

# Technical Report of WAND Search

Xin Liu

July 6, 2012

## Abstract

In this document, we present our research and details of project WAND search, which aims to provide search service for QuestionAnswer (QA) collection and deliver valuable information to users. We first will introduce the prior work on evaluation strategies. And then, we will give complete details of WAND algorithm and its implementation issues.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Prior work . . . . .	3
1.2	Our approach . . . . .	3
<b>2</b>	<b>The Two-level evaluation process</b>	<b>4</b>
2.1	Basic assumptions . . . . .	4
2.2	The wand operator . . . . .	5
2.3	Scoring . . . . .	5
2.4	Implementing the WAND iterator . . . . .	6
2.5	Setting the WAND Threshold . . . . .	9
<b>3</b>	<b>Computing Term Upper Bounds</b>	<b>10</b>
3.1	Least Upper Bounds . . . . .	10
3.2	Upper Bound Approximations . . . . .	11
3.3	NEW UPPER-BOUND APPROXIMATIONS . . . . .	13
	<b>Reference</b>	<b>16</b>

## 1 Introduction

Fast and precise text search engines are widely used in both enterprise and Web environments. While the amount of searchable data is constantly increasing, users have come to expect sub-second response time and accurate search results regardless of the complexity of the query and the size of the data set. Thus, system runtime performance is an increasingly important concern.

Broder [1] proposed an efficient query evaluation method based on a two level approach: at the first level, the method iterates in parallel over query term postings and identifies candidate documents using an approximate evaluation; at the second level, promising candidates are fully evaluated and their exact scores are computed. Compared to the naive approach of fully evaluating every document that contains at least one of the query terms. The method achieves on more than 90% reduction in the number of full evaluations.

At the heart of the approach there is the efficient implementation of WAND operator used in the first evaluation phase.

## 1.1 Prior work

Turtle and Flood [2] classify evaluation strategies into two main classes:

- *Term-at-a-time (TAAT)* strategies process query terms one by one and accumulate partial document scores as the contribution of each query term is computed.
- *Document-at-a-time (DAAT)* strategies evaluate the contributions of every query term with respect to a single document before moving to the next document.

*TAAT* strategies are more commonly used in traditional IR systems. For small corpus sizes, implementations which use *TAAT* strategies are elegant and perform well. For large corpora, *DAAT* strategies have two advantages: (1) *DAAT* implementations require a smaller run-time memory footprint because a per document intermediate score does not need to be maintained, and (2) they exploit I/O parallelism more effectively by traversing postings lists on different disk drives simultaneously.

Both *TAAT* and *DAAT* strategies can be optimized significantly by compromising on the requirement that all document scores are complete and accurate. Optimization strategies have been studied extensively in the information retrieval literature. For a comprehensive overview of optimization techniques see Turtle and Flood [2]. For *TAAT* strategies, the basic idea behind such optimization techniques is to process query terms in some order that lets the system identify the top  $n$  scoring documents without processing all query terms [3][4][5][6][7]. An important optimization technique for *DAAT* strategies is termed *max-score* by Turtle and Flood. Given a recall parameter  $n$ , it operates by keeping track of the top  $n$  scoring documents seen so far. Evaluation of a particular document is terminated as soon as it is clear that this document will not place in the top  $n$ .

Turtle and Flood demonstrated experimentally that in an environment where it is not possible to store intermediate scores in main memory, optimized *DAAT* strategies outperform optimized *TAAT* strategies. In contrast, Kaszkiel and Zobbel [8] showed that for long queries, the *max-score* strategy is much more costly than optimized *TAAT* strategies, due to the required sorting of term postings at each stage of evaluation; a process that heavily depends on the number of query terms.

## 1.2 Our approach

The two-level retrieval algorithm suited for *DAAT* strategies evaluates queries using two levels of granularity. it iterates in parallel over query term postings

and identifies candidate documents using a preliminary evaluation. Once a candidate document is identified, it is fully evaluated and its exact score is computed. Furthermore, as in the standard *DAAT* approach, our algorithm iterates in parallel over query term postings but the nature of the preliminary evaluation is such that it is possible to skip quickly over large portions of the posting lists. If the result of this "fast and rough" evaluation is above a certain threshold, then a *full evaluation* is performed and the exact score is computed.

For large systems the full evaluation is an expensive task that depends on query dependent factors such as term occurrences within the document, as well as query independent properties such as document length, topological score based on link analysis (for HTML pages), etc. Some of these factors might have to be retrieved via an I/O operation but even if all these properties can be computed efficiently, there is still a substantial cost to combine them for an final score. Therefore the intention of the two-level process is to minimize the number of full evaluations as much as possible.

This approach allows both safe optimization and approximate optimization. (This terminology has been introduced in [2]. For safe optimization, the two-level strategy makes no false-negative errors and thus it is guaranteed to return the top documents in the correct order and with accurate scores. For an approximate optimization, dynamic pruning techniques are used such that fewer documents pass the preliminary evaluation step. that is, we allow some false-negative errors at the risk of missing some candidate documents whose accurate scores would have placed them in the returned document set. The amount of pruning can be controlled by the user.

## 2 The Two-level evaluation process

In this section we provide complete details of the two-level evaluation process. Recall that the algorithm iterates over the list of documents identifying candidates using an approximate score. Once such a candidate is identified it is fully evaluated. The algorithm keeps track of the top scoring documents seen so far, under full evaluation. A valid candidate will be a document whose approximate score is greater than the minimal score of all documents in the top scoring set so far.

### 2.1 Basic assumptions

Our model assumes a traditional inverted index for IR systems in which every index term is associated with a posting list. This list contains an entry for each document in the collection that contains the index term. The entry consists of the document's unique positive identifier, DID, as well as any other information required by the system's scoring model such as

number of occurrences of the term in the document, offsets of occurrences, etc. Posting lists are ordered in increasing order of the document identifiers.

## 2.2 The wand operator

At the heart of the two-level evaluation process, there is a new Boolean predicate called **WAND** standing for Weak AND, or Weighted AND. **WAND** takes as arguments a list of Boolean variables  $X_1, X_2, \dots, X_k$ , a list of associated positive *weights*  $w_1, w_2, \dots, w_k$ , and a threshold  $\theta$ .

By definition, **WAND**( $X_1, w_1, \dots, X_k, w_k, \theta$ ) is true if

$$\sum_{1 \leq i \leq k} x_i w_i \geq \theta, \quad (1)$$

where  $x_i$  is the indicator variable for  $X_i$ , that is

$$x_i = \begin{cases} 1 & \text{if } X_i \text{ is true} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Observe that **WAND** can be used to implement **AND** and **OR** via

$$\mathbf{AND}(X_1, X_2, \dots, X_k) \equiv \mathbf{WAND}(X_1, 1, X_2, 1, \dots, X_k, 1, k),$$

and

$$\mathbf{OR}(X_1, X_2, \dots, X_k) \equiv \mathbf{WAND}(X_1, 1, X_2, 1, \dots, X_k, 1, 1).$$

Thus by varying the threshold, **WAND** can ove from being close to an **OR** to being close to an **AND**, which justifies its name as "weak and". **WAND** can be generalized by replacing condition (1) by requiring an arbitrary monotonically increasing function of the  $x_i$ s to be above the threshold, or, in particular, by requiring an arbitrary monotone Boolean formula to be True.

## 2.3 Scoring

The final score of a document involves a textual score which is based on the document textual similarity to the query, as well as other query independent factors such as connectivity for web pages, citation count for scientific papers, etc. To simplify the exposition, we shall assume for the time being that there are no such query independent factors.

We assume an additive scoring model, that is, the textual score of each document is determined by summing the contribution of all query terms belonging to the document. Thus, the textual score of a document  $d$  for query  $q$  is:

$$Score(d, q) = \sum_{t \in q \cap d} \alpha_t w(t, d) \quad (3)$$

For example, for the  $tf \times idf$  scoring model used by many IR systems,  $\alpha_t$  is a function of the number of occurrences of  $t$  in query, multiplied by the inverse document frequency ( $idf$ ) of  $t$  in the index and  $w(t, d)$  is a function of the term frequency if  $t$  in  $d$ , divided by the document length  $|d|$ .

In addition we assume that each term is associated with an upper bound on its maximal contribution to any document score,  $UB_t$  such that

$$UB_t \geq \alpha_t \max(w(t, d_1), w(t, d_2), \dots). \quad (4)$$

By summing the upper bounds of all query terms appearing in a document, we can determine an upper bound on the document's query-dependent score.

$$UB(d, q) = \sum_{t \in q \cap d} UB_t \geq Score(d, q). \quad (5)$$

Note that query terms can be simple terms, i.e., terms for which a static posting list is stored in the index, or complex terms such as phrases, for which the posting list is created dynamically during query evaluation. The model does not distinguish between simple and complex terms; each term must provide an upper bound.

Given this setup our preliminary scoring consists of evaluating for each document  $d$

$$\mathbf{WAND}(X_1, UB_1, X_2, UB_2, \dots, X_k, UB_k, \theta)$$

where  $X_i$  is an indicator variable for the presence of query term  $i$  in document  $d$  and the threshold  $\theta$  is varied during the algorithm. If **WAND** evaluates to true, then the document  $d$  undergoes a full evaluation.

The threshold  $\theta$  is set dynamically by the algorithm based on the minimum score  $m$  among the top  $n$  results found so far, where  $n$  is the number of requested documents. The larger the threshold, the more documents will be skipped and thus we will need to compute full scores for fewer documents. It is easy to see that if the term upper bounds are accurate, then the final score of a document is no greater than its preliminary upper bound, and therefore all documents skipped by **WAND** with  $\theta = m$  would not be placed in the top scoring document set by any other alternative scheme that uses the same additive scoring model.

However, as explained later, we might have only approximate upper bounds for the contribution of each term.

## 2.4 Implementing the **WAND** iterator

We now describe how to iteratively find candidates for full evaluation using the **WAND** predicate. To this end, we must build a **WAND** iterator, that is a procedure that can quickly find the documents that satisfy the predicate.

The WAND iterator is initialized by calling the `init()` method depicted in pseudo-code in Figure 1. The method receives as input the array of query terms. It sets the current document to be considered (`curDoc`) to zero and for each query term,  $t$ , it initializes its current posting `posting[t]` to be the first posting element in its posting list.

```

1. Function init(queryTerms)
2.   terms  $\leftarrow$  queryTerms
3.   curDoc  $\leftarrow$  0
4.   for each  $t \in$  terms
5.     posting[t]  $\leftarrow$  t.iterator.next(0)

```

Figure 1: The `init()` method of the WAND iterator

After calling the `init()` method, the algorithm repeatedly calls WAND's `next()` method to get the next candidate for full evaluation. The `next()` method takes as input a threshold  $\theta$  and returns the next document whose approximate score is larger than  $\theta$ . Documents whose approximate score is lower than the threshold are skipped.

Figure 2 contains pseudo-code of the `next()` method.

The WAND iterator maintains two invariants during its execution:

1. All documents with  $\mathbf{DID} \leq \text{curDoc}$  have already been considered as candidates.
2. For any term  $t$ , any document containing  $t$ , with  $\mathbf{DID} < \text{posting}[t].\mathbf{DID}$ , has already been considered as a candidate.

Note that the `init()` method establishes these invariants.

The WAND iterator repeatedly advances the individual term iterators until it finds a candidate document to return. This could be performed in a naive manner by advancing all iterators together to their next document, approximating the scores of candidate documents in DID order, and comparing to the threshold. This method would, however, be very inefficient and would require several superfluous disk I/Os and computation. Our algorithm is optimized to minimize the number of `next()` operations and the number of approximate evaluations. It first sorts the query terms in increasing order of the DIDs of their current postings. Next, it computes a pivot term—the first term in this order for which the accumulated sum of upper bounds of all terms preceding it, including it, exceeds the given threshold. The pivot DID is the smallest DID that might be a candidate. If there is no such term (meaning the sum of all term upper bounds is less than the threshold) the iterator stops and returns the constant *NoMoreDocs*.

```

1. Function next( $\theta$ )
2.   repeat
3.     /* Sort the terms in non decreasing order of
       DID */
4.     sort(terms, posting)
5.     /* Find pivot term - the first one with accumulated
       UB  $\geq \theta$  */
6.     pTerm  $\leftarrow$  findPivotTerm(terms,  $\theta$ )
7.     if (pTerm = null) return (NoMoreDocs)
8.     pivot  $\leftarrow$  posting[pTerm].DID
9.     if (pivot = lastID) return (NoMoreDocs)
10.    if (pivot  $\leq$  curDoc)
11.      /* pivot has already been considered, advance
        one of the preceding terms */
12.      aterm  $\leftarrow$  pickTerm(terms[0..pTerm])
13.      posting[aterm]  $\leftarrow$  aterm.iterator.next(curDoc+1)
14.    else /* pivot > curDoc */
15.      if (posting[0].DID = pivot)
16.        /* Success, all terms preceding pTerm belong
          to the pivot */
17.        curDoc  $\leftarrow$  pivot
18.        return (curDoc, posting)
19.      else
20.        /* not enough mass yet on pivot, advance
          one of the preceding terms */
21.        aterm  $\leftarrow$  pickTerm(terms[0..pTerm])
22.        posting[aterm]  $\leftarrow$  aterm.iterator.next(pivot)
23.    end repeat

```

Figure 2: The next() method of the WAND iterator

The pivot variable is set to the DID corresponding to the current posting of the pivot term. If the pivot is less or equal to the DID of the last document considered (curDoc), WAND picks a term preceding the pivot term and advances its iterator past curDoc, the reason being that all documents preceding curDoc have already been considered (by Invariant 1) and therefore the system should next consider a document with a larger DID. Note that this move preserves Invariant 2. (NB: In fact in Line 10 the case Pivot < curDoc cannot happen, but this requires an extra bit of proof.)

If the pivot is greater than curDoc, we need to check whether indeed the sum of contributions to the pivot document is greater than the threshold. There are two cases: if the current posting DID of all terms preceding the pivot term is equal to the pivot document, then the pivot document contains a set of query terms with an accumulated upper bound larger than



the threshold and hence `next()` sets `curDoc` to the pivot, and returns this document as a candidate for full evaluation. Otherwise, the pivot document might or might not contain all the preceding terms, that is, it might or might not have enough contributions, hence WAND picks one of these terms and advances its iterator to a location  $\geq$  the pivot location.

Note that the `next()` method maintains the invariant that all the documents with `DID < curDoc` have already been considered as candidates (Invariant 1). It is not possible for another document whose DID is smaller than that of the pivot to be a valid candidate since the pivot term by definition is the first term in the DID order for which the accumulated upper bound exceeds the threshold. Hence, all documents with a smaller DID than that of the pivot can only contain terms which precede the pivot term, and thus the upper bound on their score is strictly less than the threshold. It follows that `next()` maintains the invariant since `curDoc` is only advanced to the pivot document in the cases of success, i.e., finding a new valid candidate who is the first in the order.

The `next()` method invokes two helper methods, `sort()`, `findPivotTerm()` and `pickTerm()`. The first helper, `sort()`, sorts the terms in non decreasing order of their current DID. Note that there is no need to fully sort the terms at any stage since only terms which precede the pivot advance its iterator between consecutive calls to `sort`; hence, by using an appropriate data structure, the sorted order is maintained by modifying the position of only one term. The second helper, `findPivotTerm()`, returns the first term in the sorted order for which the accumulated upper bounds of all terms preceding it, including it, exceed the given threshold.

## 2.5 Setting the WAND Threshold

Assume that we wish to retrieve the top  $n$  scoring documents for a given query. The algorithm will maintain a heap of size  $n$  to keep track of the top  $n$  results. After calling the `init()` method of the WAND iterator, the algorithm calls the `next()` method to receive a new candidate. When a new candidate is returned by the WAND iterator, this document is fully evaluated using the systems scoring model resulting in the precise score for this document. If the heap is not full the candidate is inserted into the heap. If the heap is full and the new score is larger than the minimum score in the heap, the new document is inserted into the heap, replacing the one with the minimum score.

The threshold value that is passed to the WAND iterator is set based on the minimum score of all documents currently in the heap. Recall that this threshold determines the lower bound that must be exceeded for a document to be considered as candidate and to be passed to the full evaluation step.

The initial threshold is set based on the query type. For an OR query, or for a free-text query, the initial threshold is set to zero. The approximate

score of any document that contains at least one of the query terms would exceed this threshold and would thus be returned as a candidate. Once the heap is full and a more realistic threshold is set, we only fully evaluate documents that have enough terms to yield a high score. For an AND query, the initial threshold can be set to the sum of all term upper bounds. Only documents containing all query terms would have a high enough approximate score to be considered candidates.

### 3 Computing Term Upper Bounds

The WAND iterator requires that each query term  $t$  be associated with an upper bound,  $UB_t$ , on its contribution to any document score. In this section, we describe a uniform notation for defining the upper bounds for a term, formulate existing methods for obtaining upper bounds within this notation, and discuss their limitations. In particular, an upper bound for term  $t$  is denoted  $\sigma(t)$ . Within this section, we review the two main methods for obtaining upper bounds: the least upper bound for a given weighting model can be empirically determined at indexing time (Section 3.1); alternatively, an upper bound can be approximated at querying time using appropriate statistics (Section 3.2).

#### 3.1 Least Upper Bounds

The least upper bound (that would be observed for all occurrences of a term) for all terms may be obtained at indexing time, given prior knowledge of the weighting model and a scan of the posting list of each term [1]. In particular, the least upper bound  $\sigma_{LEAST}(t)$  for term  $t$ , is obtained using all documents in the posting list  $L(t)$ .

$$\sigma_{LEAST}(t) = \max_{d \in L(t)} score(tf_d, *d, *t), \quad (6)$$

where  $score(tf_d, *d, *t)$  is the score given by a weighting model for  $tf_d$  occurrences of term  $t$  in document  $d$ .  $*d$  denotes any other document statistics required by a particular weighting model, such as document length (which we denote  $l_d$ ).  $*t$  represents any statistics of the term necessary for the weighting model, such as **IDF**.

However, the traversal of an entire index to determine the least upper bound for each term has some disadvantages: first, such pre-computation may add overhead to the indexing phase, as the entire inverted index must be traversed and scores recorded for every weighting model likely to be used during retrieval.

Moreover, the precomputation means that the pre-specified settings of the weighting model(s) cannot be altered. However, increasingly, selective retrieval approaches are being devised which apply different rankings for

different queries. For example, using one ranking model for all the queries cannot be a suitable solution to conquering the search ranking challenge [9]. As was also found by Kang and Kim [10], a good ranking algorithm for an informational does not always perform well for a homepage finding task. Inspired by these observations, if queries could be divided properly into different groups, it is possible to design different ranking models for each group to improve the search ranking. A more recent work [11] showed that it was beneficial to exploit different ranking models for different queries, calling this process query-dependent ranking, and in He and Ounis [12] weighting models are adapted on-the-fly for different queries. Moreover, the context of a search query often provides a search engine with meaningful hints for better answering the current query [13]. Indeed, the task of query segmentation [14] results in different weighting schemes for a term depending on the context represented by the remaining query terms, and the weighting models can be trained online in light of new relevance data [15].

### 3.2 Upper Bound Approximations

we concern the approximation (e.g.,  $\mathbf{A}, \sigma_{\mathbf{A}}(t)$ ) of upper bounds, using statistics of  $t$  and some knowledge of the applied weighting model, such that  $\sigma_{LEAST}(t) \leq \sigma_{\mathbf{A}}(t)$ . Such statistics can be easily computed at indexing time, without prior knowledge of the weighting model to be applied at retrieval time.

First, we note that Fang et al. [16] describe various heuristics that are common to effective weighting models, relating to the effects of varying  $tf_d$ ,  $l_d$ , and term statistics. These heuristics include several that are of interest in obtaining upper-bound approximations.

*TFC1.* Favour a document with more occurrences of a query term than one with less (assuming equal document lengths).

*TFC2.* Ensure that the change in the score caused by increasing  $tf_d$  from 1 to 2 is larger than that caused by increasing  $tf_d$  from 100 to 101 (assuming equal document lengths).

*LNC1.* Penalize long documents (assuming equal  $tf_d$ ).

In the following, we describe how various upper bounds can be approximated, referring back to these heuristics when appropriate. The accurate approximation of these upper bounds is critical to the efficiency and effectiveness of the TAAT MaxScore, DAAT MaxScore, and DAAT Wand strategies. In particular, using term upper bounds, the strategy can know when it is not worth scoring a given posting for a particular term, as even the maximal contribution that the term could give would not impact the final top- $K$  ranked documents. However, if a term upper bound is too high, then some postings will be needlessly scored, when they would not have any impact on the top ranking of  $K$  results. If a term upper bound is too low, then some postings may incorrectly be omitted from scoring, potentially impacting on

the ranking of the top- $K$  results—making the method only approximate instead of safe-up-to-rank- $K$ . Conversely,  $\sigma_A(t) = \infty$  would be safe, but very inefficient, as it would prevent any pruning from taking place. In approximating the upper bounds, we desire the smallest  $\sigma_A(t) \geq \sigma_{LEAST}(t)$ , to ensure safeness, but maximize efficiency.

Term upper bounds were first proposed by Turtle and Flood [2], in relation to the MaxScore approaches. However, the authors did not discuss the calculation of these upper bounds. We assume that the calculations of such upper bounds were determined as trivial in the presence of simple TF.IDF models, which did not account for the length of documents (i.e., LNC1 was not considered). In particular, for TF.IDF,  $score(tf_d, *d, *t)$  is monotonically increasing as  $tf_d \rightarrow \infty$ , as specified by TFC1. Hence the least upper bound is easily found using the maximum term frequency of term  $t$  in its posting list  $L(t)$  ( $max_{d \in L(t)} tf_d$ ):

$$\sigma_{LEAST:TFIDF}(t) = score \left( max_{d \in L(t)} tf_d, *d, *t \right), \quad (7)$$

where  $tf_d$  is the term frequency of term  $t$  in document  $d$ , and  $*d$  is empty (i.e. document length is not required).

However, modern weighting models often account for document length ( $l_d$ ), in the manner of LNC1. In this case, the approximation of the term upper bounds needs to account for document length, and/or infer upper bounds using knowledge of the weighting model. Due to the interaction of TFC1 and LNC1, it is hard to analytically determine  $\sigma_{LEAST}(t)$  for such weighting models. Instead, for instance, in Broder et al. [1], the authors of Wand observed that  $score(tf_d, *d, *t)$  can be factored into a product  $w(tf_d, *d) \cdot w(*t)$ . While the authors do not specify the weighting model that they use, they state that  $w(*t)$  reflects the IDF component of the weighting model. We note that TF.IDF and BM25 [17] can be factored in this manner, with  $w(tf_d, *d)$  reflecting the factor which varies with  $d$ . Then, an approximate upper bound can be found:

$$\sigma_{Factor}(t) = C \cdot w(t) \quad (8)$$

for some tunable constant  $C$ . We call this approximation Factor. The experiments described in Broder et al. [1] did not tune  $C$ , as it depends on the weighting model applied. Instead, a factor on the current threshold score (which has an inverse relation to  $C$ ) was varied, demonstrating for the WT10G collection the efficiency/effectiveness tradeoffs for aggressive but approximate pruning. Hence, the proper setting of  $C$  to achieve a safe supremum of  $\sigma(t)$  was not investigated. It is of note that Broder et al. [1] did not discuss the use of  $max_{d \in L(t)} tf_d$  in calculating approximate upper bounds. For example, the upper bounds of BM25 (see Equation (11)) can be approximated by  $\sigma_{Factor:BM25}(t) = C \cdot (k_1 + 1) \cdot \frac{(k_3+1)qtf}{k_3+qtf} \log_2 \frac{N-N_t+0.5}{N_t+0.5}$

Later, in Lacour et al. [18], the upper bounds were approximated for the BM11 weighting model which does consider document length  $l_d$ , by replacing  $l_d$  length of all documents  $avg\ l$ :

$$\sigma_{AVGDL}(t) = score(\max_{d \in L(t)} tf_d, avg\ l, *t). \quad (9)$$

In doing so, this approximation (AVGDL) assumes a uniform document length for every posting. However, in their experiments, the retrieval performance of safe-up-to-rank  $K$  techniques were impacted at rank  $K$ , inferring that strategies such as TAAT MaxScore and DAAT MaxScore were being impacted by inexact approximation of the term upper bounds, causing the strategies to become approximate strategies only, and negatively impacting effectiveness.

All of the existing methods to find upper bounds have limitations:  $\sigma_{TFIDF}(t)$  makes no consideration of document length; Factor is limited to weighting models which can be suitably factored in particular, it is inapplicable to weighting models such as those from language modelling (LM) [19] and Divergence From Randomness (DFR) [20]; in contrast, AVGDL may be inexact and not safe. In the next section, we model the term upper-bound approximation problem and perform a mathematical proof of the safeness of upper-bounds for several state-of-the-art weighting models.

### 3.3 NEW UPPER-BOUND APPROXIMATIONS

Modern weighting models, such as BM25 and those from the language modeling and DFR families, take into account document length ( $l_d$ ) in addition to the term frequency ( $tf_d$ ), either as part of a document length normalization process (LNC1), or as part of maximum likelihood estimation ( $\frac{tf_d}{l_d}$ ). In these cases,  $\sigma_{LEAST}(t)$  cannot be exactly calculated using the  $\max_{d \in L(t)} tf_d$  alone without any knowledge of the weighting model.

To counteract the limitations of the Factor and AVGDL approximations, we formulate, in general terms, the problem of approximating a term upper bound given the weighting model. We study the problem for three weighting models from three different families and propose a new upper bound suitable for these weighting models. In particular, we focus on: BM25 [17] an example of a TF.IDF-based weighting model with document length normalization; Dirichlet language modeling (LM) [19] which smooths the maximum likelihood ( $\frac{tf_d}{l_d}$ ) using the probability of occurrence in the collection; and the parameter-free DLH13 [20] from the DFR family of weighting models which also includes document length normalization.

To simplify the notation, in the following we use  $x$  to denote  $tf_d$ ,  $y$  to denote  $l_d$  and Greek letters ( $\alpha, \beta, \gamma$ ) to denote strictly positive constants.

Given a general weighting model  $f(x, y)$  depending on term frequency ( $x$ ) and document length ( $y$ ), the problem of finding an upper bound for  $f(x, y)$

can be formulated as a *constrained maximization problem* (CMP) [21]:

$$\max f(x, y), \text{ subject to } \begin{cases} x \leq y, \\ x_{\min} \leq x \leq x_{\max}, \\ y_{\min} \leq y \leq y_{\max}, \\ x, y \in \mathbb{N}, \end{cases} \quad (10)$$

where  $x_{\min} > 0, y_{\min} > 0, x_{\max} > x_{\min}, y_{\max} > y_{\min}, x_{\max} > y_{\min}, \text{ and } y_{\max} > x_{\max}$  are reasonable assumptions for any real world IR system and document corpus. For instance, an IR system based on an inverted index will not score documents where a query term does not occur (hence  $x_{\min} > 0$ ), and similarly, empty documents (i.e.  $y_{\min} = 0$ ) will not be scored. Moreover, for any given document, any term can appear a number of times not greater than the total number of tokens in the document ( $x \leq y$ ). Finally, the upper bound on the number of occurrences on a given term in any document, and the length of any document in a posting list are finite ( $x \leq x_{\max}, y \leq y_{\max}$ ) and countable. Without loss of generality, we assume  $x_{\min} = y_{\min} = 1$ . These constraints define the admissible region of the problem, that is, the area of the  $x, y$  plane where the term statistics have acceptable values.

Our approach for approximating upper bounds uses statistics for each term (e.g.,  $x_{\max}, y_{\max}$ ), and uses these to calculate the score of a special document which would define an upper bound on the score of any permissible posting for that term. These statistics are easily calculable at indexing time, and not specific to any particular weighting model or setting. Then, based on the CMP defined by the constraints in Equation (10), we analyze to determine the position of the special document in the  $x, y$  plane which has maximal score. Such a special document probably does not exist, and hence the least upper bound will lie at some other point inside the admissible region; nevertheless, the approximate upper bound is guaranteed to be safe.

The CMP defined above is integral and nonlinear, and hence very difficult to manipulate. Since we are interested in an approximate upper bound, we relax the problem by removing the  $x, y \in \mathbb{N}$  constraint. In this case, the admissible region defined by the remaining constraints is regular because the constraints are affine [21]. Hence, the problem can be studied using constrained optimization methods. In the following, we study the analytical behaviour of the BM25, LM, and DLH13 weighting models on the relaxed admissible region, and analytically derive the solutions of the relaxed constrained maximization problems by using first order differential calculus.

In BM25 [17], the relevance score of a document  $d$  for a query term  $t$  is given by

$$\text{score}(x, y, N_t) = \frac{(k_1 + 1)x}{k_1((1 - b) + b\frac{y}{\text{avg}_l}) + x} \frac{(k_3 + 1)tf_q}{k_3 + tf_q} w^{(1)}, \quad (11)$$

where  $tf_q$  is the frequency of the query term  $t$  in the query;  $b, k_1$ , and  $k_3$  are parameters (defaults  $b = 0.75, k_1 = 1.2, \text{ and } k_3 = 1000$  [17]).  $w^{(1)}$  is

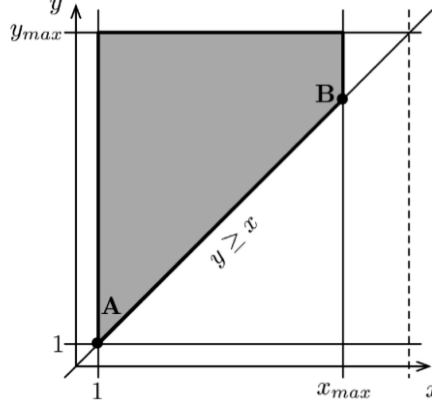


Figure 3: Admissible region for the relaxed CMP

the IDF factor, which is given by  $w^{(1)} = \log_2 \frac{NN_t+0.5}{N_t+0.5}$ .  $N$  is the number of documents in the whole collection.  $N_t$  is the document frequency of term  $t$ .

**THEOREM 1.** The solution of the relaxed CMP for Equation (11) is given by  $x = x_{max}$  and  $y = x_{max}$ .

**PROOF.** Equation (11) can be conveniently rewritten as

$$f(x, y) = \beta \frac{(k_1 + 1)x}{k_1((1 - b) + b\frac{y}{\alpha}) + x} = \frac{\beta' x}{x + \alpha' y + \gamma} \quad (12)$$

where  $\alpha' = \frac{k_1 b}{avg_l}$ ,  $\beta = (k_1 + 1) \frac{(k_3 + 1) t f_q}{k_3 + t f_q} w^{(1)}$ ,  $\gamma = k_1(1 - b)$ . It can be seen that this function is strictly monotonically decreasing in  $y$ , as suggested by LNC1, and strictly monotonically increasing in  $x$ , as per TFC1.

Figure 3 is a graphical representation of the constraints in Equation (10). The shaded region identifies the admissible region where the maximum point must lie, bounded by  $1 \leq x \leq x_{max}$ ,  $1 \leq y \leq y_{max}$ , and  $y \geq x$ . The segment of  $y = x$  within these bounds is denoted  $AB$ . Next, because  $f(x, y)$  is monotonically decreasing in  $y$  and strictly monotonically increasing in  $x$ , it follows that the maximum of Equation (11) is reached on the segment  $AB$ , because for any other point in the admissible region, it is always possible to find another point closer to  $AB$  with a greater value of  $f(x, y)$ . Along the segment  $AB$  (where  $y = x$ ), we have

$$f(x, y)|_{y=x} = f(x) = \frac{\beta' x}{(1 + \alpha')x + \gamma}, \quad (13)$$

$$\frac{\partial f}{\partial x} \Big|_{y=x} = \frac{df}{dx} = \frac{\beta' \gamma}{((1 + \alpha')x + \gamma)^2} \quad (14)$$

Hence, for  $y = x$ ,  $f(x)$  has a positive, continuous derivative, meaning that

its maximum is reached at  $B$ , where  $x = x_{max}$  and  $y = x_{max}$ .

## References

- [1] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, 2003.
- [2] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831850, 1995.
- [3] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *Proceedings of the Eighth International ACM-SIGIR Conference*, pages 97110, Montreal, Canada, June 1985.
- [4] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society of Information Science*, 41(8):581589, 1990.
- [5] M. Persin. Document filtering for fast ranking. In *Proceedings of the Seventeenth International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 339348, Dublin, Ireland, July 1994.
- [6] A. Moffat and J. Zobel. Fast ranking in limited space. In *Proceedings of the 10th IEEE International Conference on Data Engineering*, pages 4284376, Houston, TX, February 1994.
- [7] A. N. Vo and A. Moffat. Compressed inverted files with reduced decoding overheads. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 290297, Melbourne, Australia, August 1998.
- [8] M. Kaszkiel and J. Zobel. Term-ordered query evaluation versus document-ordered query evaluation for large document databases. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 343344, Melbourne, Australia, August 1998.
- [9] ZHU, Z. A., CHEN, W., WAN, T., ZHU, C., WANG, G., AND CHEN, Z. To divide and conquer search ranking by learning query difficulty. In *Proceedings of the 18th International ACM Conference on Information and Knowledge Management*. 2009.



- [10] KANG, I.-H. AND KIM, G. Query type classification for Web document retrieval. *In Proceedings of the 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2003.
- [11] GENG, X., LIU, T.-Y., QIN, T., ARNOLD, A., LI, H., AND SHUM, H.-Y. 2008. Query-dependent ranking using k-nearest neighbor. *In Proceedings of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008.
- [12] HE, B. AND OUNIS, I. A query-based pre-retrieval model selection approach to information retrieval. *In Proceedings of the 7th International Conference on Computer-Assisted Information Retrieval*, 2004.
- [13] XIANG, B., JIANG, D., PEI, J., SUN, X., CHEN, E., AND LI, H. Context-aware ranking in web search. *In Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010.
- [14] BENDERSKY, M., CROFT, W. B., AND SMITH, D. A. Two-stage query segmentation for information retrieval. *In Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009.
- [15] CROFT, W. B., METZLER, D., AND STROHMAN, T. Search Engines Information Retrieval in Practice. AddisonWesley, Reading, MA, 2009.
- [16] FANG, H., TAO, T., AND ZHAI, C. A formal study of information retrieval heuristics. *In Proceedings of the 27th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.
- [17] ROBERTSON, S. E., WALKER, S., HANCOCK-BEAULIEU, M., GULL, A., AND LAU, M. Okapi at TREC. *In Proceedings of the 1st Text REtrieval Conference*, 1992.
- [18] LACOUR, P., MACDONALD, C., AND OUNIS, I. Efficiency comparison of document matching techniques. *In Proceedings of the Efficiency Issues in Information Retrieval Workshop at ECIR*. 2008.
- [19] ZHAI, C. AND LAFFERTY, J. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inform. Syst.* 22, 2, 179214. 2004.
- [20] AMATI, G. Frequentist and bayesian approach to information retrieval. *In Proceedings of the 28th European Conference on IR Research*, 2006.

- [21] JONGEN, H. T., MEER, K., AND TRIESCH, E. Optim. Theo. Springer, Berlin, Germany. 2004.