi}^{i-1}[\phi_{t_u f_v}/s_{t_u f_v d}] = \phi_{t_u} + w_{uv}(s_{t_u f_v d} - \phi_{t_u f_v})$$

Although the replacement over a term potential is simple, deriving a concrete form of the δ_{PL2F} is more involved. Let $\psi_{t_u} = \hat{\phi}_{t_u}[\phi_{t_u f_v}/s_{t_u f_v d}]$. We can now write the terms of concern in δ_{PL2F}:

$$\delta_{PL2F} = \hat{\phi}^i - \hat{\phi}^{i-1} = \hat{\phi}^{i-1}[\phi_{t_u f_v}/s_{t_u f_v d}] - \hat{\phi}^{i-1} =$$
$$\frac{1}{\hat{\phi}_{t_u} + 1}\left(\hat{\phi}_{t_u} \log \frac{\hat{\phi}_{t_u}}{\lambda} + (\lambda - \hat{\phi}_{t_u}) \log e + 0.5 \log(2\pi\hat{\phi}_{t_u})\right) -$$
$$\frac{1}{\psi_{t_u} + 1}\left(\psi_{t_u} \log \frac{\psi_{t_u}}{\lambda} + (\lambda - \psi_{t_u}) \log e + 0.5 \log(2\pi\psi_{t_u})\right)$$

Test Set	# docs	terms	fields	(Tokens/Field) / σ
	(10^6's)	(10^6's)	(1's)	(10^6's)
R05	1.0	484	2	223 / 153.1
TB06	25.2	22,333	3	5,304 / 5,870
OL	39.4	1,418	22	64.4 / 85.2

Table 1: Statistics on the collections used in experiments. The last column shows the average number of tokens per field for that collection. The second value in that column is the standard deviation of the distribution of tokens per field.

In it's current form the δ_{PL2F} function is not very helpful — for every update we would have to calculate the original function twice. We can simplify this function somewhat, by establishing a common denominator between all the terms, and multiplying the two quantities out. After canceling and reducing terms, we finally end up with a less wasteful δ_{PL2F} form:

$$\delta_{PL2F} = \beta(\hat{\phi}_{t_u} - \psi_{t_u}) +$$
$$\log \psi_{t_u}(\psi_{t_u}\hat{\phi}_{t_u} + 0.5\hat{\phi}_{t_u} + \psi_{t_u} + 0.5) -$$
$$\log \psi_{t_u}(\psi_{t_u}\hat{\phi}_{t_u} + 0.5\psi_{t_u} + \hat{\phi}_{t_u} + 0.5)$$

where $\beta = (\log\lambda + \lambda \log e + 0.5 \log 2\pi + \log e)$. This function is still rather bulky, however optimization of this function to use fewer operations is beyond the scope of this paper. We use this form in our implementation, which we now discuss.

5. EXPERIMENTAL SETUP

We conduct experiments over three different data sets: 1) the TREC Robust 2005 track query set and AQUAINT collection (R05), 2) the TREC Terabyte 2006 track query set, over the GOV2 document collection (TB06), and 3) a download of the metadata records from the OpenLibrary (OL), along with a sample of 50 queries from the OpenLibrary Apache logs. Table 1 contains some statistics about each of the collections considered. R05 documents have a `headline` field, and the body content stored as the `text` field. The TB06 documents have similar fields (`title` and `body`), but have `a` (anchor) text indexed as an additional field. R05 and TB06 provide a relatively standard case of web pages, with only a few fields of interest, and one of the fields significantly outweighing the others in terms of content density (i.e. the `body` field is significantly bigger than the `title` or `anchor` fields).

The OL records provide a contrastive dataset - there are 22 fields, none of which contain significantly more content than the rest. The records are community-built, therefore not all fields are present in all documents. While a proper characterization of OL is beyond the scope of this paper, the simple statistics in Table 1 indicate the differing structure in OL versus R05 and TB06. The average number of tokens per field is much lower than the other two test sets, and despite the larger number of fields, the standard deviation of the number of terms in each field is lower than the other two collections.

We conduct experiments using an open-source research IR system[2]. Each run was conducted on an Intel Core2 Duo machine with 4GB RAM, with the index residing on network-

[2]Name anonymized for review

mounted storage. The operating system used is Linux CentOS 5. For all experiments, we perform one run to warm up the system, and then we repeat the same run of queries five times over. To reduce dependence on query order, we randomize the order of the queries between runs (e.g. from run 1 to run 2). The same randomization seeds are used across each set of runs (e.g. PRMS and PL2F receive the same list of seeds), so we can compare the queries in a pairwise manner. We then take the average of the five timed runs for each query, and use that average as the representative run time of the query. All times were measured in milliseconds. We consider both the *latency* of a query (time to process that query), and the *throughput* of a batch of queries (the time to process all of the queries). Our implementation is single-threaded, therefore the throughput is approximately the sum of the query latencies for a batch of queries.

To get a sense of the stability of these runs, in addition to the mean, we calculate the coefficient of variation of each set of query samples. The coefficient of variation is the standard deviation divided by mean, which provides a normalized measure of dispersion of a distribution. For example, a value of 0.05 indicates that the standard deviation of a distribution is only 5% of the value of the mean. Smaller numbers indicate a higher probability that the samples are accurate measurements of the query's actual run time. These values are then aggregated for a collection, to form a distribution of coefficients. Table 2 shows the quartiles for the distributions for each collection. The distributions are rather small, not even going above 0.16 by the 3rd quartile. This suggests that most of the samples are quite reliable. R05 and TB06 have high-end outliers, however, as the jump from the 3rd quartile to the maximum value is many times the value of the 3rd quartile itself. Despite these few outliers, the results provide us with a good deal of confidence that the timed runs are samples of good quality. To compare two runs (say

Set	*Min*	*1st Q*	*Median*	*3rd Q*	*Max*
R05	0.01044	0.03233	0.05785	0.12197	0.97532
TB06	0.00563	0.04033	0.07403	0.15820	1.49304
OL	0.00829	0.04020	0.04936	0.05922	0.10337

Table 2: Different quartiles of the distribution of coefficients of variation of queries for a particular test set. All three collections have small deviations, however R05 and TB06 have high-end outliers, as indicated by the large jump from the 3rd quartile to the maximum value.

the original PRMS formulation and δ_{PRMS}), we report the average drop in query latency as a percentage of the original query latency, which we simply refer to as *percentage improvement*. Let t_A and t_B be times produced by the original (A) and modified (B) runs, respectively. We calculate the percentage improvement as $(t_A - t_B)/t_A$. Therefore a value of 0.9 indicates that run B improved over A's latency by 90%. A negative value indicates that run B ran longer than run A. Note that this measure is upper bounded by 1 (i.e. all processing time has been eliminated), but has no lower bound (the modified run is infinitely slower than the original). The goal then is to get as close to 1.0 as possible. We report the percentage improvement for throughput of the query sets as well. Although throughput tends to be more informative for larger query sets, nonetheless it provides additional information not apparent by just reporting the latency improvements. If the improvement for throughput is higher than for latency, that implies that the longer queries in the set were improved more than the average latency reduction. We also report the number of queries improved or worsened compared to the original formulation. We also report improvement results for the original Maxscore algorithm applied to PRMS, to provide a point of comparison with a state-of-the-art approach.

R05	*μ Thru*	*μ Lat.*	*# better*	*# worse*
Maxscore	8.8	3.6	26	10
δ_{PRMS}	52.6	29.8	35	1
δ_{BM25F}	64.4	38.8	35	1
δ_{PL2F}	12.6	10.3	36	0
TB06	*μ Thru*	*μ Lat.*	*# better*	*# worse*
Maxscore	16.8	2.3	116	34
δ_{PRMS}	28.8	21.6	149	1
δ_{BM25F}	64.7	47.3	141	9
δ_{PL2F}	16.9	21.0	134	16
OL	*μ Thru*	*μ Lat.*	*# better*	*# worse*
Maxscore	6.4	5.1	50	0
δ_{PRMS}	67.9	25.8	50	0
δ_{BM25F}	66.9	18.8	34	16
δ_{PL2F}	2.5	-0.1	19	31

Table 3: Aggregate improvement for the 3 collections. 'μ Thru' refers to mean throughput improvement. 'μ Lat.' refers to mean latency improvement.

We determine statistical sigificance via a two-sample permutation test as described by Efron and Tibshirani [5]. All results are verified to be statistically significant for $p < 0.01$ unless otherwise noted. We omit retrieval effectiveness results, as the pruning algorithms are all verified in theory and implementation to be *safe-to-rank-k*.

6. RESULTS

Table 3 shows the percentage improvement on average latency (time to process one query) and throughput (time to process the entire set of queries). The δ_{PRMS} and δ_{BM25F} implementations signficantly outperform both the original and Maxscore implementations across all three data sets. Additionally, the comparison of latency and throughput indicates that the δ-function form helps more as the query takes more time to execute. Therefore, as queries get longer, the advantage of using δ-functions increases. δ_{PL2F} has more modest improvements, although for R05 and TB06 they are still substantial. We show the by-query percentage improvement of δ_{PRMS} compared to PRMS in Fig. 2, and the same measure for Maxscore compared to PRMS in Fig. 3. The queries are ordered by decreasing improvement, which makes the contrast in improvement clear. δ_{PRMS} improves a signficantly larger proportion of queries, and generally improves them to a much higher degree. The per-query improvements of the other δ-functions over TB06 and R05 look similar, but are omitted to save space.

The results over the OL collection are curious. δ_{PRMS} and δ_{BM25F} have noticeable latency improvement, but surprisingly high throughput improvement. In constrast, δ_{PL2F} tends to actually worsen the average latency, but has a slight improvement on throughput. Despite further analysis, the

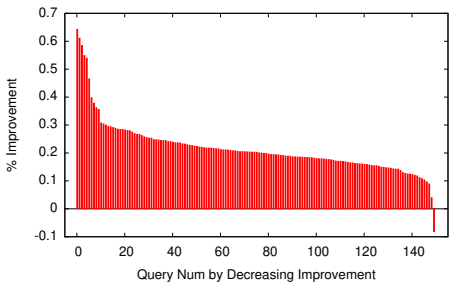

Figure 2: Per-query percentage improvement of δ_{PRMS} compared to the original PRMS. Test set is TB06.

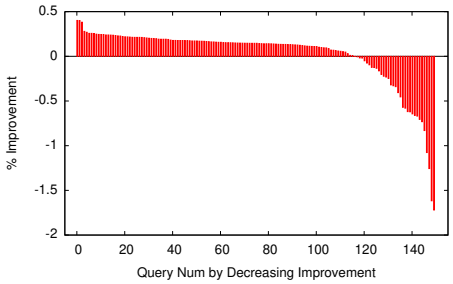

Figure 3: Per-query percentage improvement of Maxscore compared to the original PRMS. Test set is TB06.

reason for the poor performance of δ_{PL2F} on the OL set is still unknown. We look forward to investigating this behavior in future work.

6.1 δ_{PRMS} vs Maxscore

The previous results indicate that in general the δ-functions are more effective at pruning than just using Maxscore. We now compare δ_{PRMS} over the original Maxscore algorithm directly; results are shown in Table 4. Although most of the results are expected, there are a handful of queries where the original Maxscore outperforms the δ_{PRMS} implementation, which is surprising. An brief analysis of these queries does not reveal any obvious characteristics that would cause such behavior. Our current hypothesis is that the aggregation of the term/field nodes into term nodes results in a cleaner ordering of the quorum members than if the iterators are ordered at the term/field level. Note that the current ordering scheme orders the quorum members based on increasing document frequency (df), but the potentials are based on term frequency (tf). Brief statistical analysis shows that df and tf are not perfectly correlated, therefore it may be possible to construct a case where a large number of the iterators have relatively low df, but they all have high average tf. This would keep the quorum from omitting any iterators, which would degrade performance.

vs Maxscore	μ *Thru*	μ *Lat.*	# *better*	# *worse*
R05	48.0	27.5	35	1
TB06	14.3	7.8	131	19
OL	65.7	21.8	50	0

Table 4: Comparison of δ_{PRMS} to Maxscore over the 3 collections.

R05	μ *Thru*	μ *Lat.*	# *better*	# *worse*
δ_{PRMS}	10.4	-5.2	16	20
δ_{BM25F}	16.2	-0.4	17	19
δ_{PL2F}	18.7	8.0	24	12
TB06	μ *Thru*	μ *Lat.*	# *better*	# *worse*
δ_{PRMS}	11.5	-8.5	90	60
δ_{BM25F}	17.7	11.3	143	7
δ_{PL2F}	4.9	9.2	104	46
OL	μ *Thru*	μ *Lat.*	# *better*	# *worse*
δ_{PRMS}	6.1	5.1	50	0
δ_{BM25F}	14.6	1.2	15	35
δ_{PL2F}	3.2	0.2	25	25

Table 5: Aggregate improvement for the 3 collections when no pruning occurs.

6.2 Analysis of Risk

Since δ-functions are new implementations of these retrieval algorithms, their computational efficiency in isolation is unknown. The results so far are very promising, but do the newly implemented functions provide any of the observed speedup, or are they actually a liability? While unlikely, it is possible to generate a particular query set such that no query processed by the system can be pruned, and every candidate must be fully scored. If the δ-functions are inefficient, this could lead to significantly worse performance than changing nothing. To simulate an adversarial scenario where pruning does not occur, we force the quourm to keep all scoring iterators, forcing every candidate to be fully scored.

Table 5 shows the results of the no pruning simulation. We do not show Maxscore; with no pruning available, Maxscore reduces to the original implementation. Even without pruning, all of the δ-functions make a significant improvement on throughput. The improvement on latency is less pronounced, once again indicating that δ-functions provide more relative improvement as the query takes longer to process. These results indicate that instead of being a liability, δ-functions in themselves are an improvement over the original formulation.

7. CONCLUSIONS

We have introduced a new formulation technique for field-based retrieval models that allows systems to exploit dynamic pruning to its fullest. Our results show that direct, iterative scoring of a document allows for finer-grained level of pruning which reduces the execution time of most queries up to 47%. Additionally, the results indicate that δ-functions scale much better than Maxscore as the number of fields increases. This property may be critical, the structure of semi-structured documents is used more and more to improve retrieval effectiveness.

The derivation process of δ-functions is not complex. In the course of this paper we derived the δ-function form for three distinct retrieval models, and realized signficant gains in retrieval efficiency for each model. Although the work

shown here was for field-based models, the steps for deriving a δ function for a particular model can be applied to any retrieval model. The simplicity of the process makes it attractive as a general optimization procedure for various models. Once a new retrieval model has shown its utility in retrieval effectiveness, one can simply derive its δ-function to improve its overall performance, without resorting to ad-hoc solutions.

7.1 Future Work

Although δ-functions make considerable improvements for dynamic optimization for the models covered here, much can be done to improve them. Many computers today have more than one processing core; this provides an opportunity for intra-query parallelism, where various components of a single query may be computed by different cores in parallel. The current derivation process produces a function that is dependent on the previous function. While this makes calculation of a single δ-function simple as we use previous results, it also means that the current derivation cannot be easily parallelized in this way. We would like to explore other versions of these functions that allow parallelization on a multicore machine.

Currently potential estimations are static over the course of a query evaluation. Using "topdocs" to improve the retrieval efficiency has been shown to greatly cut down scoring time by providing a warmup to the scored document queue [2, 14]. In addition to lowering the R value for pruning algorithms, the potential of each term posting list is lowered, as the highest scoring documents are quickly scored first and do not need to be accounted for when making estimating the upper bound. We plan to investigate a similar technique that can adaptively lower the potential estimates during scoring. Doing so would trigger pruning earlier and often during query evaluation.

Finally, we are confident that δ-functions can find wider applicability in the space of retrieval models. As future work we intend to investigate the applicability of this iterative scoring method to training scenarios for learning-to-rank. If we can define a single-step update function for a given model, it could drastically reduce the computational effort needed to perform search during parameter optimization.

8. ACKNOWLEDGMENTS

9. REFERENCES

[1] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. of the twelfth international conference on Information and knowledge management*, CIKM '03, pages 426–434, New York, NY, USA, 2003. ACM.

[2] E. W. Brown. Fast evaluation of structured queries for information retrieval. In *Proc. of the 18th SIGIR*, pages 30–38, New York, NY, USA, 1995. ACM.

[3] M.-A. Cartright and J. Allan. Efficiency optimizations for interpolating subqueries. In *Proc. of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 297–306, New York, NY, USA, 2011. ACM.

[4] W. W. Cohen and H. Hirsh. Joins that generalize: Text classification using whirl. In *In Proc. of the Fourth Int'l Conference on Knowledge Discovery and Data Mining*, KDD '98, pages 169–173, 1998.

[5] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. New York : Chapman & Hall, 1993.

[6] R. Entlich, J. Olsen, L. Garson, M. Lesk, L. Normore, and S. Weibel. Making a digital library: the contents of the CORE project. *ACM Trans. Inf. Syst.*, 15:103–123, April 1997.

[7] E. A. Fox and O. Sornil. Digital libraries. In *Encyclopedia of Computer Science*, pages 576–581. John Wiley and Sons Ltd., Chichester, UK, 2003.

[8] N. Fuhr, N. Gövert, G. Kazai, and M. Lalmas. INEX: INitiative for the Evaluation of XML retrieval. *Proc. of the SIGIR 2002 Workshop on XML and Information Retrieval*, 2006(November):1–9, 2002.

[9] J. Kim and W. B. Croft. Ranking using multiple document types in desktop search. In *Proc. of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 50–57, New York, NY, USA, 2010. ACM.

[10] C. Macdonald, I. Ounis, and N. Tonellotto. Upper-bound approximations for dynamic pruning. *ACM Trans. Inf. Syst.*, 29:17:1–17:28, Dec. 2011.

[11] C. Macdonald, V. Plachouras, B. He, and I. Ounis. University of Glasgow at TREC 2005: Experiments in Terabyte and Enterprise tracks with Terrier. In *Proc. of TREC 2005*, TREC '05, 2004.

[12] P. Ogilvie and J. Callan. Combining document representations for known-item search. In *Proc. of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 143–150, New York, NY, USA, 2003. ACM.

[13] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In *Proc. of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 42–49, New York, NY, USA, 2004. ACM.

[14] T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *Proc. of the 28th SIGIR*, pages 219–225, New York, NY, USA, 2005. ACM.

[15] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31:831–850, November 1995.

[16] X. Yi, J. Allan, and W. B. Croft. Matching resumes and jobs based on relevance models. In *Proc. of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 809–810, New York, NY, USA, 2007. ACM.

[17] H. Zaragoza, N. Craswell, M. Taylor, S. Saria, and S. Robertson. Microsoft Cambridge at TREC-13: Web and Hard tracks. In *Proc. of TREC-2004*, TREC '04, 2004.