Efficient & Effective Selective Query Rewriting with Efficiency Predictions

Craig Macdonald
University of Glasgow
Glasgow, Scotland, UK
craig.macdonald@glasgow.ac.uk

Nicola Tonellotto
ISTI-CNR
Pisa, Italy
nicola.tonellotto@isti.cnr.it

Iadh Ounis
University of Glasgow
Glasgow, Scotland, UK
iadh.ounis@glasgow.ac.uk

ABSTRACT

To enhance effectiveness, a user's query can be rewritten internally by the search engine in many ways, for example by applying proximity, or by expanding the query with related terms. However, approaches that benefit effectiveness often have a negative impact on efficiency, which has impacts upon the user satisfaction, if the query is excessively slow. In this paper, we propose a novel framework for using the predicted execution time of various query rewritings to select between alternatives on a per-query basis, in a manner that ensures both effectiveness and efficiency. In particular, we propose the prediction of the execution time of *ephemeral* (e.g., proximity) posting lists generated from uni-gram inverted index posting lists, which are used in establishing the permissible query rewriting alternatives that may execute in the allowed time. Experiments examining both the effectiveness and efficiency of the proposed approach demonstrate that a 49% decrease in mean response time (and 62% decrease in 95th-percentile response time) can be attained without significantly hindering the effectiveness of the search engine.

ACM Reference format:
Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2017. Efficient & Effective Selective Query Rewriting with Efficiency Predictions. In *Proceedings of SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan, ,* 10 pages.
DOI: http://dx.doi.org/10.1145/3077136.3080827

1 INTRODUCTION

Search engines, such as those for the Web, are required to be effective at answering users' queries but yet also efficient. In particular, while the relevance of the results are important for users' satisfaction, users are in general not willing to wait long for the results to arrive [36]. While search engines are operated in distributed retrieval settings that can be scaled horizontally to reduce response times, the ramification of this is increased cost (in terms of both capital outlay and running, e.g., for power) for the search engine infrastructure. This being the case, the efficiency of the search engine is key to providing effective results without excessive financial burden. Typically, the infrastructure is designed to maintain a service level, where high percentile response time (the so-called "tail latencies" [16, 19]) should not exceed a given target. Indeed, for the Bing search engine, the target is reported to be that 99% percentile response time should not exceed 100 ms [19].

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

DOI: http://dx.doi.org/10.1145/3077136.3080827

Table 1: Example rewrites for the query 'poker tournament'.

Original query:	poker tournament
Stemming:	poker #syn(tournaments tournament)
Proximity:	poker tournament #1(poker tournament)∧0.1
	#uw8(poker tournament)∧0.1
Stemming and Proximity:	poker #syn(tournaments tournament)
	#1(poker #syn(tournaments tournament))∧0.1
	#uw8(poker #syn(tournaments tournament))∧0.1

On the other hand, techniques that benefit the effectiveness of a search engine may hinder efficiency [41], due to their complex nature. For example, in a modern search engine deploying learning-to-rank approaches, the number of features to be computed and the learned models both contribute complexity, and have been the subject of recent studies (e.g., [25]). However, the time to traverse the inverted index's posting lists for the query terms, to identify the top K documents — which are then re-ranked by the learned approach — takes significant time [14].

Often the query submitted by the user is internally *rewritten* by the search engine to improve the quality of the search results [20, 33]. For instance, traditional pseudo-relevance feedback approaches typically results in a much larger query, with significant negative impact on efficiency. More recently, less aggressive query rewriting approaches such as term proximity [30], query substitutions [20] and query-time stemming [34] have been deployed by search engines. Each query rewriting approaches can lead to a query with additional terms, resulting in prolonged execution times.

Table 1 shows three possible rewritings of the query 'poker tournament', based on application of combinations of stemming [34] and sequential dependence (proximity) [30]. In the rewritten examples using an Indri-like query language, *complex query operators* [37] denote additional postings lists that must be traversed during retrieval, namely: #syn, which combines the constituent terms into a single posting list; #1 creates a posting list representing an exact occurrence of an n-gram; and #uwλ creates a posting list that provides the number of times a n-gram appears in an unordered window of size λ [37]. While the search engine may have indexed posting lists for some n-grams, not all possible query operators may have existing posting lists, and hence *ephemeral* posting lists are required, which need to be created on-the-fly from the constituent terms. Moreover, statistics such as the total number of postings in an ephemeral posting list are unknown, and hence the number of postings to be processed and the resulting execution time of a query containing complex operators cannot be known in advance.

Indeed, while recent work in query efficiency prediction has shown the possibility of estimating the execution time of a query prior to its processing [19, 21, 29, 38], none of the existing work

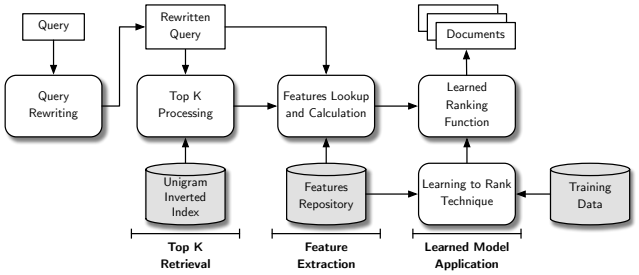

Figure 1: A pictorial representation of the reference web search engine architecture that we consider in this work.

has considered the execution time of queries containing query operators that generate ephemeral posting lists, such as #syn, or #1. This makes it difficult to select among query rewriting strategies that use such operators, as their likely execution time is unknown. Hence, in this work, we study the cost of scoring ephemeral posting lists, and use these observations to define accurate query efficiency predictions for advanced query operators. Furthermore, we use these query efficiency predictions to instantiate a novel mechanism that selects the best strategy among alternative query rewritings, to improve efficiency, while minimising impact on effectiveness.

Our conducted experiments to measure the efficiency of our proposed selective mechanism upon TREC Web track test collections show that a 49% decrease in mean response time, and 62% decrease in tail (95th-percentile) response time, can be attained without significantly hindering the effectiveness of the search engine. The contributions of this work are as follows: we show how to make query efficiency predictions for ephemeral posting lists created by complex operators such as #syn and #1; we use these advanced query efficiency predictions to propose a selector mechanism that permits the query to be rewritten in an effective manner while considering a target response time that the search engine should aim to meet.

The remainder of this paper is structured as follows: Section 2 provides an overview of a reference search engine architecture, describing the necessary background; Section 3 positions our contributions with respect to existing work; Section 4 describes our mechanism for selecting among query rewrites; Section 5 proposes new query efficiency predictors suitable for application to complex query operators. Our experimental setup and results follow in Sections 6 & 7. Finally, we provide concluding remarks in Section 8.

2 PRELIMINARIES

In this section, we provide some essential background on index organisation and query processing in search engines. In doing so, we follow the reference architecture for a search engine depicted by Figure 1. The following section summarises and discusses the state-of-the-art query rewriting techniques and approaches addressing efficient but effective retrieval.

Index Organisation. Given a collection $\mathcal{D}$ of documents, each document is identified by a non-negative integer called *document identifier*, or *docid* d_i. A *posting list* I_t is associated to each *term* t appearing in the collection, containing the list of the docids of all the documents in which the term occurs at least once. The collection of the posting lists for all of the terms is called the *inverted index* of $\mathcal{D}$, while the set of the terms is usually referred to as the *lexicon*. For each term t, the lexicon stores a pointer to its posting list as well as additional information on the statistics of the term in the collection, such as its *document frequency* N_t, i.e., the length of its posting list, and the total number of occurrences of the term in the collection F_t. Each *posting* in a posting list typically contains additional information about the term's occurrences in the document, such as the number of occurrences $f_{t,d}$, and the set of *positions*, $p_{t,d}$, where the term t occurs [13]. This position information facilitates phrasal retrieval without resort to large n-gram index data structures.

The docids in a posting list can be sorted in increasing order enabling the use of efficient compression algorithms and query processing [31]; or the posting lists can be frequency-sorted [39] or impact-sorted [2], allowing for good compression rates, but also presenting practical disadvantages such as their difficulty of use for phrasal queries [22, 37]. As such, in this paper, we focus on the more common search scenario of docid-sorted index layouts [15].

Query Processing. The top K ranked retrieval stage identifies the K highest scored documents in the collection, where the *relevance score* is a function of the query-document pair. Multi-stage retrieval systems have become the dominant model for efficient and effective web search engines [14]. In such systems (see Figure 1), a first "top K" stage retrieves from the inverted index a relatively small set of K possibly-relevant documents matching the user query, focusing on optimising recall. Subsequent stages compute additional query dependent (e.g., fields [28], proximity) and query independent features, before applying a learning-to-rank technique to re-rank the K documents coming from the first stage, aiming to maximise measures like NDCG [25]. The inverted index posting lists are processed in the first stage only, to produce a small set of candidate documents, that will be re-ranked in the subsequent stage(s).

Documents are typically scored in the first stage retrieval by the (weighted) linear combinations of weighting model (e.g., BM25, language models) functions computed for each term-document pair. Such weighting models are usually monotonically increasing in the number of occurrences of the term in the document $f_{t,d}$. An obvious way to compute the top K scored documents is to exhaustively apply the weighting model to all the documents that match at least one query term in the inverted index. As such an exhaustive method is very expensive for large collections, several *dynamic pruning* techniques have been proposed in the last few years. Dynamic pruning makes use of the inverted index, augmented with additional data structures, to skip documents that cannot reach a sufficient score to enter the top K. Thus, the final result is the same as an exhaustive evaluation, but obtained with significantly less work. These techniques include MaxScore [39], WAND [6], and BMW [17]. In this paper, we focus our attention on the WAND strategy, since we deal with (rewritten) queries that can have a large number of terms (i.e., long queries). Indeed, several previous studies have confirmed that MaxScore performs better than WAND for short queries while the opposite happens for long queries [18, 31] while, as we will discuss in Sec. 5, the BMW techniques are not suitable for processing rewritten queries.

WAND augments the posting list of each term t with an *upper bound* σ_t on the maximum score of that term among all documents in the list. While processing the query by iterating on the posting lists of its terms, it records the top K scores among the documents

evaluated thus far. To enter the top K, a new document needs to have a larger score than the current K-th score, which we call the *min score*. WAND maintains the posting list iterators sorted by increasing docid; at every step, it sums up the maximum scores of the lists in increasing order, until the min score is reached. It can be seen that the current docid of the first list that exceeds the min score is the first docid that can reach a score higher than the min score, so the other iterators can safely skip all the documents up to that docid. The alignment of the posting lists during WAND processing is achieved by means of a $\mathsf{next}_t(d)$ method upon the posting list iterators, which returns the smallest docid in the posting list I_t that is greater than or equal to d. This functionality significantly enhances the retrieval speed exhibited by WAND, by skipping docids that would never be retrieved in the top K, and hence avoiding their decompression and scoring. Indeed, smaller values of K allow for more skipping, since the threshold is in general larger for small values of K than for large values, resulting in smaller query processing times, as reported, for example, in [38].

3 RELATED WORK

In the following, we survey existing work in query rewriting and in query efficiency predictions, and position our work accordingly.

Query Rewriting. There are a number of related works across the areas of query rewriting and efficient yet effective retrieval. The internal rewriting of a user's query within an IR system has a long history, including pseudo-relevance feedback in the form of automatic query expansion, first deployed by the SMART system in TREC-3 [7], while others considered the correction of spelling errors or the application of ontologies to identify related concepts [13, Ch. 6]. Indeed, query expansion, which adds additional terms to the query based upon their appearance in the top ranked documents, has been shown to be effective for adhoc retrieval tasks in evaluation forums such as TREC [42]. For web search, the significantly longer generated queries, as well as the need to conduct two retrieval phases, make pseudo-relevance feedback approaches infeasible for efficient retrieval. A further risk is the possibility that the topic of the expanded query can drift from the intent of the initial query.

Instead, some of the techniques widely deployed in web search have focused on rewriting the query based on the large amounts of user interaction data available to a web search engine. For instance Jones et al. [20] describe a way to mine common query reformulation patterns, based on log likelihood ratio, that can be automatically applied to refine a new query. Random walks on the query-click graph [12] offer similar possibilities for identifying common paraphrasing queries.

Rewriting the query to include common variants of the original query terms can have an important effectiveness benefit in addressing the word mismatch problem. Indeed, a query-side approach to stemming has a marked advantage of index-time stemming, in that the other words within the query can be taken into account to decide if the stemming is appropriate for a given word. For instance, Peng et al. [34] describe a context-sensitive stemming approach where *query segments*[1] are carefully considered for stemming, by comparing the language model generation probability of both the original and the replacement segments. Naturally, adding additional terms to the query can have a marked negative impact on efficiency, hence, as noted by Peng et al., it is not desirable to rewrite queries unnecessarily. In our work in this paper, query rewriting by application of stemming is one of the query rewriting techniques that we consider.

One method of query rewriting that has gained significant benefits in effectiveness is the application of term dependence (proximity) operators, to boost the retrieval of documents where the query terms occur close together [30, 35]. In particular, Metzler & Croft's Markov Random Field *sequential dependence model* makes use of the Indri complex query operators #1 and #uwλ formed from adjacent pairs of query terms, added to the original query terms with low weights (typically [0.05,0.1]). However, such rewritten queries have a negative efficiency impact, in that more posting lists must be traversed, while if the index only has unigram posting lists, ephemeral posting lists must be created to handle the #1 and #uwλ operators[2]. Another variant, the *full dependence model* – which adds complex query operators for each pair of query terms – is generally considered too inefficient for common retrieval use [4, 30]. Hence, for large-scale environments, there has to be a perceived benefit in deploying such a term dependence model, due to its inherent negative efficiency impact. For this reason, we note various works that extract term dependence proximity features at the re-ranking stage [28, 38] – an approach that we deploy within our baseline retrieval system in this paper.

On the other hand, motivated by the inefficiencies in deploying sequential dependence, Wang et al. [41] proposed an efficient variant where the weights for bi-gram operators were adjusted to jointly optimise combinations of effectiveness and efficiency, and bi-grams predicted not to be useful were eliminated. In this way, the work of Wang et al. is one of the closest to our work. However, their features for estimating the cost of the #1 and #uwλ bi-gram complex operators assumed the existence of bi-gram index statistics, something that our approach does not require; Moreover, their experiments did not consider deployment under a dynamic pruning strategy, where the efficiency cost of additional complex proximity operators, – which have low weighting in the query (see Table 1) – may be markedly reduced by the pruning. Finally, as we consider more than just proximity rewriting, our work is more general than that of Wang et al. [41].

Efficiency Predictions. Using a docid-sorted index layout, the time taken for a query to execute is correlated with the length of the posting lists of the query's constituent terms, as these posting lists must be traversed during execution. Dynamic pruning techniques such as WAND and BMW offer some relief as they offer the potential to safely skip the decompression of postings and the scoring of documents that cannot make the current top K. This makes the exact response time of a query difficult to predict, as not every posting in the postings lists will be decompressed and scored. Nevertheless recent work has considered making accurate predictions on the efficiency of a query, either in terms of absolute response time [29], or in terms of those queries with response times exceeding a threshold [19, 21].

Efficiency predictions facilitate a number of applications for ensuring efficient yet effective retrieval - for instance, routing queries

[1] N-gram subsequences of queries that demonstrate the underlying grammatical structure, usually determined by dividing longer sequences to maximise n-gram language model probabilities.

[2] For instance, according to the recent IR system reproducibility effort [23], Indri's average response time is decreased by a factor of 6 on deploying sequential dependence on the Gov2 corpus.

among busy replicated query shard servers [29]; selectively deploying multiple CPU cores for slow queries [19, 21]; or adjusting the pruning aggressiveness or size of K for different queries [5, 14, 38]. Of these, the work of Tonellotto et al. [38] is among the most similar to ours, in that they vary the number of documents to be retrieved, K, as well as the pruning aggressiveness, before passing to a learning-to-rank re-ranking phase, based on the predicted execution time of the query. Similarly, in a very recently published work, Culpepper et al. [14] defined an approach for training the rank cutoff in a multi-stage ranking system based on closeness in overlap to a "reference" system. However, their approach has a key disadvantage in that they use a simple reference system, and hence would not demonstrate the benefit in going beyond that system, for instance in deploying advanced query rewrites.

Indeed, differently from [14] & [38], in this paper, we go further by considering a prediction of the execution time of possible rewritings of the users' original queries. This is made possible by the novel prediction of the execution time of complex query operators such as #syn and #uwλ. In the following, we firstly define the problem and introduce our selection mechanism (Section 4), before defining how to obtain query efficiency predictions for queries involving complex operators in Section 5.

4 SELECTING AMONG QUERY REWRITINGS

4.1 Problem Statement

This work considers the efficient yet effective rewriting of a given query q. Indeed, the search engine may consider several possible ways to reformulate the query into a refined instance q', for instance, by splitting compound words, adding alternative words identified using stemming algorithms or common query reformulations approaches, or adding proximity terms such as #1. Some such query rewritings can result in a longer query formulation, hindering its efficiency compared to the original query [33].

Reformulating the query in a multi-stage ranking system – such as one deploying learning to rank – can be seen as aiming to improve the recall of the K documents retrieved in the Top K Retrieval stage. This ensures that when re-ranking the documents by the application of the learned model, the search engine has a high chance of identifying the most relevant documents for promotion to the top of the ranked list for presentation to the user. Naturally, improving the formulation of the query may also increase the high precision of these K documents – i.e., by retrieving more relevant documents towards the top of that initial list. This suggests that some rewritten queries only need a smaller $K' < K$. Moreover, as mentioned above, the efficiency of dynamic pruning techniques like WAND is benefited by smaller K.

Hence, with various different rewriting techniques available, a natural question arises: for a given query q, which possible rewritings are appropriate to be applied to the query, $q'_1 \dots q'_n$, such that effectiveness may be improved, and/or, K reduced to improve efficiency, without significantly damaging the overall effectiveness.

4.2 Selection Mechanism

To achieve this, we make use of query efficiency predictions that estimate the execution time of different rewritings of the query, before one is selected and executed. A particular challenge in doing so is making accurate estimations of the execution times of rewritten queries that use operators such as #syn, #1 and #uwλ. We discuss this further in Section 5.

Firstly, we generate all possible rewritings $\{q'_1 \dots q'_n\}$ of the original query q. We note that the size of this set varies for each query q, as not all rewritings are applicable to each query – for instance, no term dependence can be applied to a query with only a single term, or no stemming may be applicable for each query term. At the very least, the original query will always be present.

We also consider m different K values for the number of documents to retrieve in the first retrieval phase, which will then be re-ranked by application of the learned model, namely $K_1 \dots K_m$. In doing so, our intuition is that for some queries, simply identifying $K = 20$ documents will be sufficient to identify enough relevant documents, leading to marked efficiency benefits, particularly if the rewritten query permits higher recall of relevant documents within the top K. A possible *plan* for executing a query can be denoted as the tuple $\langle q'_i, K_j \rangle$. All possible query plans, denoted $\mathcal{P}(q)$, for executing the query can be generated from the Cartesian product:

$$\mathcal{P}(q) = \left\{ \langle q'_i, K_j \rangle \right\} = \{q'_1 \dots q'_n\} \times \{K_1 \dots K_m\} \tag{1}$$

The aim is then to rank and eliminate plans $\langle q'_i, K_j \rangle$ based on their predicted efficiency and expected effectiveness. While the number of possible rewrites for each query varies, $|\mathcal{P}(q)| \geq m$.

Consider that the search engine has a service level agreement in place [19], which aims to maximise the number of queries answered in time τ. Some plans for queries may exceed τ. We use query efficiency predictions, denoted $\widehat{t}(\langle q'_i, K_j \rangle)$ to eliminate such plans:

$$\mathcal{EP}(q, \tau) = \left\{ \langle q'_i, K_j \rangle \ \in \mathcal{P}(q) \,|\, \widehat{t}(\langle q'_i, K_j \rangle) \leq \tau \right\} \tag{2}$$

Naturally, plans will vary in effectiveness. One possible approach to select among the feasible plans $\mathcal{EP}(q, \tau)$ would be to try to predict the effectiveness of a given rewriting of a query. However, Tonellotto et al. [38] examined the usefulness of query performance (effectiveness) predictors within their selective pruning mechanism, and found them to have little correlation with maintaining effectiveness while enhancing efficiency. Moreover, we are not aware of any existing works that make effectiveness predictions in the presence of complex operators used by some rewritings.

Instead, to determine the likely effectiveness of a plan $\langle q'_i, K_j \rangle$, we measure the expectation of the effectiveness for some measure μ (e.g., NDCG) of that given rewriting upon a set of training queries Q^{tr}. Denoting with $\langle Q^{\text{tr}}_i, K_j \rangle$ the set of query plans for the rewritten queries according to the i-th rewriting, we compute the expected effectiveness of measure μ over all such query plans, $E_\mu(\langle Q^{\text{tr}}_i, K_j \rangle)$. However, due to excessively long posting lists, some queries may not have any plans that can be executed in time τ, i.e., $|\mathcal{EP}(q, \tau)| = 0$. For this reason, for such queries, we resort to a best attempt, by selecting the plan with the fastest predicted execution time. The final selection mechanism to identify the best plan $\mathcal{FP}(q, \tau)$ for executing the (possibly rewritten) query q in time τ is as follows:

$$\mathcal{FP}(q, \tau) = \begin{cases} \underset{\langle q'_i, K_j \rangle \in \mathcal{EP}(q,\tau)}{\text{argmax}} \ E_\mu(\langle Q^{\text{tr}}_i, K_j \rangle) & \text{if } |\mathcal{EP}(q, \tau)| > 0 \\ \underset{\langle q'_i, K_j \rangle \in \mathcal{EP}(q,\tau)}{\text{argmin}} \ \widehat{t}(\langle q'_i, K_j \rangle) & \text{otherwise} \end{cases} \tag{3}$$

The usefulness of this mechanism is driven by the need to have accurate estimation of the time to execute a given query plan, namely $\widehat{t}(\langle q'_i, K_j \rangle)$. Moreover, as mentioned above, the estimation of the

Table 2: Complex operators summary table.

Complex operator	Complex term	Ephemeral posting list
#syn	#syn(car, cars)	Disjunctive/OR
#1	#1(new, york)	Conjunctive/AND
#uwλ	#uw8(divx, codec)	Conjunctive/AND

execution time of complex query operators, particularly under dynamic pruning strategies such as WAND, have not previously been addressed. In the next section, we propose a novel method to address this problem, by using machine-learned models to predict the execution times of query plans that use complex operators.

5 EFFICIENCY PREDICTION FOR COMPLEX OPERATORS

The complex operators involved in stemming and proximity rewritings are summarised in Table 2. Such complex operators can be applied to two or more uni-gram terms, as well as other complex operators, i.e., complex operators can be nested. Complex operators, such as #1, generate complex terms once instantiated, such as #1(new, york). In the following, we discuss how the posting lists for such complex terms are generated (Section 5.1) and how they can be processed together with the original terms in the WAND strategy (Section 5.2). Section 5.3 presents our approach for predicting the query processing time of queries containing complex terms.

5.1 Complex terms and ephemeral posting lists

While posting lists for simple (e.g., uni-gram) terms are stored in the inverted index, it is not feasible to pre-compute statistics and posting lists for complex terms, since their space occupancy will quickly become unmanageable, particularly if complex operators can be nested. Hence, for a complex term #op($t_1, \ldots,$ #op$_1, \ldots$), its posting list and statistics must be generated on-the-fly, such that it can be processed together with other simple and complex terms. Such *ephemeral posting lists* are materialised as required, depending on the complex operator and the involved terms. We consider two types of ephemeral posting lists: disjunctive/OR-based lists and conjunctive/AND-based lists (see Table 2). OR-based posting lists are used with #syn operators, where the posting lists of two or more terms are merged into a single posting list containing all docids appearing in at least one of the terms' posting lists. AND-based posting lists are used with #1 and #uwλ operators, where two or more terms' posting lists must be intersected, i.e., a posting appears in the ephemeral posting list only if it is present in all of the involved posting lists.

The involved complex operator defines how to merge postings into ephemeral posting lists. In OR-based ephemeral posting lists, when the postings of two different terms that refer to the same docid are merged, their term frequencies in a document must be added, and the positions arrays must be merged. Similarly, while the terms' frequencies in the collection (F_t) must be summed, we cannot know in advance the document frequencies (N_t) of the resulting ephemeral posting list, as the number of docids in common between the two posting lists is unknown. In AND-based ephemeral posting lists, there is no general way to merge term frequencies in documents, while term positions must be stored separately and processed depending on the semantics of the complex operator [24]. Again, the document frequency N_t of the resulting ephemeral posting list, i.e., its length, cannot be known in advance.

5.2 Query Processing

Given a user query q composed of *simple terms*, let us assume it is rewritten into a query composed of simple terms and *complex terms*, collectively denoted by q'. Complex terms are obtained by applying complex operators – summarised in Table 2 – to simple or other complex terms, in a nested fashion. The score of a document d w.r.t. the rewritten query q' can be expressed as follows:

$$s(q', d) = \sum_{t \in q \cap d} w_t \, s_{t,d} + \sum_{t' \in (q' \setminus q) \cap d} w_{t'} \, s_{t',d}, \tag{4}$$

where w_t (resp. $w_{t'}$) denotes the simple (resp. complex) terms weights (see Table 1), while $s_{t,d}$ and $s_{t',d}$ are weighting models for simple and complex terms respectively. The linearity of Eq. (4) makes it easily usable in any exhaustive query processing algorithm, by exploiting ephemeral posting lists as normal posting lists.

Conversely, dynamic pruning techniques are not directly usable, since they rely on the maximum score σ_t of each query term, calculated among all documents in the respective posting lists, i.e., $\sigma_t = w_t \cdot \max_{d \in I_t} s_{t,d}$. These maximum scores can be computed offline by taking the score value of the top document of a single term query stored in the lexicon. However, since ephemeral posting lists are materialised on-the-fly, such a computation cannot be performed offline, hence we must resort to a runtime estimation for upper bounds on these maximum scores. In [26], the authors proposed a general framework to approximate the term upper bounds for proximity weighting models that monotonically increase with respect to the frequency variable, called MaxTF. In that framework, the upper bound σ_t for a term t is computed by using the maximum term frequency $f_{\max}(t)$ that appears in the term's posting list as input for the scoring model, i.e.,

$$\sigma_t = w_t \, s_{t,d}\Big(f_{\max}(t)\Big), \text{ where } f_{\max}(t) = \max_{d \in I_t} f_{t,d}. \tag{5}$$

In our experiments, we exploit the DLH13 weighting model [1] for #syn operators and simple terms, and the pBiL dependence weighting model [35] for #uwλ and #1 operators[3]. Both models are a generalisation of the parameter-free hypergeometric DFR model in a binomial case, DLH13 for simple terms while pBiL for n-grams, and both are monotonically increasing with respect to the frequency variable $f_{t,d}$. Hence, given a complex term, we can compute a term upper bound by using the maximum term/n-gram frequency computed from the corresponding ephemeral posting list, as in the MaxTF framework. Unfortunately, even computing these maximum frequencies is impossible without a complete view of the posting lists, hence we must resort to a further upper bound on those frequencies, easily computed as follows:

$$\begin{aligned} f_{\max}\Big(\#\mathsf{syn}(t_1, t_2)\Big) &= f_{\max}(t_1) + f_{\max}(t_2), \\ f_{\max}\Big(\#\mathsf{1}(t_1, t_2)\Big) &= \min\Big\{f_{\max}(t_1), f_{\max}(t_2)\Big\}, \\ f_{\max}\Big(\#\mathsf{uw}\lambda(t_1, t_2)\Big) &= \min\Big\{f_{\max}(t_1), f_{\max}(t_2)\Big\}. \end{aligned} \tag{6}$$

[3] We use these models as they are parameter free, and their MaxTF upper-bound approximations were proven in [26]; However, the method we describe here is equally applicable for scoring simple and complex terms using Dirichlet language modelling. BM25 cannot be applied, as N_t is not available while scoring complex terms.

For instance, consider an AND-based ephemeral posting list (such as #1): no document can have more occurrences of the n-gram than the *minimum* of the maximum frequencies of the involved terms that has been observed in their posting lists. For the OR-based #syn operator, the worst-case maximum frequency would occur if a single document had the maximum frequencies of the constituent terms. Hence, the maximum frequency within a #syn ephemeral posting list cannot be greater than the sum of the maximum observed frequencies of the respective constituent posting lists.

Such term upper bounds, computed by substituting Equations (6) into Equation (5) as required, can be used in dynamic pruning strategies such as MaxScore and WAND, which leverage *global* upper bounds on each term, both simple and complex. Conversely, query processing strategies such as BMW leverage *local* term upper bounds: each posting list is split into consecutive blocks of constant size, e.g., 128 postings per block, and, for each block, a score upper bound is computed and stored, together with largest docid of each block. This cannot be done with ephemeral posting lists, since their sequences of postings is not known in advance. Hence, no block score upper bounds can be computed or stored.

5.3 Predicting Complex Operator Efficiency

The time spent processing a query in dynamic pruning strategies depends on several factors, the most important being: (i) the number of terms to be processed, (ii) the total number of postings to be processed and (iii) the relative importance of terms, i.e., their score contribution to the overall relevance of a document [29]. As shown in [19, 21, 29, 38], by using statistics derived from terms and queries it is possible to estimate the query processing time. However, since we must deal with ephemeral posting lists, most of the statistics used in prior research are not applicable. For example, the mean scores of postings (used in [29]) cannot be precomputed for an ephemeral posting list, since the postings and their statistics for all possible n-grams cannot feasibly be computed offline. Hence, we must resort to "estimators" of the quantities of interest, in particular for the total number to be processed, i.e., the simple and ephemeral posting list lengths, and the corresponding term upper bounds.

To provide an upper bound approximation to the number of postings in an ephemeral posting list, consider a generic complex term $\#\mathrm{op}(t_1, t_2)$, where t_1 and t_2 can be simple or (nested) complex terms. If #op corresponds to an OR-based posting list such as #syn, then the size of its posting list cannot be greater than the sum of the document frequencies of its constituent terms. If #op corresponds to an AND-based posting list, then the total size of its postings list cannot be greater than the minimum of the document frequencies of its constituent terms, i.e., the smallest constituent posting list.

For term scores upper bound approximations, we leverage the MaxTF framework, adapted to complex operators, as summarised in Eq. (6). Upper bound approximations are scaled according to the term-specific weight resulting from query rewriting (see Table 1).

We adopt a machine-learned approach to predict the processing times of complex queries, i.e., $\widehat{t}(\langle q'_i, K_j \rangle)$, for different values of K. As we use the WAND dynamic pruning strategy (as justified in Section 2), the response times heavily depend on the K number of documents to be retrieved. Hence, we train a different model for each value of K, but all models share the same set of statistics. All previous works on query processing time prediction use of query statistics and aggregations (max, min, mean etc.) of term-based statistics across the query terms, as reported in Table 3. However, different complex operators acting on the same set of terms can have very different impacts on the running time of a query. Hence, we propose an additional, intermediate step between statistics generation and aggregation, namely *projection*. In this step, all query- and term-based statistics are divided into subsets, whose elements are grouped depending on the nature of the term. Thus we have several statistics projections, one for every complex operator, one for simple terms and one considering all terms globally. Term-based aggregators are then applied to these subsets of statistics. Note that, due to the different nature of AND- and OR-based posting lists, we do not consider the sum and variance operators, while we include the minimum operator, as suggested by [19]. In our experiments, we use #uw8 and #uw12 operators, hence we have 6 query-based features and 2×6×5 term-based features, for a total of 66 features. Section 7.1 provides the details and parameters of the learning algorithm using these feature to predict the processing times of complex queries.

Table 3: Prediction statistics, projectors and aggregators.

	Statistics
1.	Number of terms (query-based)
2.	Document frequency (term-based)
3.	Score upper bound (term-based)
	Projectors
1.	Global
2.	Original-only terms
3.	#syn-only terms
4.	#1-only terms
5-...	#uwλ-only terms, one per different λ
	Aggregators
1–2.	Minimum, Maximum
3–5.	Arithmetic, Harmonic, Geometric Means

6 EXPERIMENTAL SETUP

6.1 Research Questions

In the following, we experiment to address two research questions:

- RQ1: How accurate are our query efficiency predictions for queries with complex operators? (Section 7.1)
- RQ2: Can we maintain effectiveness while reducing response time when selectively rewriting queries? (Section 7.2)

In the remainder of this section, we define the experimental setup under which our experiments are conducted.

6.2 Datasets & Retrieval System

Our experiments address both efficiency and effectiveness, and hence require diverse setups to ensure accurate conclusions can be drawn for both types of measures. All of our experiments are conducted on the TREC ClueWeb09 category B corpus[4], which consists of 50M Web documents. For testing effectiveness, we use the 197 queries from the TREC Web tracks 2009-2012 that have corresponding relevance assessments on a 4-point scale [9], and denote this as WT. For testing efficiency, we follow best practices in

[4] http://lemurproject.org/clueweb09/

sampling a significant number of queries from a real search engine, namely 1,956 successive queries from the MSN 2006 query log[5] [11].

We index all 50M documents of the ClueWeb09 corpus using the Terrier IR platform [32], including also the anchor text of incoming hyperlinks to each document. Position information is recorded for each term. Our index is compressed using Elias-Fano encoding provided in [40], widely considered to be the state-of-the-art in terms of fast decompression.

For retrieval, we conduct efficiency timings using a machine equipped with an AMD Opteron Processor 6276 with 6 MB L3 cache and 128 GB RAM. The entire index is loaded in memory. All experiments are performed on a single core. While the resulting retrieval times using a single machine for retrieval are marginally higher than would be expected for interactive retrieval in a deployed Web search engine, following previous work [41], this does not detract from the generality of the findings, and avoids the complexities of performing experiments in a distributed retrieval environment.

Finally, in our timing experiments, we do not include the time to rewrite the query, calculate efficiency predictions, nor to apply the learned model. Each of these stages is comparatively cheap: for instance, the rewriting approaches discussed below are commonly deployed in search engines; query efficiency predictions can be calculated quickly using only term statistics from the lexicon [19, 29]; and the application of a learned model is also relatively less expensive than the top-K retrieval [38].

6.3 Query Plans

Besides the None strategy, where the original queries are not rewritten, we deploy the following rewriting approaches:

MRF. To encapsulate proximity in rewriting a query, we deploy sequential dependence [30] proximity, by considering #1, #uw8 and #uw12 query operators for adjacent query terms.

Naïve. To address word mismatch, we rewrite the query by adding alternatives to query terms within a #syn query operator[6]. Inspired by Peng et al. [34], who noted corpus analysis as being a suitable method to determine similar words (based on which words they co-occur with). We select alternative terms for a given term t that each have the same stem as t based on Porter's stemmer, and which are among the M most similar terms to t within a word embedding space. We use Deeplearning4j's word2vec tool and word embeddings vectors trained on Wikipedia and the Gigaword corpus[7] and select words with common stems in the top $M = 20$ for each query term t. This results in a less aggressive stemming than either index-time stemming or the equivalent query-side stemming using all alternatives identified by a Porter stemmer.

NaïveMRF. Finally, we mix Naïve and MRF rewritings to create a final strategy of generating time-expensive query plans.

Table 4 reports the statistics of each query set, incl. the number of simple and complex terms generated by each rewriting approach. Finally, with regards to the value of K, i.e., the number of documents retrieved by WAND during the top K retrieval, we select 4 values, namely 20, 100, 1000 and 5000. This gives us a total of 16 query plans to be plugged into our selective mechanism, and for which we need to train and test our efficiency predictors.

[5] From a sample of 2000 queries, 44 queries had no matching terms in our collection, so were removed. [6] Initial experiments using the context-sensitive approach of [34] showed no effectiveness benefit over this simpler Naïve stemming. [7] Available from http://nlp.stanford.edu/projects/glove/

Table 4: Query sets statistics per rewriting.

Dataset	Queries	Rewriting	simple	#syn	#1	#uw8	#uw12
Train	978	NaïveMRF	1458	1125	1133	1133	0
		MRF	2556	0	1578	1578	781
		Naïve	1458	1125	0	0	0
		None	2556	0	0	0	0
Test	978	NaïveMRF	1452	1028	1185	1185	0
		MRF	2468	0	1491	1491	768
		Naïve	1452	1028	0	0	0
		None	2468	0	0	0	0
WT	197	NaïveMRF	121	126	113	113	0
		MRF	244	0	146	146	82
		Naïve	121	126	0	0	0
		None	244	0	0	0	0

Table 5: Query-dependent (QD) & -independent (QI) ranking features within our experiments.

QD	DLH13, Coordinate Level	2
QD	pBiL (term dependence)	2
QI	Inlinks, Outlinks, PageRank	3
QI	URL features	6
QI	Content quality [3]	4
QI	Spam score	1
Total		18

6.4 Ranking features

As mentioned in Section 5, in the top K retrieval phase, we use the DLH13 term weighting model [1] from the Divergence from Randomness framework for weighting simple and #syn terms; For #1 and #uwλ terms, we use the DFR pBiL model [35]. Next, Table 5 lists the 18 features used for re-ranking the top K results during the application of the learned model. Note, that regardless of whether term dependency complex operators are deployed as a rewritten query, we include the score for the term dependency operators in the feature set. This allows the learner to consider the term dependence features separately from the score used in the initial ranking phase.

For learning and ranking, we use the Jforests implementation[8] of LambdaMART [8], a gradient-boosted decision tree learning-to-rank technique. Effectiveness experiments on the 197 TREC Web track queries (denoted WT in Table 4) use a 5-fold cross validation, where each fold has 60% training, 20% validation and 20% test queries. We report NDCG@20.

7 RESULTS

In the following, we report the results and analysis addressing our two research questions concerning the accuracy of our efficiency predictions for queries with complex operators (Section 7.1), and the application of the selective rewriting mechanism based upon those efficiency predictions (Section 7.2).

[8] https://github.com/yasserg/jforests/

Table 6: Pearson correlation on the test query set, per rewriting and K value, of the baseline (Base) and the proposed predictors (Pred). Statistically significant improvements over the baseline are denoted in bold.

Rewriting	K							
	20		100		1000		5000	
	Base	Pred	Base	Pred	Base	Pred	Base	Pred
NaïveMRF	0.639	**0.833**	0.722	**0.840**	0.827	**0.881**	0.861	0.868
MRF	0.612	**0.803**	0.697	**0.813**	0.788	**0.870**	0.790	**0.884**
Naïve	0.620	**0.848**	0.735	**0.878**	0.828	**0.915**	0.886	**0.935**
None	0.548	**0.738**	0.691	**0.842**	0.802	**0.907**	0.884	**0.952**

7.1 RQ1: Efficiency Prediction

We use the 66 efficiency prediction features derived from statistics, projectors and aggregators summarised in Table 3 to train a machine-learned model on the 978 training queries (see Table 4), for every combination of rewriting and K value. The regression algorithm employed is gradient boosted regression trees, as provided by the `scikit-learn` toolbox. Denoting the 978 train queries as Q^{tr}, we perform a 5-fold cross validation to train each $\langle Q_i^{\text{tr}}, K_j \rangle$ model. Each model was trained with 20 trees, learning rate of 0.1, max depth of 5, and the least square loss function. To evaluate the accuracy of each model, we then used the 978 test queries Q^{te} to compute the Pearson correlation, reported in Table 6. We compare each of our models (denoted Pred) with a linear regressor trained using the sum of number of postings per each simple term in the original terms and complex operators (Base). Our models perform very well, with correlations always higher than 0.8, and almost always significantly improving over the Base baseline predictor (according to a Fisher Z-transform, $p < 0.05$). To further assess the quality of our predictors, in Table 7, we compare the mean actual query times and the mean predicted query times for each model. Our models incur a mean prediction error no greater than 13%, even if they tend to underestimate the actual processing time, in particular for the models using the original queries (None).

Moreover, since we use the trained model to compare the predicted execution times of query plans versus a given time τ, in Table 8 we report the precision/recall measures of our models when classifying queries with a predicted execution time greater than 0.750 seconds. As can be observed from the table, the recall of our models is close or above 0.9 for NaïveMRF, MRF and Naïve rewritings. The smaller recall value for None is not problematic, since very few queries exhibit an execution time greater than 0.750 seconds.

Finally, we use the feature importance metric within gradient boosted trees to assess the contribution of the 66 prediction features. To combine the feature importances across various models, for each model we rank features by decreasing importance, and measure mean reciprocal ranks (Mean RR) across models. Table 9 shows the top features across all trained models (reported features attained Mean RR greater than 0.1). All proposed statistics and aggregators appear in the top features, with the exception of geometric mean (which appears, but it is not reported, as next in the list). Notably, also the minimum document frequency across all terms is in the list, validating the assumption that the minimum aggregator is useful when processing AND-based posting lists. Moreover, all projectors appear in the top features list, with the notable exception of #uw12. We explain this by observing that, according to Table 4, the corresponding complex operator occurs infrequently in the processed queries compared with the frequency of other complex operators.

Hence, in addressing RQ1, we have experimentally evaluated the accuracy of our predictions at estimating the execution times of rewritten queries processed using WAND for different K. Overall, with correlations > 0.8 for all tested rewritings and values of K, we conclude that our efficiency predictions are indeed accurate.

Table 7: Average actual and predicted query processing times on the test query set (in ms), per rewriting and K value, with relative errors (in %).

Rewriting	K							
	20		100		1000		5000	
Actual times								
NaïveMRF	2718		3272		4244		5090	
MRF	1438		1757		2618		3310	
Naïve	992		1373		1893		2381	
None	374		551		802		1042	
Predicted times								
NaïveMRF	2618	(−3.65)	3087	(−5.63)	4109	(−3.18)	4608	(−9.46)
MRF	1511	(5.12)	1762	(0.29)	2655	(1.44)	3101	(−6.31)
Naïve	953	(−3.92)	1261	(−8.16)	1866	(−1.42)	2339	(−1.78)
None	369	(−1.19)	485	(−12.13)	829	(−3.38)	1029	(−1.29)

Table 8: Precision/Recall accuracy to classify queries taking more than 750 milliseconds to process.

Rewriting	K							
	20		100		1000		5000	
	P	R	P	R	P	R	P	R
NaïveMRF	0.752	0.978	0.790	0.991	0.767	1.000	0.828	1.000
MRF	0.831	0.936	0.843	0.984	0.819	0.996	0.858	1.000
Naïve	0.769	0.908	0.813	0.935	0.843	0.979	0.867	0.998
None	0.800	0.520	0.839	0.719	0.836	0.881	0.783	0.963

Table 9: Top features ranked by the mean reciprocal rank of importance across all trained models.

Aggregator	Feature	Projector	Mean RR
Arithmetic mean	Document frequency	global	0.404
Maximum	Document frequency	#syn	0.350
Arithmetic mean	Score upper bound	global	0.306
Maximum	Document frequency	original	0.257
Maximum	Document frequency	#uw8	0.243
Maximum	Document frequency	global	0.215
Harmonic mean	Document frequency	global	0.185
–	Number of terms	global	0.171
Maximum	Document frequency	#1	0.152
Arithmetic mean	Document frequency	original	0.129
Minimum	Document frequency	global	0.109

7.2 RQ2: Selective Rewriting

In this section, we experiment to determine the levels of efficiency and effectiveness obtainable when using the selective mechanism proposed in Section 4, and using the predictors for complex operators evaluated in the preceding section. In terms of setup, our selective mechanism ranks the query plans by measuring $\mu = \text{NDCG}$ based on the validation set for each fold of the WT queries.

We consider the uniform application of ⟨None, $K = 5000$⟩ to all queries as the baseline that we compare to in terms of effectiveness. In particular, the use of $K = 5000$ for retrieving on ClueWeb09 has been used by various previous work on the same test collection [10, 28] and empirically verified in [27]. We denote the uniform application of this query plan as "Default". In our experiments, we aim to be more efficient than Default, while not experiencing a significant decrease in effectiveness. In addition to Default, we also report the efficiency and effectiveness of the uniform application of the Fastest, Slowest and Most Effective plans.

Table 10 provides the main findings of the efficiency and effectiveness of the uniform query plans. Note that, as discussed in Section 6, we use different query sets for measuring efficiency and effectiveness; in particular, we report mean and 95th percentile (or "tail") response times in milliseconds (MRT & TRT, respectively) on the 978 test queries from the MSN 2006 query log; for effectiveness, we report NDCG@20 for the 197 TREC Web track queries. The first group of rows reports efficiency and effectiveness for the Uniform plans, while the lower group reports the results for the Selection mechanism as threshold τ is varied. Finally, for each row, Rw denotes the percentage of queries rewritten from None.

On inspection of the uniform plan in the top half of Table 10, we note that the MRF and NaïveMRF uniform plans result in very high response times (up to 4.86 *times* slower than Default). In terms of effectiveness, deploying MRF to increase the recall in the sample results in a marked (but not statistically significant) increase in NDCG effectiveness of the system (0.1877→0.2001), however, this comes at the cost of retrieval that is 3.2 times slower than Default.

Next, we consider the selective mechanism in the bottom half of Table 10. As expected, as τ is varied we note changes in both effectiveness and efficiency. In particular, for $\tau = 500$, we find that a mean response time of 537 ms can be achieved (49% decrease in mean response time, and 62% decrease in tail response time, compared to Default), without significantly degrading effectiveness (0.1877 → 0.1751). For this setting, 8-10% of queries are being rewritten. With higher levels of τ, we observe similar increases in effectiveness and observed response times, and with more queries being rewritten. Finally, we note that the selection mechanism does not strictly observe the τ threshold, as can be observed by the tail response times – this is expected, as the selection mechanism expressed in Equation (3) will default to the predicted fastest plan available (usually ⟨None, $K = 20$⟩) if no plans can be executed within τ.

To provide a graphical illustration of the efficiency/effectiveness tradeoff, Figure 2 presents mean response times and mean NDCG. Lines are provided for both our selective mechanism, Full, denoted by a solid line, as well as the same selective mechanism where the candidate plans are restricted only to those involving None (i.e., with $K = \{20, 100, 1000, 5000\}$), denoted None-only, and indicated by a dashed line. For most thresholds, the Full selective mechanism can be observed to offer the best tradeoff, with points closer to the upper left hand corner. For low mean response times (e.g., < 400 ms), the None-only line rises above Full, as only None plans can be deployed to achieve such low response times. On the other hand, for larger mean response times (e.g., > 650 ms), the Full line is more effective; this supports a central tenet of our work, i.e., when time allows, appropriate rewriting of queries results in increased effectiveness. Moreover, as indicated by the ▽ points of the None-only

Table 10: Efficiency/effectiveness results using the selective mechanism. * denotes NDCG values that significantly differ from that of ⟨None, $K = 5000$⟩ (paired t-test, $p < 0.05$). TRT denotes the tail response time (95%-th percentile), Rw denotes the % of queries rewritten from None. Times are in ms.

Strategy	Efficiency (Test)			Effectiveness (WT)	
	MRT	TRT	Rw	NDCG@20	Rw
Uniform Plans					
Default (⟨None, $K = 5000$⟩)	1037	5281	0	0.1877	0
Fastest (⟨None, $K = 20$⟩)	376	1779	0	0.1375*	0
Slowest (⟨NaïveMRF, $K = 5000$⟩)	5045	17994	100	0.1755	100
Most Effective (⟨MRF, $K = 5000$⟩)	3281	12099	100	0.2001	100
Selective Mechanism					
$\tau = 200$	449	1986	1	0.1642*	0
$\tau = 300$	472	1986	1	0.1670*	1
$\tau = 400$	495	1986	4	0.1711*	6
$\tau = 500$	537	2004	8	0.1751	10
$\tau = 600$	577	2023	13	0.1762	14
$\tau = 700$	606	2060	19	0.1782	18
$\tau = 750$	617	2060	21	0.1779	19
$\tau = 800$	639	2128	22	0.1807	19
$\tau = 900$	691	2226	27	0.1825	21
$\tau = 1000$	733	2256	30	0.1828	21

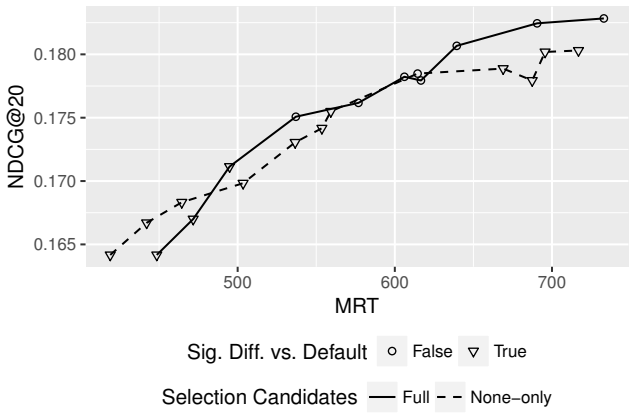

Figure 2: Efficiency/effectiveness tradeoff. The best tradeoff occurs for points closest to the upper left corner. Points denoted with ▽ are significantly less effective than ⟨None, $K = 5000$⟩ (paired t-test, $p < 0.05$).

line in Figure 2, to achieve the response times savings without selectively rewriting the query, we would be forced to accept significant degradations in effectiveness compared to the Default baseline.

Finally, to give a flavour of the impact of our selective mechanism, we inspect the queries rewritten for $\tau = 500$ and select the query 'disneyland hotel'. This query was rewritten to use the plan $\langle$MRF, $K = 100\rangle$ (as per the proximity example in Table 1), which had a predicted execution time of 486 ms, due to the relatively informative term disneyland (which only appears in $N_t = 78422$ documents). Hence, the more effective MRF rewrite was applied, which resulted in a 14% increase in NDCG@20 compared the Default plan (which had a predicted execution time of 542 ms).

Overall, in addressing RQ2 we have determined that our selective mechanism can achieve a 49% decrease in mean response time, and 62% decrease in tail (95th-percentile) response time without significant degradations in effectiveness.

8 CONCLUSIONS

This work has considered the selective rewriting of web search queries, with the aim of attaining efficient retrieval without hindering the system's effectiveness. In particular, we showed that it is possible to accurately measure the response time of a search engine in answering a query with complex operators such as #syn and #1, even when using the WAND dynamic pruning strategy. Our detailed experimental setup involved experiments upon the open TREC ClueWeb09 dataset, using TREC Web track queries for measuring effectiveness in terms of NDCG@20, and real search engine user queries for measuring efficiency. Moreover, we deployed three strategies to rewrite each query. Our experiments showed not only the accuracy of the newly proposed query efficiency predictions for queries involving complex operators, but also the ability of our proposed selective mechanism to enhance efficiency benefits (a 49% decrease in mean response time, and 62% decrease in tail, i.e., 95th-percentile, response time) without any significant degradation in mean NDCG@20. Overall, our results demonstrate that by selectively rewriting the query (when there is the time to execute the rewritten query), effectiveness can be at least maintained while markedly benefiting response times. Our proposed selective rewriting mechanism can be further extended, for instance to more query rewriting techniques, such as those based on query reformulation and query-click patterns [12, 20], and modelling the effectiveness of rewriting plans through risk rather than mean effectiveness alone.

REFERENCES

[1] Gianni Amati. 2006. Frequentist and Bayesian Approach to IR. In *ECIR*. 13–24.
[2] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. 2001. Vector-space ranking with effective early termination. In *SIGIR*. 35–42.
[3] Michael Bendersky, W. Bruce Croft, and Yanlei Diao. 2011. Quality-biased ranking of web documents. In *WSDM*. 95–104.
[4] Michael Bendersky, Donald Metzler, and W. Bruce Croft. 2010. Learning Concept Importance Using a Weighted Dependence Model. In *WSDM*. 31–40.
[5] Daniele Broccolo, Craig Macdonald, Salvatore Orlando, Iadh Ounis, Raffaele Perego, Fabrizio Silvestri, and Nicola Tonellotto. 2013. Load-sensitive Selective Pruning for Distributed Search. In *CIKM*. 379–388.
[6] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Y. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *CIKM*. 426–434.
[7] Chris Buckley, Gerard Salton, James Allan, and Amit Singhal. 1995. Automatic query expansion using SMART: TREC 3. In *TREC 3*. 69–80.
[8] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report MSR-TR-2010-82.
[9] Charles L. A. Clarke, Nick Craswell, and Ellen M. Voorhees. 2012. Overview of the TREC 2012 Web Track. In *TREC*.
[10] Nick Craswell, Dennis Fetterly, Marc Najork, Stephen Robertson, and Emine Yilmaz. 2010. Microsoft Research at TREC 2009. In *TREC*.
[11] Nick Craswell, Rosie Jones, Georges Dupret, and Evelyne Viegas (Eds.). 2009. *Proceedings of the Web Search Click Data Workshop at WSDM 2009*.
[12] Nick Craswell and Martin Szummer. 2007. Random walks on the click graph. In *SIGIR*. 239–246.
[13] W. Bruce Croft, Donald Metzler, and Trevor Strohman. 2009. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company.
[14] J. Shane Culpepper, Charles L. A. Clarke, and Jimmy Lin. 2016. Dynamic Cutoff Prediction in Multi-Stage Retrieval Systems. In *ADCS*. 17–24.
[15] Jeffrey Dean. 2009. Challenges in building large-scale information retrieval systems: invited talk. In *WSDM*.
[16] Jeffrey Dean and Luiz Andr Barroso. 2013. The Tail at Scale. *Commun. ACM* 56 (2013), 74–80.
[17] Shuai Ding and Torsten Suel. 2011. Faster top-k document retrieval using block-max indexes. In *SIGIR*. 993–1002.
[18] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Y. Zien. 2011. Evaluation Strategies for Top-k Queries over Memory-Resident Inverted Indexes. *PVLDB* 4, 12 (2011), 1213–1224.
[19] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L. Cox, and Scott Rixner. 2014. Predictive Parallelization: Taming Tail Latencies in Web Search. In *SIGIR*. 253–262.
[20] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating Query Substitutions. In *WWW*. 387–396.
[21] Saehoon Kim, Yuxiong He, Seung-won Hwang, Sameh Elnikety, and Seungjin Choi. 2015. Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search. In *WSDM*. 7–16.
[22] Nicholas Lester, Alistair Moffat, William Webber, and Justin Zobel. 2005. Space-Limited Ranked Query Evaluation Using Adaptive Pruning. In *WISE*. 470–477.
[23] Jimmy Lin, Matt Crane, Andrew Trotman, Jamie Callan, Ishan Chattopadhyaya, John Foley, Grant Ingersoll, Craig Macdonald, and Sebastiano Vigna. 2016. Toward Reproducible Baselines: The Open-Source IR Reproducibility Challenge. In *ECIR*. 408–420.
[24] Xiaolu Lu, Alistair Moffat, and J. Shane Culpepper. 2015. On the Cost of Extracting Proximity Features for Term-Dependency Models. In *CIKM*. 293–302.
[25] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. 2015. QuickScorer: A Fast Algorithm to Rank Documents with Additive Ensembles of Regression Trees. In *SIGIR*. 73–82.
[26] Craig Macdonald, Iadh Ounis, and Nicola Tonellotto. 2011. Upper-bound Approximations for Dynamic Pruning. *ACM Trans. Inf. Syst.* 29, 4 (2011), 17:1–17:28.
[27] Craig Macdonald, Rodrygo LT Santos, and Iadh Ounis. 2013. The whens and hows of learning to rank for web search. *Information Retrieval* 16, 5 (2013), 584–628.
[28] Craig Macdonald, Rodrygo L.T. Santos, Iadh Ounis, and Ben He. 2013. About Learning Models with Multiple Query-dependent Features . *ACM Trans. Inf. Syst.* 31, 3 (2013), 11:1–11:39.
[29] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2012. Learning to predict response times for online query scheduling. In *SIGIR*. 621–630.
[30] Donald Metzler and W. Bruce Croft. 2005. A Markov Random Field Model for Term Dependencies. In *SIGIR*. 472–479.
[31] Giuseppe Ottaviano, Nicola Tonellotto, and Rossano Venturini. 2015. Optimal Space-time Tradeoffs for Inverted Indexes. In *WSDM*. 47–56.
[32] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. 2006. Terrier: A High Performance & Scalable IR Platform. In *OSIR*.
[33] Jan Pederson. 2010. Query Understanding at Bing. In *Invited Talk, SIGIR 2010 Industry Day*.
[34] Fuchun Peng, Nawaaz Ahmed, Xin Li, and Yumao Lu. 2007. Context Sensitive Stemming for Web Search. In *SIGIR*. 639–646.
[35] Jie Peng, Craig Macdonald, Ben He, Vassilis Plachouras, and Iadh Ounis. 2007. Incorporating Term Dependency in the DFR Framework. In *SIGIR*. 843–844.
[36] Eric Shurman and Jake Brutlag. 2009. Performance related changes and their user impacts. In *Velocity: Web Performance and Operations Conference*.
[37] Trevor Strohman, Howard Turtle, and W. Bruce Croft. 2005. Optimization Strategies for Complex Queries. In *SIGIR*. 219–225.
[38] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. 2013. Efficient and Effective Retrieval Using Selective Pruning. In *WSDM*. 63–72.
[39] Howard Turtle and James Flood. 1995. Query evaluation: Strategies and optimizations. *Information Processing & Management* 31, 6 (1995), 831 – 850.
[40] Sebastiano Vigna. 2013. Quasi-succinct indices. In *WSDM*. 83–92.
[41] Lidan Wang, Jimmy Lin, and Donald Metzler. 2010. Learning to Efficiently Rank. In *SIGIR*. 138–145.
[42] Jinxi Xu and W. Bruce Croft. 1996. Query Expansion Using Local and Global Document Analysis. In *SIGIR*. 4–11.